TRANSACTIONS ON DATA PRIVACY 17 (2024) 89–121

Degree-Preserving Randomized Response for Graph Neural Networks under Local Differential Privacy

Seira Hidano*, **Takao Murakami****,***

*KDDI Research, Inc., 2-1-15 Ohara, Fujimino, Saitama, 356-8502, Japan.

**ISM, 10-3 Midori-cho, Tachikawa, Tokyo, 190-8562, Japan.

***AIST, 2-4-7 Aomi, Koto-ku, Tokyo, 135-0064, Japan.

E-mail: se-hidano@kddi.com, tmura@ism.ac.jp

Received 22 November 2023; received in revised form 27 March 2024; accepted 17 May 2024

Abstract. Differentially private GNNs (Graph Neural Networks) have been recently studied to provide high accuracy in various tasks on graph data while strongly protecting user privacy. In particular, a recent study proposes an algorithm to protect each user's feature vector in an attributed graph, which includes feature vectors along with node IDs and edges, with LDP (Local Differential Privacy), a strong privacy notion without a trusted third party. However, this algorithm does not protect edges (friendships) in a social graph, hence cannot protect user privacy in unattributed graphs, which include only node IDs and edges. How to provide strong privacy with high accuracy in unattributed graphs remains open. In this paper, we propose a novel LDP algorithm called the *DPRR (Degree-Preserving Randomized Response)* to provide LDP for edges in GNNs. Our DPRR preserves each user's degree hence a graph structure while providing edge LDP. Technically, our DPRR uses Warner's RR (Randomized Response) and *strategic* edge sampling, where each user's sampling probability is automatically tuned using the Laplacian mechanism to preserve the degree information under edge LDP. We also propose a privacy budget allocation method to make the noise in both Warner's RR and the Laplacian mechanism small. We focus on graph classification as a task of GNNs and evaluate the DPRR using three social graph datasets. Our experimental results show that the DPRR significantly outperforms three baselines and provides accuracy close to a non-private algorithm in all datasets with a reasonable privacy budget, e.g., $\varepsilon = 1$. Finally, we introduce data poisoning attacks to our DPRR and a defense against the attacks. We evaluate them using the three social graph datasets and discuss the experimental results.

Keywords. local differential privacy, graph neural networks, graph classification, randomized response, degree.

*The first and second authors made equal contribution.

1 Introduction

Many real-world data are represented as graphs, e.g., social networks, communication networks, and epidemiological networks. GNNs (Graph Neural Networks) [41] have recently attracted much attention because they provide state-of-the-art performance in various tasks on graph data, such as node classification [38], graph classification [63], and community detection [16]. However, the use of graph data raises serious privacy concerns, as it may reveal some sensitive data, such as sensitive edges (i.e., friendships in social graphs) [28].

DP (Differential Privacy) [20] has been widely studied to protect user privacy strongly and is recognized as a gold standard for data privacy. DP provides user privacy against adversaries with any background knowledge when a parameter called the privacy budget ε is small, e.g., $\varepsilon \leq 1$ or 2 [29, 39]. According to the underlying architecture, DP can be divided into two types: *centralized DP* and *LDP (Local DP)*. Centralized DP assumes a centralized model in which a trusted server has the personal data of all users and releases obfuscated versions of statistics or machine learning models. However, this model has a risk that the personal data of all users are leaked from the server by illegal access [43] or internal fraud [15]. In contrast, LDP assumes a local model where a user obfuscates her personal data by herself; i.e., it does not assume a trusted third party. Thus, LDP does not suffer from the data leakage issue explained above, and therefore it has been widely adopted in both the academic field [10, 12, 48, 57] and industry [19, 22].

DP has been recently applied to GNNs [34, 40, 44, 46, 52, 53, 60, 61, 70], and most of them adopt centralized DP. However, they suffer from the data breach issue explained above. Moreover, they cannot be applied to *decentralized SNSs (Social Networking Services)* [47] such as diaspora* [2] and Mastodon [5]. In decentralized SNSs, the entire graph is distributed across many servers, and each server has only the data of the users who choose it. Because centralized DP algorithms take the entire graph as input, they cannot be applied to the decentralized SNSs. We can also consider *fully decentralized SNSs*, where a server does not have any edge. For example, we can consider applications where each user sends noisy versions of her friends to the server, which then calculates some graph statistics [31, 32, 66] or generates synthetic graphs [49]. Centralized DP cannot be applied to these applications either.

LDP can be applied to these decentralized SNSs and does not suffer from data leakage. Sajadmanesh and Gatica-Perez [52] apply LDP to each user's feature vector (attribute values such as age, gender, city, and personal website) in GNNs. They focus on node classification and show that their algorithm provides high classification accuracy with a small privacy budget ε in LDP, e.g., $\varepsilon = 1$.

However, they focus on hiding feature vectors and do not hide edges (friendships). In practice, edges include highly sensitive information, i.e., sensitive friendships. In addition, many social graphs are *unattributed* graphs [64], which does not include feature vectors and include only node IDs and edges. Unfortunately, the algorithm in [52] cannot be used to protect user privacy in unattributed graphs.

To fill this gap, we focus on *LDP for edges in an unattributed graph*, i.e., DP in a scenario where each user obfuscates her edges (neighbor list or friend list) by herself. We also focus on *graph classification* as a task of GNNs (see Section 4.1 for details) because a state-of-the-art GNN [63] provides high accuracy for unattributed social graphs in this task. A recent study [71] also shows that a lot of private information can be inferred from the output of GNNs for graph classification. Therefore, LDP algorithms for edges in graph classification are urgently needed.

We first show that Warner's RR (Randomized Response) [58], which is widely used to

provide LDP for edges in applications other than GNNs [31, 32, 49, 66], is insufficient for GNNs. Specifically, the RR flips each 1 (edge) or 0 (no edge) with some probability. The flipping probability is close to 0.5 when ε is close to 0. Consequently, it makes a sparse graph dense and destroys a graph structure[1]. In particular, GNNs use a neighborhood aggregation (or message passing) strategy, which updates each node's feature vector by aggregating feature vectors of adjacent nodes. In an unattributed graph, a constant value is often used as a feature vector [23, 63], and in this case, the sum of adjacent feature vectors is a degree. Moreover, the degree distribution correlates with the graph type, as shown in our experiments. Thus, each user's degree is especially important in GNNs. The RR does not preserve the degree information and therefore does not provide high accuracy in GNNs for sparse graphs.

In addition, the RR does not provide high accuracy even in a *customized privacy setting* (as in Facebook [35]) where some users hide their neighbor lists and other users reveal their neighbor lists. We refer to the former users as *private users* and the latter as *non-private users*. The RR makes the neighbor lists of the private users dense. Consequently, it destroys the graph structure and ruins the neighborhood aggregation for non-private users. Thus, the accuracy is hardly increased with an increase in the non-private users, as shown in our experiments. Moreover, the RR has large time and space complexity and is impractical for large-scale graphs; e.g., the memory size is 1 TB in the Orkut social network [65] with three million users.

To address these issues, we propose a novel LDP algorithm, which we call the *DPRR (Degree-Preserving Randomized Response)*. Our DPRR uses edge sampling [13, 21, 32, 59] preceded with Warner's RR, and works as follows. First, it adds the Laplacian noise to each user's degree to provide edge LDP [49]. Then, it tunes the sampling probability based on the noisy degree. Finally, it applies Warner's RR and edge sampling so that each user's degree information is preserved under edge LDP. We use two privacy budgets, ε_1 and ε_2, for the Laplacian mechanism and Warner's RR, respectively. By the (general) sequential composition [39], our DPRR provides $(\varepsilon_1 + \varepsilon_2)$-edge LDP.

Since our DPRR preserves each user's degree information, it is suitable for the neighborhood aggregation strategy in GNNs. It also works very well in the customized privacy setting – the accuracy is rapidly increased with an increase in non-private users. We show that our DPRR significantly outperforms Warner's RR, especially in the customized setting. Moreover, we show that the DPRR is much more efficient than the RR in that the DPRR needs much less time for both training and classification and much less memory; e.g., the memory size is about 30 MB even in the Orkut social network explained above.

We also compare our DPRR with two other private baselines: (i) a local model version of LAPGRAPH (Laplace Mechanism for Graphs) [61] and (ii) an algorithm that discards neighbor lists of private users and uses a graph composed of only non-private users. We denote the former and latter baselines by LocalLap and NonPriv-Part, respectively. The latter baseline, NonPriv-Part, is an algorithm that has not been studied in the literature. NonPriv-Part uses only the information of non-private users. We introduce this baseline to show the effectiveness of using the information of both private and non-private users. We show that the DPRR significantly outperforms LocalLap and NonPriv-Part and provides high accuracy with a reasonable privacy budget, e.g., $\varepsilon = 1$.

Note that we assume an *honest-but-curious* setting, as with most of the existing work on LDP (e.g., [10, 12, 22, 31, 32, 48, 49, 57, 66]). That is, we assume that each user honestly

[1]Note that some studies [32, 54] propose a variant of Warner's RR that makes a graph sparse. We explain the difference between our DPRR and [32, 54] at the end of Section 1.

applies an LDP algorithm to her neighbor list and sends the noisy neighbor lists to the data collector. However, recent studies [14, 17] show that LDP algorithms are vulnerable to *data poisoning attacks*, which inject malicious user accounts and send fake data from these accounts to degrade the accuracy of statistical analysis results or machine learning models. Therefore, we finally evaluate the robustness of our DPRR against data poisoning attacks. Specifically, we introduce data poisoning attacks to our DPRR and a defense against the attacks. We evaluate them through experiments and discuss the results.

Our Contributions. Our contributions are as follows:

- We propose the DPRR for GNNs under LDP for edges. Technically, we use edge sampling [13, 21, 32, 59] after Warner's RR. In particular, our main technical novelty lies in what we call *strategic* edge sampling, where each user's sampling probability is automatically tuned using the Laplacian mechanism to preserve the degree information under edge LDP (see Section 4.3).
- As explained above, the DPRR divides the privacy budget ε into two budgets, ε_1 and ε_2 ($\varepsilon = \varepsilon_1 + \varepsilon_2$). ε_1 is for the Laplacian mechanism, whereas ε_2 is for Warner's RR. We also propose a privacy budget allocation method to make the noise in both of the mechanisms small and thereby provide high accuracy in GNNs (see Section 4.4).
- We prove that our DPRR approximately preserves the degree information under edge LDP. We also show that our DPRR has much smaller time and space complexity than Warner's RR (see Sections 4.5 and 4.6).
- We focus on graph classification and evaluate our DPRR using three social graph datasets. We show that our DPRR outperforms three private baselines (RR, LocalLap, and NonPriv-Part) in terms of accuracy and the RR in terms of efficiency. We also compare the DPRR with a fully non-private algorithm that does not add any noise for all users, including private users (denoted by NonPriv-Full). For all datasets, we show that our DPRR provides accuracy close to NonPriv-Full (e.g., the difference in the classification accuracy or AUC is smaller than 0.1 or so) with a reasonable privacy budget, e.g., $\varepsilon = 1$ (see Section 5).
- We finally introduce data poisoning attacks to our DPRR and a defense against the attacks. We evaluate them using the three datasets and discuss the results (see Section 6).

Our code is available on GitHub [8].

Technical Novelty. As described above, the technical novelty of our DPRR is two-fold: (i) strategic edge sampling that tunes each user's sampling probability to preserve the degree information under edge LDP and (ii) privacy budget allocation.

Regarding the first point, Imola *et al.* [32] propose the ARR (Asymmetric RR) [32], which uses edge sampling after Warner's RR. Edge sampling has been widely studied to improve the scalability in counting triangles in a graph [13, 21, 59]. The authors in [32] use edge sampling to improve the communication efficiency in triangle counting under LDP. However, they use a sampling probability common to all users and manually set the sampling probability. In this case, the accuracy is not improved by the sampling, because the graph structure remains destroyed. In statistics, random sampling is used to improve efficiency at the expense of accuracy. In fact, edge sampling in [32] decreases the accuracy of triangle counting. In contrast, our DPRR provides higher accuracy than Warner's RR because it automatically tunes each user's sampling probability to preserve the degree information.

In other words, our DPRR is different from the ARR [32] (and other existing edge sampling algorithms [13, 21, 59]) in that ours is a strategic sampling algorithm to improve *both accuracy and efficiency* in GNNs by preserving the graph structure under edge LDP. The novelty of our strategic sampling technique is not limited to the privacy literature – there are no sampling techniques to improve both the accuracy and efficiency, even in (non-private) triangle counting or graph machine learning, to our knowledge.

Salas *et al.* [54] propose a noise-graph mechanism that changes 1 (edge) to 0 (no edge) with probability $1 - p_1$ and 0 to 1 with probability $1 - p_0$, where $p_1, p_0 \in [0, 1]$. This mechanism is equivalent to the ARR in [32] when $p_1 \leq p_0$. More specifically, let $p, q \in [0, 1]$ be the parameters of Warner's RR and edge sampling, respectively, in [32]; Warner's RR flips $1/0$ with probability $1-p$, and edge sampling changes 1 to 0 with probability $1-q$. Then, $p_1 = pq$ and $p_0 = 1-(1-p)q$. Thus, as with [32], the noise-graph mechanism in [54] uses a sampling probability common to all users and manually sets the sampling probability. Consequently, it does not improve the accuracy of Warner's RR. In contrast, our DPRR automatically tunes each user's sampling probability to improve both accuracy and efficiency.

In summary, our DPRR is new in that it adopts strategic edge sampling to improve both accuracy and efficiency. In addition, our privacy budget allocation method that makes the noise in the Laplacian mechanism and Warner's RR small is also new. We show that they are effective and outperform three baselines in terms of accuracy and efficiency.

2 Related Work

DP on GNNs. In the past year or two, differentially private GNNs [34, 40, 44, 46, 52, 53, 60, 61, 70] (or graph data synthesis [33, 69]) have become a very active research topic. Most of them assume a centralized model [33, 44, 46, 53, 61, 69, 70] where the server has the entire graph (or the exact number of edges in the entire graph [61]). They suffer from the data leakage issue and cannot be applied to decentralized (or fully decentralized) SNSs, as explained in Section 1.

Some studies [34, 60] focus on GNNs under LDP in different settings than ours. Specifically, Jin and Chen [34] assume that each user has a graph and apply an LDP algorithm to a graph embedding calculated by each user. In contrast, we consider a totally different scenario, where each user is a node in a graph and the server does not have an edge. Thus, the algorithm in [34] cannot be applied to our setting. Wu *et al.* [60] assume that each user has a user-item graph and propose a federated GNN learning algorithm for item recommendation. In their work, an edge represents that a user has rated an item. Therefore, their algorithm cannot be applied to our setting where a node and edge represent a user and friendship, respectively. We also note that federated (or collaborative) learning generally requires many interactions between users and the server [37, 56]. In contrast, our DPRR requires only *one-round* interaction.

Sajadmanesh and Gatica-Perez [52] apply LDP to each user's feature vectors in an attributed graph. However, they do not hide edges, which are sensitive in a social graph. Consequently, their algorithm cannot be applied to unattributed graphs, where GNNs provide state-of-the-art performance [63].

To our knowledge, a recent work [40] is the only one that attempts to hide edges in GNNs under LDP. However, the authors in [40] fail to provide a better algorithm than Warner's RR in unattributed graphs. Specifically, they propose to denoise a noisy adjacency matrix after applying Warner's RR by minimizing the l_1 norm regularized least-squares of the denoised adjacency matrix. However, when we consider a binary adjacency matrix, the

optimal solution is *either a noisy adjacency matrix obtained by Warner's RR or a zero matrix* (i.e., trivial solution)[2] and does not improve Warner's RR.

We also note that LAPGRAPH [61], which assumes the centralized model, can be modified so that it works in the local model setting, as shown in this paper. However, the local model version of LAPGRAPH suffers from low accuracy, as it does not preserve each user's degree information. In Section 5, we show that our DPRR significantly outperforms both Warner's RR and the local model version of LAPGRAPH.

LDP. LDP has been widely studied for tabular data where each row corresponds to a user. The main task in this setting is statistical analysis, such as distribution estimation [10, 22, 57] and heavy hitter estimation [12, 48].

LDP has also been used for graph applications other than GNNs, e.g., calculating subgraph counts [31, 32], estimating graph metrics [66], and generating synthetic graphs [49]. Qin *et al.* [49] propose a synthetic graph data generation technique called LDPGen, which requires two-round interaction between users and the server. However, two-round interaction is impractical for many practical scenarios, as it needs a lot of user effort and synchronization – the server must wait for all users' responses in each round. This is prohibitively time-consuming when the number of users is large. Thus, we focus on algorithms based on *one-round* interaction between users and the server. Our DPRR is a one-round algorithm and is much more practical than LDPGen [49].

Qin *et al.* [49] also propose a one-round algorithm that applies Warner's RR to each edge. Similarly, the studies in [31, 32, 66] use Warner's RR to calculate subgraph counts or graph metrics[3]. Since Warner's RR can also be applied to GNNs, we use it as a baseline. As described in Section 1, Warner's RR makes a sparse graph dense and destroys the graph structure. Consequently, it does not provide high accuracy in GNNs, as shown in our experiments. The study in [45] reduces the number of 1s (edges) in Warner's RR by sampling without replacement. However, their proof of DP relies on the independence of each edge and is incorrect, as pointed out in [32].

Note that for categorical data of large domain size $k \gg 2$, the RR is outperformed by other LDP algorithms, such as OLH (Optimized Local Hashing) [57], OUE (Optimized Unary Encoding) [57], and HR (Hadamard Response) [10]. However, it is proved in [9] that they are not better than the RR in binary domains ($k = 2$). Moreover, we consider a setting where both the input domain and the output range are binary (i.e., "edge" or "no edge") to make LDP mechanisms applicable to GNNs. In this setting (i.e., when output data are compressed to binary bits), all of the OLH, OUE, and HR are identical to Warner's RR, as they are symmetric. We also note that applying them to an entire neighbor list ($k = 2^n$) results in prohibitively large noise, as k is too large. Therefore, Warner's RR for each bit of the neighbor list has been widely used in graphs [31, 32, 49, 66]. We also use Warner's RR as a building block.

[2] Their optimization problem is $\min_{\mathbf{A}} ||\tilde{\mathbf{A}} - \mathbf{A}||_F^2 + \eta||\mathbf{A}||_1$, where $\mathbf{A}$ is a binary adjacency matrix, $\tilde{\mathbf{A}}$ is a noisy adjacency matrix after Warner's RR, and η is a real number. The optimal solution $\mathbf{A}^*$ to this problem is: $\mathbf{A}^* = \tilde{\mathbf{A}}$ if $\eta \leq 1$ and $\mathbf{A}^* = \mathbf{0}$ (zero matrix) otherwise.

[3] The study in [66] also proposes an algorithm to estimate the clustering coefficient based on Warner's RR, claiming that the clustering coefficient is useful for generating a synthetic graph based on the graph model BTER (Block Two-Level Erdös-Rényi) [55]. However, this claim is incorrect – BTER does not use the clustering coefficient. Moreover, BTER has two parameters ρ and η, which are determined by manual experimentation to fit the original graph (see Section IV in [55]). It is unclear how to automatically determine them.

Table 1: Basic notation in this paper.

Symbol	Description
n	Number of users.
$\mathcal{G}$	Set of possible graphs.
$G = (V, E)$	Graph with users V and edges E.
v_i	i-th user.
$\mathbf{A} = (a_{i,j})$	Adjacency matrix.
$\mathbf{a}_i$	Neighbor list of user v_i.
d_i	Degree of user v_i.
$\mathcal{N}(v_i)$	Set of nodes adjacent to user v_i.
$\mathcal{R}_i$	Local randomizer of user v_i.

3 Preliminaries

In this section, we describe some preliminaries needed for this paper. Section 3.1 defines basic notation. Section 3.2 reviews LDP on graphs. Section 3.3 explains GNNs.

3.1 Basic Notation

Let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{N}$, $\mathbb{Z}_{\geq 0}$ be the sets of real numbers, non-negative real numbers, natural numbers, and non-negative integers, respectively. For $a \in \mathbb{N}$, let $[a] = \{1, 2, \cdots, a\}$.

Consider an unattributed social graph with $n \in \mathbb{N}$ nodes (users). Let $\mathcal{G}$ be the set of possible graphs with a finite number of nodes, and $G = (V, E) \in \mathcal{G}$ be a graph with a set of nodes $V = \{v_1, v_2, \cdots, v_n\}$ and a set of edges $E \subseteq V \times V$. The graph G can be either directed or undirected. In a directed graph, an edge $(v_i, v_j) \in E$ represents that user v_i follows user v_j. In an undirected graph, an edge (v_i, v_j) represents that v_i is a friend with v_j.

A graph G can be represented as an adjacency matrix $\mathbf{A} = (a_{i,j}) \in \{0, 1\}^{n \times n}$, where $a_{i,j} = 1$ if and only if $(v_i, v_j) \in E$. Note that the diagonal elements are always 0; i.e., $a_{1,1} = \cdots = a_{n,n} = 0$. If G is an undirected graph, $\mathbf{A}$ is symmetric. Let $\mathbf{a}_i = (a_{i,1}, a_{i,2}, \cdots, a_{i,n}) \in \{0, 1\}^n$ be the i-th row of $\mathbf{A}$. $\mathbf{a}_i$ is called the *neighbor list* [49] of user v_i. Let $d_i \in \mathbb{Z}_{\geq 0}$ be the degree of user v_i. Note that $d_i = ||\mathbf{a}_i||_1$, i.e., the number of 1s in $\mathbf{a}_i$. Let $\mathcal{N}(v_i)$ be the set of nodes adjacent to v_i, i.e., $v_j \in \mathcal{N}(v_i)$ if and only if $(v_i, v_j) \in E$.

We focus on a local privacy model [32, 31, 66, 49], where user v_i obfuscates her neighbor list $\mathbf{a}_i$ using a *local randomizer* $\mathcal{R}_i$ and sends the obfuscated data $\mathcal{R}_i(\mathbf{a}_i)$ to a server. Table 1 shows the basic notation used in this paper.

3.2 Local Differential Privacy on Graphs

Edge LDP. The local randomizer $\mathcal{R}_i$ of user v_i adds some noise to her neighbor list $\mathbf{a}_i$ to hide her edges. Here, we assume that the server and other users can be honest-but-curious adversaries and can obtain all edges in G other than edges of v_i as background knowledge. To strongly protect edges of v_i from these adversaries, we use DP as a privacy metric. In graphs, there are two types of DP: *node DP* and *edge DP* [50]. Node DP hides one node along with its edges from the adversary. However, many applications in the local privacy model require a user to send her user ID, and we cannot use node DP for these

applications. Therefore, we use edge DP in the same way as the previous work on graph LDP [31, 32, 49, 66].

Edge DP hides one edge between any two users from the adversary. Its local privacy model version, called *edge LDP* [49], is defined as follows:

Definition 1 (ε-edge LDP [49])**.** Let $\varepsilon \in \mathbb{R}_{\geq 0}$ and $i \in [n]$. A local randomizer $\mathcal{R}_i$ of user v_i with domain $\{0,1\}^n$ provides *ε-edge LDP* if and only if for any two neighbor lists $\mathbf{a}_i, \mathbf{a}'_i \in \{0,1\}^n$ that differ in one bit and any $s \in \mathrm{Range}(\mathcal{R}_i)$,

$$\Pr[\mathcal{R}_i(\mathbf{a}_i) = s] \leq e^{\varepsilon} \Pr[\mathcal{R}_i(\mathbf{a}'_i) = s]. \tag{1}$$

For example, the randomized neighbor list in [49] applies Warner's RR (Randomized Response) [58], which flips $0/1$ with probability $\frac{1}{e^{\varepsilon}+1}$, to each bit of $\mathbf{a}_i$ (except for $a_{i,i}$). By (1), this randomizer provides ε-edge LDP.

The parameter ε is called the privacy budget, and the value of ε is crucial in DP. By (1), the likelihood of $\mathbf{a}_i$ is almost the same as that of $\mathbf{a}'_i$ when ε is close to 0. However, they can be very different when ε is large. For example, it is well known that $\varepsilon \leq 1$ or 2 is acceptable for many practical scenarios, whereas $\varepsilon \geq 5$ is unsuitable in most scenarios [29, 39]. Based on this, we set $\varepsilon \leq 2$ in our experiments.

Edge LDP can be used to hide each user's neighbor list $\mathbf{a}_i$. For example, in Facebook, user v_i can change her setting so that anyone except for server administrators cannot see her neighbor list $\mathbf{a}_i$. By using edge LDP with small ε, we can hide $\mathbf{a}_i$ even from the server administrators.

Relationship DP. In an undirected graph, each edge (v_i, v_j) is shared by two users v_i and v_j. Thus, both users' outputs can leak the information about (v_i, v_j). Imola *et al.* [31] define *relationship DP* to protect each edge during the whole process:

Definition 2 (ε-relationship DP [31])**.** Let $\varepsilon \in \mathbb{R}_{\geq 0}$. A tuple of local randomizers $(\mathcal{R}_1, \cdots, \mathcal{R}_n)$ provides *ε-relationship DP* if and only if for any two undirected graphs $G, G' \in \mathcal{G}$ that differ in one edge and any $(s_1, \ldots, s_n) \in \mathrm{Range}(\mathcal{R}_1) \times \ldots \times \mathrm{Range}(\mathcal{R}_n)$,

$$\begin{aligned} &\Pr[(\mathcal{R}_1(\mathbf{a}_1), \ldots, \mathcal{R}_n(\mathbf{a}_n)) = (s_1, \ldots, s_n)] \\ &\leq e^{\varepsilon} \Pr[(\mathcal{R}_1(\mathbf{a}'_1), \ldots, \mathcal{R}_n(\mathbf{a}'_n)) = (s_1, \ldots, s_n)], \end{aligned} \tag{2}$$

where $\mathbf{a}_i$ (resp. $\mathbf{a}'_i$) is the i-th row of the adjacency matrix of G (resp. G').

Proposition 3 (Edge LDP and relationship DP [31])**.** *In an undirected graph, if each of local randomizers $\mathcal{R}_1, \ldots, \mathcal{R}_n$ provides ε-edge LDP, then $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ provides 2ε-relationship DP.*

In Proposition 3, relationship DP has the doubling factor in ε because the presence/absence of one edge (v_i, v_j) affects two bits $a_{i,j}$ and $a_{j,i}$ in neighbor lists in an undirected graph.

Note that even if user v_i hides her neighbor list $\mathbf{a}_i$, her edge with her friend v_j will be disclosed when v_j releases $\mathbf{a}_j$. This is inevitable in social networks based on undirected graphs. For example, in Facebook, user v_i can change her setting so that no one can see $\mathbf{a}_i$. However, she can also change the setting so that $\mathbf{a}_i$ is public. Thus, if user v_i hides $\mathbf{a}_i$ and her friend v_j reveals $\mathbf{a}_j$, their edge (v_i, v_j) is disclosed. To prevent this, v_i needs to ask v_j not to reveal $\mathbf{a}_j$.

In other words, relationship DP requires some trust assumptions, unlike LDP; i.e., if a user wants to keep all her edges secret, she needs to trust her friends not to reveal their neighbor lists. However, even if her $k \in \mathbb{N}$ friends reveal their neighbor lists, only her k edges will

be disclosed. Therefore, the trust assumption of relationship DP is much weaker than that of centralized DP, where the server can leak all edges.

Global Sensitivity. As explained above, Warner's RR is one of the simplest approaches to providing edge LDP. Another well-known approach is to use global sensitivity [20]:

Definition 4. In edge LDP, the global sensitivity GS_f of a function $f : \{0,1\}^n \to \mathbb{R}$ is given by

$$GS_f = \max_{\mathbf{a}_i, \mathbf{a}'_i \in \{0,1\}^n, \mathbf{a}_i \sim \mathbf{a}'_i} |f(\mathbf{a}_i) - f(\mathbf{a}'_i)|,$$

where $\mathbf{a}_i \sim \mathbf{a}'_i$ represents that $\mathbf{a}_i$ and $\mathbf{a}'_i$ differ in one bit.

For $b \in \mathbb{R}_{\geq 0}$, let $\mathrm{Lap}(b)$ be the Laplacian noise with mean 0 and scale b. Then, adding the Laplacian noise $\mathrm{Lap}(\frac{GS_f}{\varepsilon})$ to f provides ε-edge LDP.

3.3 Graph Neural Networks

Graph Classification. We focus on graph classification as a task of GNNs. The goal of graph classification is to predict a label of the entire graph, e.g., type of community, type of online discussion (i.e., subreddit [6]), and music genre users in the graph are interested in.

More specifically, we are given multiple graphs, some of which have a label. Let $\mathcal{G}_l = \{G_1, \cdots, G_{|\mathcal{G}_l|}\} \subseteq \mathcal{G}$ be the set of labeled graphs, and $\mathcal{G}_u = \{G_{|\mathcal{G}_l|+1}, \cdots, G_{|\mathcal{G}_l|+|\mathcal{G}_u|}\} \subseteq \mathcal{G}$ be the set of unlabeled graphs. Graph classification is a task that finds a mapping function ϕ that takes a graph $G \in \mathcal{G}$ as input and outputs a label $\phi(G)$. We train a mapping function ϕ from labeled graphs $\mathcal{G}_l$. Then, we can predict labels for unlabeled graphs using the trained function ϕ.

The GNN is a machine learning model useful for graph classification. Given a graph $G \in \mathcal{G}$, the GNN calculates a feature vector h_G of the entire graph G. Then it predicts a label based on h_G, e.g., by a softmax layer.

Neighborhood Aggregation. Most GNNs use a neighborhood aggregation (or message passing) strategy, which updates a node feature of each node v_i by aggregating node features of adjacent nodes $\mathcal{N}(v_i)$ and combining them [24, 41]. Specifically, each layer in a GNN has an aggregate function and a combine (or update) function. For $k \in \mathbb{Z}_{\geq 0}$, let $h_i^{(k)}$ be a feature vector of v_i at the k-th layer. Since we focus on an unattributed graph G, we create the initial feature vector $h_i^{(0)}$ from the graph structure, e.g., a one-hot encoding of the degree of v_i or a constant value [23, 63].

At the k-th layer, we calculate $h_i^{(k)}$ as follows:

$$m_i^{(k)} = \mathsf{AGGREGATE}^{(k)}(\{h_j^{(k-1)} : v_j \in \mathcal{N}(v_i)\}) \tag{3}$$

$$h_i^{(k)} = \mathsf{COMBINE}^{(k)}(h_i^{(k-1)}, m_i^{(k)}). \tag{4}$$

$\mathsf{AGGREGATE}^{(k)}$ is an aggregate function that takes feature vectors $h_j^{(k-1)}$ of adjacent nodes $\mathcal{N}(v_i)$ as input and outputs a message $m_i^{(k)}$ for user v_i. Examples of $\mathsf{AGGREGATE}^{(k)}$ include a sum [63], mean [26, 38], and max [26]. $\mathsf{COMBINE}^{(k)}$ is a combine function that takes a feature vector $h_i^{(k-1)}$ of v_i and the message $m_i^{(k)}$ as input and outputs a new feature vector $h_i^{(k)}$. Examples of $\mathsf{COMBINE}^{(k)}$ include one-layer perceptrons [26] and MLPs (Multi-Layer Perceptrons) [63].

Note that (3) uses the set $\mathcal{N}(v_i)$ of nodes adjacent to v_i and therefore can leak the edge information. Since the algorithm in [52] hides only node features, it reveals edge information in (3) and violates edge LDP. In other words, the algorithm in [52] cannot be used to protect privacy in unattributed graphs.

For graph classification, the GNN outputs a feature vector h_G of the entire graph. h_G is calculated by aggregating feature vectors $h_i^{(K)}$ of the final iteration $K \in \mathbb{N}$ as follows:

$$h_G = \mathsf{READOUT}(\{h_i^{(K)} : v_i \in V\}). \tag{5}$$

READOUT is a readout function, such as a sum [63], mean [63], and hierarchical graph pooling [68].

4 Degree-Preserving Randomized Response

We propose a local randomizer that provides LDP for edges in the original graph while keeping high classification accuracy in GNNs. A simple way to provide LDP for edges is to use Warner's RR (Randomized Response) [58] to each bit of a neighbor list. However, it suffers from low classification accuracy because the RR makes a sparse graph dense and destroys a graph structure, as explained in Section 1. To address this issue, we propose a local randomizer called the *DPRR (Degree-Preserving Randomized Response)*, which provides edge LDP while preserving each user's degree.

Section 4.1 explains a system model assumed in our work. Section 4.2 describes the overview of our DPRR. Section 4.3 explains our DPRR in detail. Section 4.4 proposes a privacy budget allocation method for our DPRR. Section 4.5 shows the degree preservation properties of our DPRR. Finally, Section 4.6 shows the time and space complexity of our DPRR.

4.1 System Model

Figure 1 shows a system model in this paper. First, we assume that there are multiple social graphs, some of which have a label. We can consider several practical scenarios for this.

For example, some social networks (e.g., Reddit [6]) provide online discussion threads. In this case, each discussion thread can be represented as a graph, where an edge represents that a conversation happens between two users, and a label represents a category (e.g., subreddit) of the thread. The REDDIT-MULTI-5K and REDDIT-BINARY [64] are real datasets in this scenario and are used in our experiments.

For another example, we can consider multiple decentralized SNSs, as described in Section 1. Some SNSs may have a topic, such as music, game, food, and business [4, 5, 7]. In this scenario, each SNS corresponds to a graph, and a label represents a topic.

Based on the labeled and unlabeled social graphs, we perform graph classification privately. Specifically, for each graph $G = (V, E)$, each user $v_i \in V$ obfuscates her neighbor list $\mathbf{a}_i \in \{0,1\}^n$ using a local randomizer $\mathcal{R}_i$ providing ε_i-edge LDP and sends her noisy neighbor lists $\tilde{\mathbf{a}}_i \in \{0,1\}^n$ to a server. Then, the server calculates a noisy adjacency matrix $\tilde{\mathbf{A}} \in \{0,1\}^{n \times n}$ corresponding to $\tilde{\mathbf{a}}_1, \cdots, \tilde{\mathbf{a}}_n$. The server trains the GNN using noisy adjacency matrices $\tilde{\mathbf{A}}$ of labeled graphs. Then, the server predicts a label for each unlabeled graph based on its noisy adjacency matrix $\tilde{\mathbf{A}}$ and the trained GNN.

Note that we consider a personalized setting [36], where each user v_i can set her privacy budget ε_i. In particular, we consider the following two basic settings:

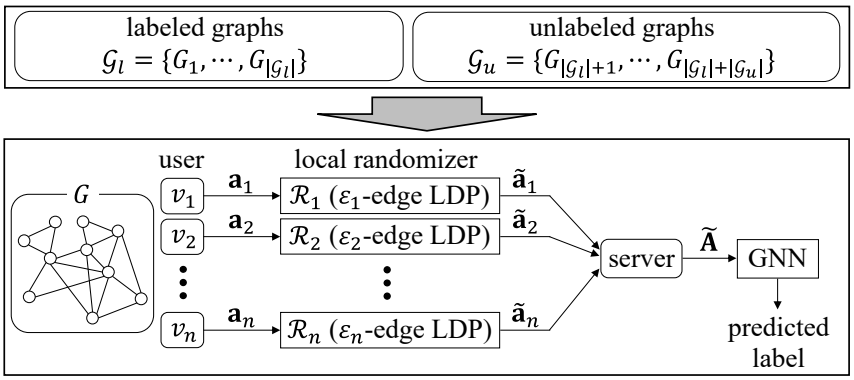


Figure 1: System model. For each graph G, users $v_1, \cdots, v_n$ send their noisy neighbor lists $\tilde{\mathbf{a}}_1, \cdots, \tilde{\mathbf{a}}_n$ providing edge LDP. Then, the server calculates a noisy adjacency matrix $\tilde{\mathbf{A}}$ corresponding to $\tilde{\mathbf{a}}_1, \cdots, \tilde{\mathbf{a}}_n$. The server trains the GNN using matrices $\tilde{\mathbf{A}}$ of labeled graphs and predicts a label for each unlabeled graph using its matrix $\tilde{\mathbf{A}}$ and the trained GNN.

- **Common Setting.** In this setting, all users adopt the same privacy budget ε, i.e., $\varepsilon = \varepsilon_1 = \varepsilon_2 = \cdots = \varepsilon_n$. This is a scenario assumed in most studies on DP. By Proposition 3, a tuple of local randomizers $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ provides 2ε-relationship DP in the common setting when the graph G is undirected.
- **Customized Setting.** In this setting, some private users adopt a small privacy budget (e.g., $\varepsilon = 1$) and other non-private users make their neighbor lists public (i.e., $\varepsilon = \infty$). This is similar to Facebook's setting. Specifically, in Facebook, each user v_i can change her setting so that no one (except for the server) can see $\mathbf{a}_i$. User v_i can also change her setting so that $\mathbf{a}_i$ is public. Our customized setting is stronger than Facebook's setting in that a private user v_i hides $\mathbf{a}_i$ even from the server.

Recall that in an undirected graph, even if v_i hides her neighbor list $\mathbf{a}_i$, her edge with v_j can be disclosed when v_j makes $\mathbf{a}_j$ public. In other words, a tuple of local randomizers $(\mathcal{R}_1, \ldots, \mathcal{R}_n)$ does not provide relationship DP in the customized setting where one or more users are non-private.

However, the customized setting still makes sense because it is a stronger version of Facebook's setting; i.e., our customized setting hides the neighbor list $\mathbf{a}_i$ of the private user v_i even from the server. In our customized setting, even if $k \in \mathbb{N}$ friends are non-private, only k edges with them will be disclosed, as described in Section 3.2. All edges between private users will be kept secret, even from the server.

In our experiments, we show that our DPRR is effective in both common and customized settings.

4.2 Algorithm Overview

Figure 2 shows the overview of our DPRR, which takes a neighbor list $\mathbf{a}_i \in \{0,1\}^n$ of user v_i as input and outputs a noisy neighbor list $\tilde{\mathbf{a}}_i \in \{0,1\}^n$. We also show the details of the RR and edge sampling in Figure 3.

As shown in Figures 2 and 3, we use edge sampling after Warner's RR to avoid a dense noisy graph. For each bit of the neighbor list $\mathbf{a}_i$, Warner's RR outputs $0/1$ as is with probability $p \in [0,1]$ and flips $0/1$ with probability $1-p$. Then, for each 1 (edge), edge sampling

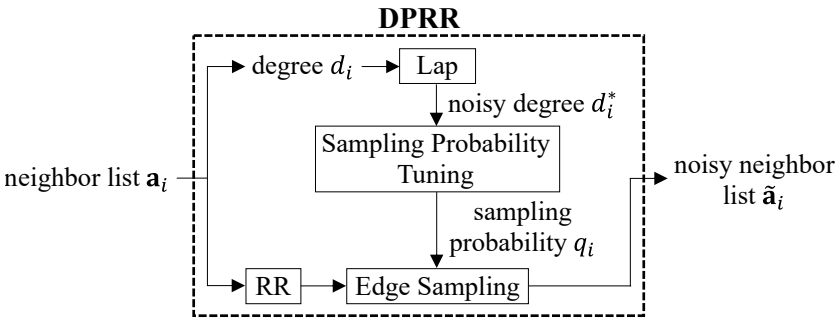


Figure 2: Overview of the DPRR (Degree-Preserving Randomized Response).

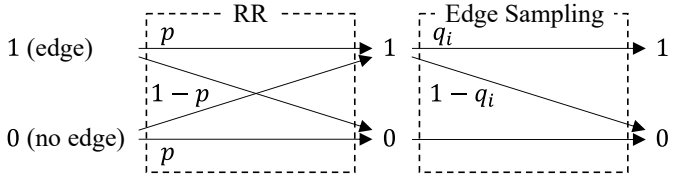


Figure 3: RR and edge sampling.

outputs 1 with probability $q_i \in [0, 1]$ and 0 with probability $1 - q_i$. Here, the degree information of each user is especially important in GNNs because it represents the number of adjacent nodes in the aggregate step. Thus, we carefully tune the sampling probability q_i of each user v_i so that *the number of 1s in* $\tilde{\mathbf{a}}_i$ *is close to the original degree* d_i $(= ||\mathbf{a}_i||_1)$.

Note that we cannot use d_i itself to tune the sampling probability, because d_i leaks the information about edges of v_i. To address this issue, we tune the sampling probability by replacing d_i with a *private estimate of* d_i providing edge LDP. Note that the private estimate of d_i is calculated locally, and therefore it provides "local" DP. Both the RR and the private estimate of d_i provide edge LDP. Thus, by the (general) sequential composition [39], the noisy neighbor list $\tilde{\mathbf{a}}_i$ is also protected with edge LDP.

Specifically, the DPRR works as follows. We first calculate a degree d_i from the neighbor list $\mathbf{a}_i$ and add the Laplacian noise to the degree d_i[4]. Consequently, we obtain a private estimate $d_i^* \in \mathbb{R}$ of d_i with edge LDP. Then, we tune the sampling probability q_i using d_i^* so that the expected number of 1s in $\tilde{\mathbf{a}}_i$ is equal to d_i. Finally, we apply Warner's RR to $\mathbf{a}_i$ and then randomly sample 1s with the sampling probability q_i. As a result, we obtain the noisy neighbor list $\tilde{\mathbf{a}}_i$ whose noisy degree $\tilde{d}_i = ||\tilde{\mathbf{a}}_i||_1$ is almost unbiased; i.e., the expectation of $\tilde{d}_i$ is almost equal to d_i. In Section 4.3, we explain the details of the DPRR algorithm.

Note that the DPRR uses two privacy budgets. One is for the Laplacian noise, and the other is for Warner's RR. In Section 4.4, we propose a privacy budget allocation method so that the variance of the noisy degree $\tilde{d}_i$ is small. Since the noisy degree $\tilde{d}_i$ is almost unbiased and has a small variance, the noisy neighbor list $\tilde{\mathbf{a}}_i$ preserves the degree information of user v_i. In Section 4.5, we formally prove this property.

[4]We could use the Geometric mechanism, a discrete version of the Laplacian mechanism, for degrees. However, the Geometric mechanism does not improve the Laplacian mechanism when $\varepsilon_1 < 1$ [18] (we set $\varepsilon_1 < 0.2$ in our experiments). Thus, we use the Laplacian mechanism.

Algorithm 1 Degree-Preserving Randomized Response

Input: neighbor list $\mathbf{a}_i \in \{0,1\}^n$, $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_{\geq 0}$
Output: noisy neighbor list $\tilde{\mathbf{a}}_i \in \{0,1\}^n$

1: $d_i \leftarrow ||\mathbf{a}_i||_1$
2: $d_i^* \leftarrow d_i + \mathrm{Lap}(\frac{1}{\varepsilon_1})$
3: $p \leftarrow \frac{e^{\varepsilon_2}}{e^{\varepsilon_2}+1}$
4: $q_i \leftarrow \frac{d_i^*}{d_i^*(2p-1)+(n-1)(1-p)}$
5: $q_i \leftarrow \texttt{Proj}(q_i)$
6: $\tilde{\mathbf{a}}_i \leftarrow \texttt{RR}(\mathbf{a}_i, p)$
7: $\tilde{\mathbf{a}}_i \leftarrow \texttt{EdgeSampling}(\tilde{\mathbf{a}}_i, q_i)$
8: **return** $\tilde{\mathbf{a}}_i$

4.3 Algorithm Details

Algorithm 1 shows an algorithm for the DPRR. It assigns privacy budgets $\varepsilon_1, \varepsilon_2 \in \mathbb{R}_{\geq 0}$ to the Laplacian noise and RR, respectively.

First, we add the Laplacian noise $\mathrm{Lap}(\frac{1}{\varepsilon_1})$ to the degree d_i $(= ||\mathbf{a}_i||_1)$ of user v_i to obtain a private estimate d_i^* of d_i: $d_i^* = d_i + \mathrm{Lap}(\frac{1}{\varepsilon_1})$ (lines 1-2). Then we tune the sampling probability q_i as follows:

$$q_i = \frac{d_i^*}{d_i^*(2p-1) + (n-1)(1-p)}, \tag{6}$$

where $p = \frac{e^{\varepsilon_2}}{e^{\varepsilon_2}+1}$ (lines 3-4). p represents the probability that Warner's RR sends an input value as is; i.e., it flips $0/1$ with probability $1-p = \frac{1}{e^{\varepsilon_2}+1}$. In Section 4.5, we show that the noisy degree $\tilde{d}_i = ||\tilde{\mathbf{a}}_i||_1$ becomes almost unbiased by setting q_i by (6).

Note that there is a small probability that q_i in (6) is outside of $[0,1]$. Thus, we call the `Proj` function, which projects q_i onto $[0,1]$; i.e., if $q_i < 0$ (resp. $q_i > 1$), then we set $q_i = 0$ (resp. $q_i = 1$) (line 5).

Next, we apply Warner's RR to a neighbor list $\mathbf{a}_i$ of v_i. Specifically, we call the `RR` function, which takes $\mathbf{a}_i$ and p as input and outputs a noisy neighbor list $\tilde{\mathbf{a}}_i$ by sending each bit as is with probability p (line 6). Finally, we apply edge sampling to $\tilde{\mathbf{a}}_i$. Specifically, we call the `EdgeSampling` function, which randomly samples each 1 (edge) with probability q_i (line 7).

In summary, we output $\tilde{\mathbf{a}}_i$ by the RR and edge sampling with the following probability:

$$\forall j \in [n] \setminus \{i\},\ \Pr(\tilde{a}_{i,j} = 1) = \begin{cases} pq_i & (\text{if } a_{i,j} = 1) \\ (1-p)q_i & (\text{otherwise}), \end{cases} \tag{7}$$

where $\tilde{a}_{i,j} \in \{0,1\}$ is the j-th element of $\tilde{\mathbf{a}}_i$.

Privacy. Below, we show the privacy property of the DPRR.

Proposition 5. *The DPRR (Algorithm 1) provides ε-edge LDP, where $\varepsilon = \varepsilon_1 + \varepsilon_2$.*

The proof is given in Appendix A. Since DP is immune to post-processing [20], the GNN model trained from the noisy adjacency matrix $\tilde{\mathbf{A}}$ provides $(\varepsilon_1 + \varepsilon_2)$-edge LDP as well.

4.4 Privacy Budget Allocation

Our DPRR uses two privacy budgets: ε_1 for the Laplacian noise and ε_2 for the RR. When ε_1 is too small, the Laplacian noise becomes too large and causes a large variance of the noisy degree $\tilde{d}_i = ||\tilde{\mathbf{a}}_i||_1$. In contrast, when ε_2 is too small, the RR adds too much noise to each edge. Below, we propose a method to allocate ε_1 and ε_2 to avoid these two issues. In a nutshell, our privacy allocation method sets ε_1 to guarantee a small variance of $\tilde{d}_i$ while keeping a large value of ε_2 to make the noise in the RR small.

Specifically, let $n_{max} \in \mathbb{R}_{\geq 0}$ be the maximum number of nodes (users) in all graphs $\mathcal{G}_l$ and $\mathcal{G}_u$. In Section 4.5, we show that if $\varepsilon_1 < \sqrt{\frac{8}{n_{max}-1}}$, the Laplacian noise is too large and causes a large variance of $\tilde{d}_i$. Taking this into account, our privacy budget allocation method sets ε_1 as follows:

$$\varepsilon_1 = \max\left\{\sqrt{\frac{8}{n_{max}-1}}, (1-\alpha)\varepsilon\right\}, \tag{8}$$

where α is a constant close to 1 ($\alpha = 0.9$ in our experiments).

This setting makes the variance of $\tilde{d}_i$ small (i.e., $\varepsilon_1 \geq \sqrt{\frac{8}{n_{max}-1}}$). It also allows the Laplacian noise to decrease with increase in ε. Moreover, it makes the noise in the RR small (i.e., it makes ε_2 large), as α is large. In our experiments, we show that this setting makes the variance of the noisy degree $\tilde{d}_i$ small and provides high classification accuracy in GNNs.

4.5 Degree Preservation Property

We now show the degree preservation property of the DPRR. Specifically, we analyze the expectation and variance of the noisy degree $\tilde{d}_i = ||\tilde{\mathbf{a}}_i||_1$.

Expectation of the Noisy Degree. We first analyze the expectation $\mathbb{E}[\tilde{d}_i]$ of the noisy degree $\tilde{d}_i$ over the randomness in the Laplacian noise, the RR, and the edge sampling. By the law of total expectation, we have

$$\mathbb{E}[\tilde{d}_i] = \mathbb{E}[\mathbb{E}[\tilde{d}_i|d_i^*]], \tag{9}$$

where d_i^* is a private estimate of d_i (see line 2 of Algorithm 1). The original neighbor list $\mathbf{a}_i$ has d_i 1s and $n-1-d_i$ 0s (except for $a_{i,i}$). By (7), the DPRR sends 1 as is with probability pq_i and flips 0 to 1 with probability $(1-p)q_i$. Thus, we have

$$\begin{aligned}\mathbb{E}[\tilde{d}_i|d_i^*] &= d_i pq_i + (n-1-d_i)(1-p)q_i \\ &= (d_i(2p-1) + (n-1)(1-p))q_i \\ &= \tfrac{d_i(2p-1)+(n-1)(1-p)}{d_i^*(2p-1)+(n-1)(1-p)} d_i^* \quad \text{(by (6))}.\end{aligned}$$

In practice, real social graphs are sparse, which means that $d_i, d_i^* \ll n$ holds for the vast majority of nodes. In addition, $p = \frac{e^{\varepsilon_2}}{e^{\varepsilon_2}+1}$ is not close to 1 when ε_2 is small, e.g., $p = 0.73$ when $\varepsilon_2 = 1$. We are interested in such a value of p; otherwise, we cannot provide edge privacy. Thus, we have $d_i(2p-1), d_i^*(2p-1) \ll (n-1)(1-p)$, hence

$$\mathbb{E}[\tilde{d}_i|d_i^*] \approx \tfrac{(n-1)(1-p)}{(n-1)(1-p)} d_i^* = d_i^*. \tag{10}$$

By (9) and (10), we have

$$\mathbb{E}[\tilde{d}_i] \approx \mathbb{E}[d_i^*] = d_i \ \text{ (as the mean of Lap}(\tfrac{1}{\varepsilon_1}) \text{ is 0)}, \tag{11}$$

Table 2: Time and space complexity of three LDP algorithms per graph.

Algorithm	Time (Each User)	Time (Server)	Space (Each User)	Space (Server)
DPRR	$O(n)$	$O(\|E\|)$	$O(n)$	$O(\|E\|)$
RR	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$
LocalLap	$O(n)$	$O(\|E\|)$	$O(n)$	$O(\|E\|)$
NonPriv-Part	$O(n)$	$O(\|E\|)$	$O(n)$	$O(\|E\|)$

which means that the noisy degree $\tilde{d}_i$ is almost unbiased.

Variance of the Noisy Degree. Next, we analyze the variance $\mathbb{V}[\tilde{d}_i]$ of the noisy degree $\tilde{d}_i$. By the law of total variance, we have

$$\mathbb{V}[\tilde{d}_i] = \mathbb{E}\left[\mathbb{V}[\tilde{d}_i|d_i^*]\right] + \mathbb{V}\left[\mathbb{E}[\tilde{d}_i|d_i^*]\right]. \tag{12}$$

Recall that the original neighbor list $\mathbf{a}_i$ has d_i 1s and $n-1-d_i$ 0s. By (7), we have

$$\begin{aligned}&\mathbb{V}[\tilde{d}_i|d_i^*]\\ &= d_i pq_i(1-pq_i) + (n-1-d_i)(1-p)q_i(1-(1-p)q_i)\\ &\leq \tfrac{n_{max}-1}{4},\end{aligned} \tag{13}$$

where the equality holds if and only if $n = n_{max}$, $p = \frac{1}{2}$ and $q_i = 1$. By (10), the second term of (12) can be written as follows:

$$\mathbb{V}\left[\mathbb{E}[\tilde{d}_i|d_i^*]\right] \approx \mathbb{V}[d_i^*] = \mathbb{V}[\mathrm{Lap}(\tfrac{1}{\varepsilon_1})] = \tfrac{2}{\varepsilon_1^2}. \tag{14}$$

By (12), (13), and (14), we have

$$\mathbb{V}[\tilde{d}_i] \approx \mathbb{E}\left[\mathbb{V}[\tilde{d}_i|d_i^*]\right] + \mathbb{V}\left[d_i^*\right] \leq \tfrac{n_{max}-1}{4} + \tfrac{2}{\varepsilon_1^2}. \tag{15}$$

The first term of (15) is caused by the randomness of the RR, whereas the second term of (15) is caused by the randomness of the Laplacian noise.

If $\varepsilon_1 < \sqrt{\frac{8}{n_{max}-1}}$, the second term of (15) is larger than the first term of (15); i.e., the Laplacian noise is dominant. Thus, our privacy allocation method sets $\varepsilon_1 \geq \sqrt{\frac{8}{n_{max}-1}}$ (see (8)). In this case, the variance is bounded as follows: $\mathbb{V}[\tilde{d}_i] \leq \frac{n_{max}-1}{2}$.

Summary. Our DPRR makes the noisy degree $\tilde{d}_i = ||\tilde{\mathbf{a}}_i||_1$ almost unbiased by automatically tuning the sampling probability q_i by (6). In addition, our privacy allocation method makes the variance of $\tilde{d}_i$ small (i.e., $\mathbb{V}[\tilde{d}_i] \leq \frac{n_{max}-1}{2}$) by setting ε_1 by (8). Consequently, the noisy neighbor list $\tilde{\mathbf{a}}_i$ preserves the degree information of user v_i. We also show this degree-preserving property of the DPRR through experiments.

4.6 Time and Space Complexity

Finally, we show that our DPRR has much smaller time and space complexity than Warner's RR. Table 2 shows the time and space complexity of three LDP algorithms: the DPRR,

Warner's RR applied to each bit of the adjacency matrix $\mathbf{A}$, a local model version of LAPGRAPH [61] (LocalLap), and an algorithm that uses a graph composed of only non-private users (NonPriv-Part). We explain the details of LocalLap in Section 5.1. In Table 2, we assume that the server runs efficient GNN algorithms whose time complexity is linear in the number of edges, e.g., GIN (Graph Isomorphism Network) [63], GCN (Graph Convolutional Networks) [38], and GraphSAGE (Graph Sample and Aggregate) [26]. Some of the other GNN algorithms are less efficient; see [62] for details.

Table 2 shows that the time and space complexity on the server side is $O(n^2)$ in Warner's RR. This is because the RR makes a graph dense; i.e., $|\tilde{\mathbf{A}}| = O(n^2)$. In contrast, the DPRR preserves each user's degree information, and consequently, $|\tilde{\mathbf{A}}| = O(|E|)$, where $|E|$ is the number of edges in the original graph G. Thus, the DPRR has the time and space complexity of $O(|E|)$ on the server side. Since $|E| \ll n^2$ in practice, the DPRR is much more efficient than the RR.

For example, the Orkut social network [65] includes 3072441 nodes and 117185083 edges. The RR needs a memory size of 1 TB to store the noisy graph on the server side. In contrast, the memory size of the DPRR required to store the noisy graph on the server side is only about 30 MB, which is much smaller than that of the RR. This comes from the fact that the server has only sparse noisy neighbor lists generated by the DPRR. In our experiments, we also show that the DPRR needs much less time for both training and classification.

LocalLap has the same time and space complexity as the DPRR. In our experiments, we show that our DPRR significantly outperforms LocalLap in terms of accuracy.

Note that each user's time and space complexity is much smaller than the server's. Specifically, all of the DPRR, RR, and LocalLap have the time and space complexity of $O(n)$ because the length of each user's neighbor list is $O(n)$.

5 Experimental Evaluation

Based on the theoretical properties of our DPRR in Sections 4.5 and 4.6, we pose the following research questions:

RQ1. How does our DPRR compare with other private algorithms in terms of accuracy and time complexity?

RQ2. How accurate is our DPRR compared to a non-private algorithm?

RQ3. How much does our DPRR preserve each user's degree information?

We conducted experiments to answer these questions.

5.1 Experimental Set-up

Dataset. We used three unattributed social graph datasets:

- **REDDIT-MULTI-5K.** REDDIT-MULTI-5K [64] is a graph dataset in Reddit [6], where each graph represents an online discussion thread. An edge between two nodes represents that a conversation happens between the two users. A label represents the type of subreddit, and there are five subreddits: worldnews, videos, AdviceAnimals, aww, and mildyinteresting.

Table 3: Statistics of graph datasets. "#nodes" represents the number of nodes in one graph.

Dataset	#graphs	#classes	#nodes (max)	#nodes (avg)	degree (max)	degree (avg)
REDDIT-MULTI-5K	4999	5	3782	508.5	2011	2.34
REDDIT-BINARY	2000	2	3648	429.6	3062	2.32
Github StarGazers	12725	2	957	113.8	755	4.12

- **REDDIT-BINARY.** REDDIT-BINARY [64] is a social graph dataset in Reddit. The difference from REDDIT-MULTI-5K lies in labels. In REDDIT-BINARY, a label represents a type of community, and there are two types of communities: a question/answer-based community and a discussion-based community.

- **Github StarGazers.** Github StarGazers [51] includes a social network of developers who starred popular machine learning and web development repositories until August 2019. A node represents a user, and an edge represents a follower relationship. A label represents the type of repository: machine learning or web.

Table 3 shows statistics of each graph dataset.

LDP Algorithms. For comparison, we evaluated the following four private algorithms:

- **DPRR.** Our proposed algorithm providing ε-edge LDP, where $\varepsilon = \varepsilon_1 + \varepsilon_2$ (Algorithm 1). As described in Section 4.4, we used our privacy budget allocation method and set $\varepsilon_1 = \max\left\{\sqrt{\frac{8}{n_{max}-1}}, (1-\alpha)\varepsilon\right\}$ and $\varepsilon_2 = \alpha\varepsilon$, where n_{max} is the maximum number of nodes. We set $\alpha = 0.9$ as a default value to make the noise in the RR small (i.e., to make ε_2 large), as described in Section 4.4. We also change α to various values when we compare our privacy budget allocation method with a baseline that sets $\varepsilon_1 = (1-\alpha)\varepsilon$ and $\varepsilon_2 = \alpha\varepsilon$.

- **RR.** Warner's RR for each bit of the neighbor list [31, 32, 49, 66]. It flips each 0/1 with probability $\frac{1}{e^{\varepsilon}+1}$. It provides ε-edge LDP.

- **LocalLap.** A local model version of LAPGRAPH [61]. LAPGRAPH is an algorithm providing edge DP in the centralized model. Specifically, LAPGRAPH adds Lap($\frac{1}{\varepsilon_1}$) to the total number of edges in a graph G. Then it adds Lap($\frac{1}{\varepsilon_2}$) to each element in the upper-triangular part of $\mathbf{A}$ and selects T largest elements (edges), where T is the noisy number of edges. Finally, it outputs a noisy graph with the selected edges.

  LAPGRAPH cannot be used in the local model, as it needs the total number of edges in G as input. Thus, we modified LAPGRAPH so that each user v_i sends a noisy degree d_i^* ($= d_i + \text{Lap}(\frac{1}{\varepsilon_1})$) to the server and the server calculates the noisy number T of edges as: $T = (\sum_{i=1}^{n} d_i^*)/2$. We call this modified algorithm LocalLap. LocalLap provides ε-edge LDP, where $\varepsilon = \varepsilon_1 + \varepsilon_2$. We set $\varepsilon_1 = \frac{\varepsilon}{10}$ and $\varepsilon_2 = \frac{9\varepsilon}{10}$.

- **NonPriv-Part.** An algorithm that discards neighbor lists of private users and uses only neighbor lists of non-private users. Specifically, it constructs a graph composed of only non-private users and uses it as input to GNN.

Note that NonPriv-Part is an algorithm in the customized setting and cannot be used in the common setting.

We also evaluated NonPriv-Full, a fully non-private algorithm that does not add any noise for all users. Note that NonPriv-Full does not protect privacy for private users, unlike the four private algorithms explained above. The accuracy of private algorithms is high if it is close to that of NonPriv-Full.

Finally, we emphasize that the algorithm in [52] cannot be evaluated in our experiments, because it violates edge LDP (see Section 3.3).

GNN Models and Configurations. Following [23, 63], we used a constant value as the initial feature vector. In this case, GCN [38] and GraphSAGE [26] do not perform better than random guessing, as proved in [63]. Therefore, we used the GIN [63], which provides state-of-the-art performance in graph classification, as a GNN model. The GIN uses a sum function as $\mathsf{AGGREGATE}^{(k)}$ in (3), MLPs as $\mathsf{COMBINE}^{(k)}$ in (4), and a sum function as $\mathsf{READOUT}$ in (5).

We used the implementation in [3] and used the same parameters and configurations as GIN-0 in [63], which provides the best empirical performance. Specifically, we used the mean readout as a readout function. We applied linear mapping and a dropout layer to a graph feature vector. All MLPs had two layers. We used the Adam optimizer. We applied Batch normalization to each hidden layer. The batch size was 64.

Following [23, 67], we tuned hyper-parameters via grid search. The hyper-parameters are: (i) the number of GNN layers $\in \{1, 2, 3, 4\}$ (resp. $\{2, 3, 4, 5\}$) in Github StarGazers (resp. REDDIT-MULTI-5K and REDDIT-BINARY); (ii) the number of hidden units $\in \{16, 32, 64, 128\}$; (iii) the initial learning rate $\in \{10^{-4}, 10^{-3}, 10^{-2}\}$; (iv) the dropout ratio $\in \{0, 0.5\}$.

Training, Validation, and Testing Data. For Github StarGazers, we followed [67] and randomly selected 65%, 15%, and 20% of the graphs for training, validation, and testing, respectively. For REDDIT-MULTI-5K and REDDIT-BINARY, we randomly selected 75%, 10%, and 15% for training, validation, and testing, respectively, as they have smaller numbers of graphs. We tuned the hyper-parameters explained above using the graphs for validation. Then we trained the GNN using the training graphs. Here, following [23, 27], we applied early stopping. The training was stopped at 100 to 200 epochs in most cases (500 epochs at most).

Finally, we evaluated the classification accuracy and AUC (Area Under the Curve) using the testing graphs. We attempted ten cases for randomly dividing graphs into training, validation, and test sets and evaluated the average classification accuracy and AUC over the ten cases.

5.2 Experimental Results

Accuracy. First, we compared the accuracy of our DPRR with that of the three baselines (RR, LocalLap, and NonPriv-Part). We randomly selected λn users as non-private users, where λ is the proportion of non-private users. We set ε for private users to $\varepsilon = 1$ and set λ to $\lambda = 0$, 0.1, 0.2, or 0.5. The value $\lambda = 0$ corresponds to the common setting, whereas the value $\lambda = 0.1$, 0.2, or 0.5 corresponds to the customized setting. Then, we set $\lambda = 0$ (i.e., common setting) and set ε to $\varepsilon = 0.2$, 0.5, 1, or 2.

Figures 4 and 5 show the results. Here, a dashed line represents the accuracy of NonPriv-Full. Note that NonPriv-Full is equivalent to NonPriv-Part with $\lambda = 1$; i.e., NonPriv-Full regards all users as non-private.

We observe that our DPRR provides the best (or almost the best) performance in all cases. The DPRR significantly outperforms the RR in the customized setting where $\lambda > 0$. The accuracy of the RR is hardly increased with an increase in λ. This is because the RR makes

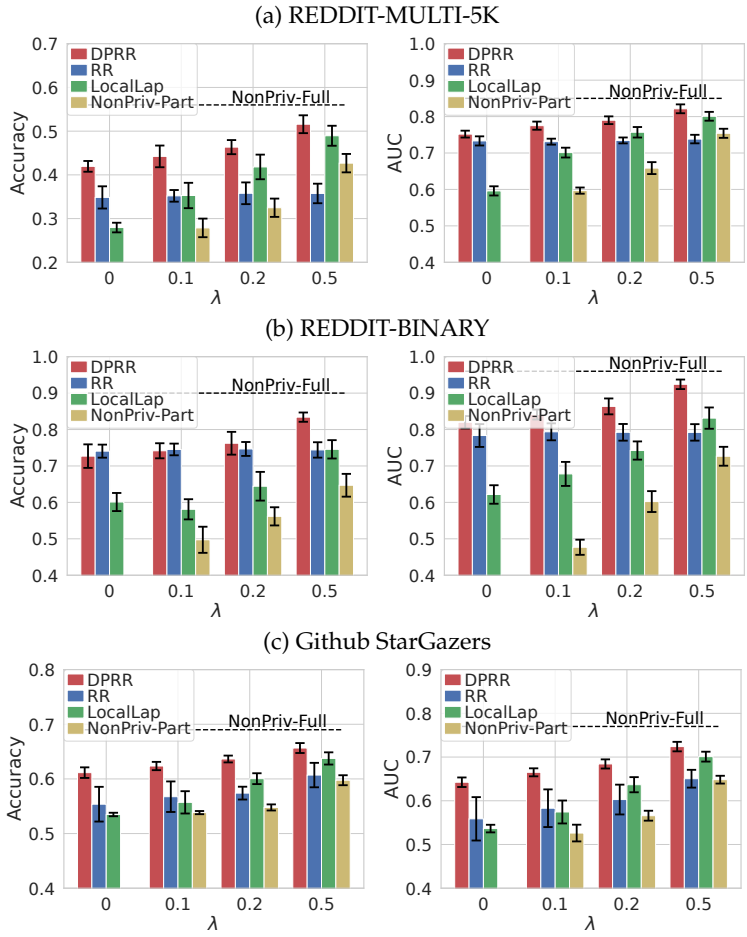


Figure 4: Classification accuracy and AUC for different proportions λ of non-private users ($\varepsilon = 1$). An error bar represents the standard deviation. Note that the RR is also inefficient in terms of the training/classification time and the memory size.

neighbor lists of private users dense. This ruins the neighborhood aggregation for non-private users. In contrast, the accuracy of our DPRR dramatically increases with an increase in λ. We also emphasize that the DPRR is much more efficient than the RR, as shown later.

We also observe that our DPRR significantly outperforms LocalLap in all cases. One reason for this is that LocalLap does not preserve each user's degree information, as shown later. Our DPRR also significantly outperforms NonPriv-Part, which means that the DPRR effectively uses neighbor lists of private users.

Moreover, our DPRR provides accuracy close to NonPriv-Full with a reasonable privacy budget. For example, when $\varepsilon = 1$ and $\lambda = 0.2$, the difference in the classification accuracy or AUC is smaller than 0.1 or so in all datasets. These results demonstrate the effectiveness of the DPRR.

In our experiments, the proportion λ of non-private users is the same between training graphs and testing graphs. In practice, λ can be different between them. However, we note that we can make λ the same between training and testing graphs by adding DP noise for

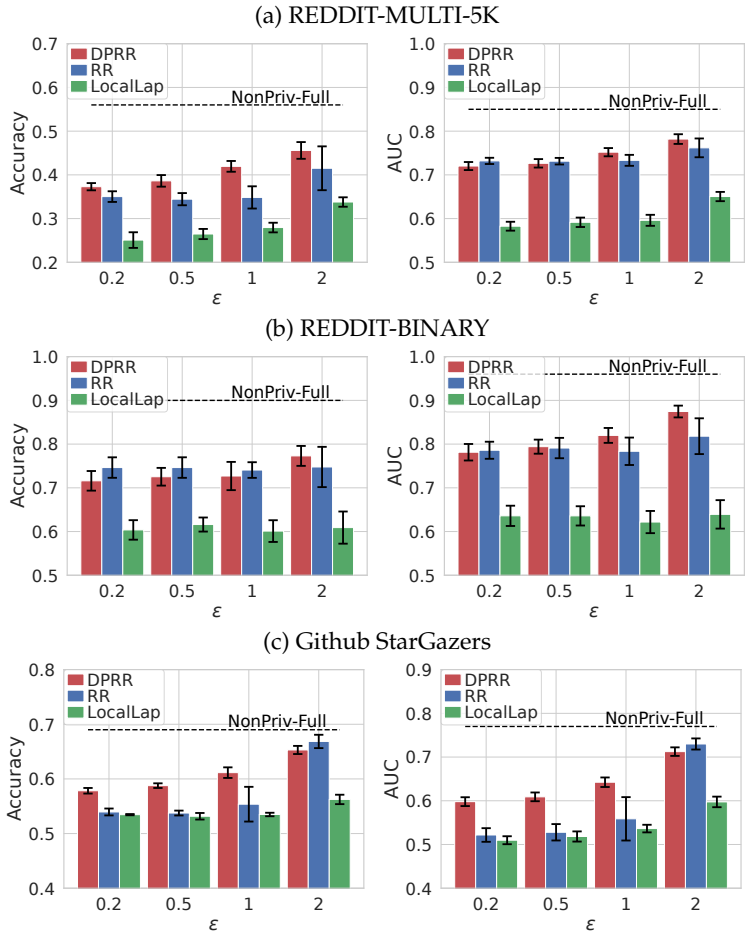


Figure 5: Classification accuracy and AUC for different privacy budgets ε ($\lambda = 0$). An error bar represents the standard deviation. Note that the RR is also inefficient in terms of the training/classification time and the memory size.

some non-private users. For example, assume that $\lambda = 0.1$ in training graphs and $\lambda = 0.2$ in testing graphs. By adding DP noise for some non-private users in the testing graphs, we can make $\lambda = 0.1$ for both the training and testing graphs. Figure 4 shows that our DPRR is effective for all values of λ including $\lambda = 0$.

Privacy Budget Allocation. Next, we examined the effectiveness of our privacy budget allocation method in Section 4.4. Specifically, we compared our proposed method with a baseline that always sets ε_1 and ε_2 to $\varepsilon_1 = (1-\alpha)\varepsilon$ and $\varepsilon_2 = \alpha\varepsilon$. We set $\varepsilon = 0.2$ and the proportion λ of non-private users to $\lambda = 0$ (i.e., common setting) or 0.5 (i.e., customized setting). Note that we set ε to a small value ($= 0.2$) so that a difference occurs between our proposed method and the baseline. Then, we changed α to $\alpha = 0.5, 0.6, 0.7, 0.8$, or 0.9.

Figure 6 shows the results. "DPRR (Proposal)" represents our DPRR with our privacy budget allocation method. Note that $\sqrt{\frac{8}{n_{max}-1}}$ in (8) is 0.046, 0.047, and 0.091 in REDDIT-MULTI-5K, REDDIT-BINARY, and Github StarGazers, respectively. Thus, "DPRR (Proposal)" is identical to "DPRR ($\varepsilon_1 = (1-\alpha)\varepsilon$, $\varepsilon_2 = \alpha\varepsilon$)" when $\alpha \leq 0.7$, $\alpha \leq 0.7$, and $\alpha = 0.5$

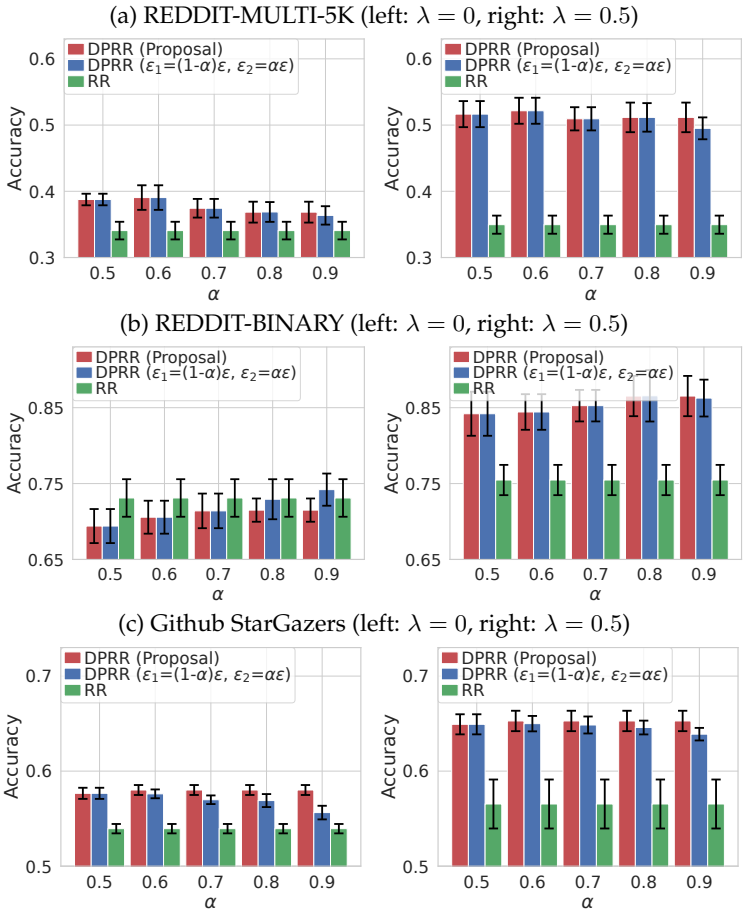


Figure 6: Classification accuracy for different parameters α in our privacy budget allocation method ($\varepsilon = 0.2$, $\lambda = 0$ or 0.5). An error bar represents the standard deviation.

in REDDIT-MULTI-5K, REDDIT-BINARY, and Github StarGazers, respectively.

Figure 6 (a) and (c) show that when α is large, our privacy budget allocation method outperforms the baseline ($\varepsilon_1 = (1-\alpha)\varepsilon$, $\varepsilon_2 = \alpha\varepsilon$) in REDDIT-MULTI-5K and Github StarGazers. This is because our proposed method makes the Laplacian noise small. The difference is larger in Github StarGazers because the original degree d_i is smaller (see Figure 7) and is more susceptible to the Laplacian noise.

However, the left figure of Figure 6 (b) shows that when $\lambda = 0$, our privacy budget allocation method provides lower accuracy than the baseline ($\varepsilon_1 = (1-\alpha)\varepsilon$, $\varepsilon_2 = \alpha\varepsilon$) in REDDIT-BINARY. This is because our DPRR is slightly outperformed by Warner's RR in this case. In other words, the degree information does not contribute much to classification accuracy. In this case, it might be better to set the privacy budget ε_2 for Warner's RR as much as possible. In contrast, the right figure of Figure 6 (b) shows that when $\lambda = 0.5$, our privacy budget allocation method slightly outperforms the baseline ($\varepsilon_1 = (1-\alpha)\varepsilon$, $\varepsilon_2 = \alpha\varepsilon$) in REDDIT-BINARY. This is because our DPRR outperforms Warner's RR in this case.

In summary, our privacy budget allocation method works well, especially when the degree information contributes to classification accuracy and the original degree d_i is small.

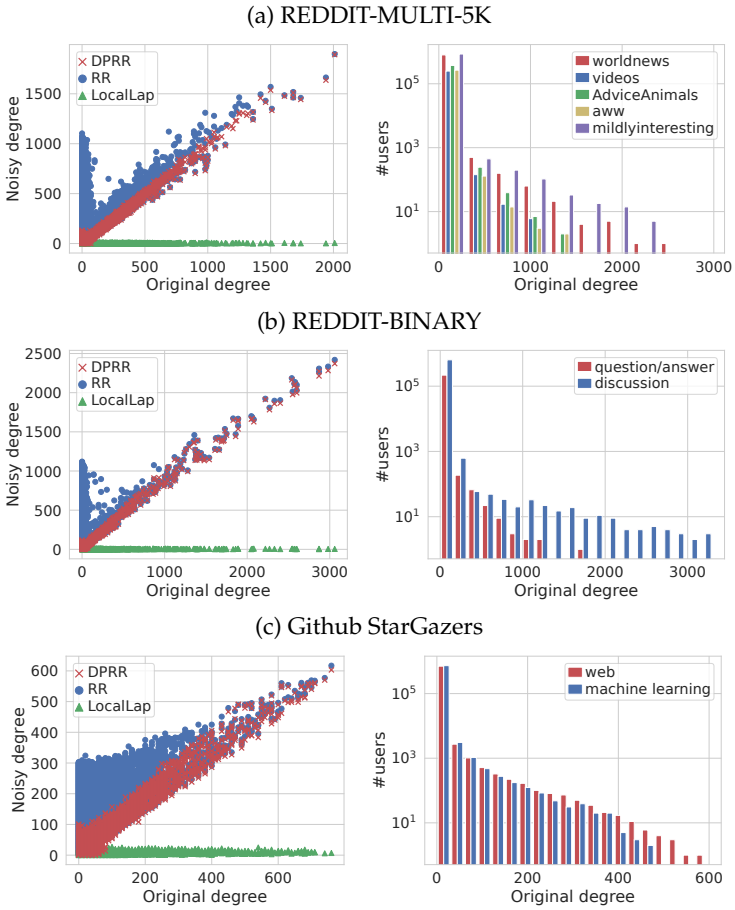


Figure 7: Relationship between original degree d_i and noisy degree $\tilde{d}_i$ (left) and degree distribution for each graph type (right) ($\varepsilon = 1$). In the left panels, each point shows the result for one user.

Degree Preservation. We also examined how well the DPRR preserves each user's degree information to explain the reason that the DPRR outperforms the RR and LocalLap. The left panels of Figure 7 show the relationship between each user v_i's original degree d_i and noisy degree $\tilde{d}_i$. We also examined how the degree information of a graph correlates with its type. The right panels of Figure 7 show a degree distribution (i.e., distribution of d_i) for each graph type.

The left panels of Figure 7 show that the DPRR preserves the original degree very well. This holds especially when the original degree is small, e.g., $d_i < 500$, 1000, and 200 in REDDIT-MULTI-5K, REDDIT-BINARY, and Github StarGazers, respectively. This is because the expectation of $\tilde{d}_i$ is almost equal to d_i (see (11)) when $d_i \ll n$. In other words, the experimental results are consistent with our theoretical analysis. Moreover, the right panels of Figure 7 show that $d_i < 500$, 1000, and 200 in almost all cases in REDDIT-MULTI-5K, REDDIT-BINARY, and Github StarGazers, respectively. Thus, $\tilde{d}_i \approx d_i$ holds for the vast majority of users in the DPRR.

The right panels of Figure 7 show that the degree information differs for each graph type.

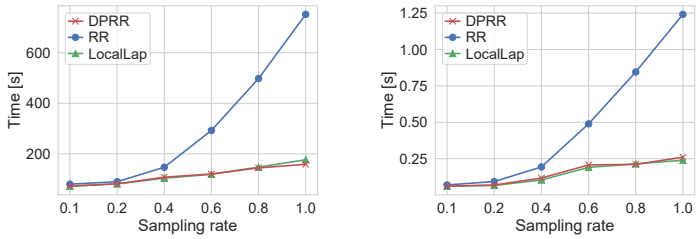


Figure 8: Run time vs. sampling rate γ in REDDIT-BINARY (left: training time, right: classification time).

For example, in REDDIT-MULTI-5K, the worldnews community tends to have a larger degree than the videos community; i.e., the worldnews community tends to attract more people.

The left panels of Figure 7 show that the RR and LocalLap do not preserve the degree information for the vast majority of users. The RR makes neighbor lists of these users dense and destroys the graph structure. LocalLap makes neighbor lists sparse, irrespective of the original degrees. In contrast, the DPRR preserves the degree information for these users. This explains why the DPRR outperforms the RR and LocalLap.

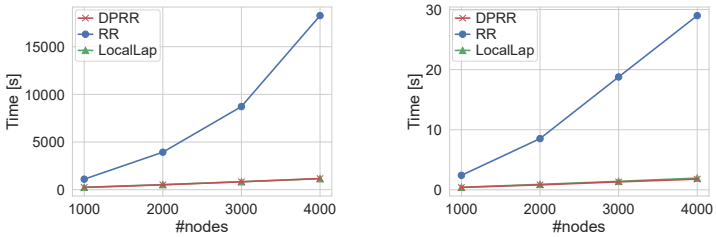
Training/Classification Time. Finally, we measured the time for training and classification on the server side in the DPRR, RR, and LocalLap. Here, we used two datasets: REDDIT-BINARY and a synthetic graph dataset based on the BA (Barabási-Albert) model [11]. The BA model is a graph generation model that has a power-law degree distribution. It generates a synthetic graph by adding new nodes one by one. Each node has $m \in \mathbb{N}$ new edges, and each edge is connected to an existing node with probability proportional to its degree. The average degree of the BA graph is $2m$. We used the NetworkX library [25] to generate the BA graph.

For each dataset, we evaluated the relationship between the training/classification time and the graph size. Specifically, in REDDIT-BINARY, we used 75% of the graphs (i.e., 1500 graphs) and 15% of the graphs (i.e., 300 graphs) for training and classification, respectively, as explained in Section 5.1. For each graph, we randomly sampled γn nodes, where $\gamma \in [0, 1]$ is a sampling rate, and used a subgraph composed of the sampled nodes. In the BA graph dataset, we generated 1000 and 200 graphs for training and classification, respectively. We set $m = 3$ or 5 and $n = 1000$, 2000, 3000, or 4000. We evaluated the relationship between the run time and γ (resp. n) in REDDIT-BINARY (resp. BA graph dataset).

We measured the run time using a supercomputer in [1]. We used one computing node, which consists of two Intel Xeon Platium 8360Y processors (2.4 GHz, 36 Cores) and 512 GiB main memory. For training, we measured the time to run 100 epochs because the training was stopped at 100 to 200 epochs in most cases, as described in Section 5.1. For classification, we measured the time to classify all graphs for classification. We used the implementation in [3] as a code of GNN. Note that the DPRR, RR, and LocalLap only differ in the input to GNN, and therefore the comparison is fair.

Figures 8 and 9 show the results. The run time of the RR is large and almost quadratic in the graph size, i.e., the sampling rate γ in Figure 8 and the number n of nodes in Figure 9. In contrast, the run time of the DPRR and LocalLap is much smaller than the RR. Figure 9 shows that when the average degree is constant ($= 2m$), the run time of the DPRR and LocalLap is almost linear in n. These results are consistent with Table 2.

(a) $m = 3$



(b) $m = 5$

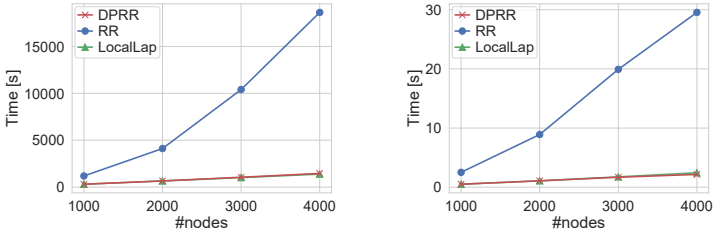


Figure 9: Run time vs. #nodes n in the BA graphs (left: training time, right: classification time).

We can also estimate the run time for larger n based on Figure 8. For example, Figure 8 (c) shows that when $m = 5$ and $n = 4000$, the training time of the RR and our DPRR is about 19000 and 1500 seconds, respectively. Thus, when $m = 5$ and $n = 40000$, the training time of the RR and our DPRR is estimated to be about 21 days ($= 19000 \times 10^2$ seconds) and 4 hours ($= 1500 \times 10/3600$ seconds), respectively. Therefore, our DPRR is much more efficient and practical than the RR.

Summary. In summary, our answers to the three research questions at the beginning of Section 5 are as follows. [RQ1]: Our DPRR is much more efficient than the RR and provides higher accuracy than the RR, especially in the customized setting. Our DPRR also provides much higher accuracy than the other private baselines (LocalLap and NonPriv-Part) in terms of accuracy. [RQ2]: Our DPRR provides accuracy close to a non-private algorithm (NonPriv-Full) with a reasonable privacy budget, e.g., $\varepsilon = 1$. [RQ3]: Our DPRR preserves each user's degree information very well, whereas the RR and LocalLap do not. In addition, the degree information of a graph correlates with its type. These results explain why the DPRR outperforms the RR and LocalLap in terms of accuracy.

6 Data Poisoning Attacks and Defenses

As with most of the existing work on LDP (e.g., [10, 12, 22, 31, 32, 48, 49, 57, 66]), we have so far assumed that users are honest. That is, we assumed that each user v_i honestly applies a local randomizer $\mathcal{R}_i$ to her neighbor list $\mathbf{a}_i$ and reports the noisy neighbor lists $\tilde{\mathbf{a}}_i$. However, recent studies [14, 17] show that LDP algorithms are vulnerable to data poisoning attacks, as described in Section 1. Therefore, we finally evaluate the robustness of our DPRR against data poisoning attacks.

Section 6.1 introduces a general data poisoning attack to our DPRR, which includes the

all-ones attack and the *random attack* as concrete examples. Section 6.2 introduces a defense against the attacks that can be applied to both directed and undirected graphs. Section 6.3 evaluates the all-ones attack, the random attack, and the defense through experiments and discusses the results.

6.1 Data Poisoning Attacks

Threat Model. Assume that βn users are malicious, where $\beta \in [0, 1]$ is the proportion of malicious users. Each malicious user v_i can change her noisy neighbor list $\tilde{\mathbf{a}}_i$ to an arbitrary value. This is referred to as the general attack [17]. The adversaries' goal is to degrade the classification accuracy of the GNN as much as possible.

Attack Algorithms. To achieve the above goal, we consider the following attack algorithm. Each malicious user v_i applies a local randomizer $\mathcal{R}_i$ to her neighbor list $\mathbf{a}_i$. Then, v_i changes the noisy neighbor list $\tilde{\mathbf{a}}_i \in \{0, 1\}^n$ to a *fake* neighbor list $\tilde{\mathbf{a}}_i^* \in \{0, 1\}^n$ with the following probability:

$$\forall j \in [n] \setminus \{i\},\ \Pr(\tilde{a}_{i,j}^* = 1) = \begin{cases} \omega_1 & (\text{if } \tilde{a}_{i,j} = 1) \\ \omega_2 & (\text{otherwise}), \end{cases} \tag{16}$$

where $\tilde{a}_{i,j}$ and $\tilde{a}_{i,j}^*$ are the j-th elements of $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{a}}_i^*$, respectively, and $\omega_1, \omega_2 \in [0, 1]$. Finally, the malicious user v_i copies $\tilde{\mathbf{a}}_i^*$ to $\tilde{\mathbf{a}}_i$ and sends $\tilde{\mathbf{a}}_i$ to the data collector, who calculates a noisy adjacency matrix $\tilde{\mathbf{A}}$ from $\tilde{\mathbf{a}}_1, \cdots, \tilde{\mathbf{a}}_n$. We denote this attack by Poison(ω_1, ω_2).

This attack is general and includes a lot of concrete attacks. For example, when $\omega_1 = \omega_2 = 1$, the malicious user v_i always generates an all-ones vector $\tilde{\mathbf{a}}_i^* = (1, 1, \ldots, 1)$ (except for the i-th element $\tilde{a}_{i,i}^* = 0$). When $\omega_1 = \omega_2 = 0$, v_i generates an all-zeros vector $\tilde{\mathbf{a}}_i^* = (0, 0, \ldots, 0)$. When $\omega_1 = \omega_2 = 0.5$, v_i generates a uniformly random n-dim binary vector as $\tilde{\mathbf{a}}_i^*$. Note that v_i honestly sends $\tilde{\mathbf{a}}_i$ when $\omega_1 = 1$ and $\omega_2 = 0$. Thus, v_i slightly changes $\tilde{\mathbf{a}}_i$ when ω_1 and ω_2 are close to 1 and 0, respectively.

Since many social graphs are sparse in practice, the adversaries can degrade the classification accuracy of the GNN, especially when the fake neighbor list $\tilde{\mathbf{a}}_i^*$ is dense. Taking this into account, we evaluate Poison$(1, 1)$ and Poison$(0.5, 0.5)$ in our experiments. We refer to Poison$(1, 1)$ and Poison$(0.5, 0.5)$ as an *all-ones attack* and *random attack*, respectively.

6.2 Defenses

Existing Defense. Imola *et al.* [30] propose a defense against poisoning attacks for graph degree estimation. They focus on undirected graphs and use the fact that the adjacency matrix $\mathbf{A}$ is symmetric in this case. Specifically, their defense compares the i-th row of the noisy adjacency matrix $\tilde{\mathbf{A}}$ with the i-th column of $\tilde{\mathbf{A}}$. Then, it determines that user v_i is malicious if the number of inconsistent elements is larger than a threshold.

Unfortunately, it is difficult to apply their defense to our setting for two reasons. First, the defense in [30] assumes that Warner's RR is used as a local randomizer $\mathcal{R}_i$, making it easy to set a threshold so that the detection error probability is controlled. However, our DPRR applies edge sampling after Warner's RR, and the sampling probability q_i is different from user to user. Moreover, the data collector does not know the value of q_i[5]. Thus, it is difficult to set a threshold to control the detection error probability. Second, the defense in

[5]It is also possible for our DPRR to output the sampling probability q_i, as it provides ε_1-edge DP. However, it does not address the issue, because the adversaries can change the value of q_i to an arbitrary value.

[30] cannot be applied to directed graphs, as the adjacency matrix $\mathbf{A}$ is asymmetric in this case. Since we consider both directed and undirected graphs, a new defense is needed.

Our Defense. We introduce a new defense that is applicable to both directed and undirected graphs. Our defense is based on the observation that the adversaries can degrade the accuracy of the GNN, especially when the fake neighbor list is dense (as described in Section 6.2).

Specifically, recall that $\tilde{d}_i$ $(= ||\tilde{\mathbf{a}}_i||_1)$ is the number of 1s in the noisy neighbor list $\tilde{\mathbf{a}}_i$. In addition, recall that the parameter p in Warner's RR is given by $p = \frac{e^{\varepsilon_2}}{e^{\varepsilon_2}+1}$. In our defense, the data collector determines that user v_i is *malicious* if

$$\tilde{d}_i \geq \tau,$$

where τ is a threshold given by

$$\tau = (n-1)p\left(1 + \frac{\log\frac{1}{\theta} + \sqrt{(\log\frac{1}{\theta})^2 + 8(n-1)p\log\frac{1}{\theta}}}{2(n-1)p}\right) \tag{17}$$

and $\theta \in [0, 1]$ is a required value for the *false positive probability* (i.e., the probability that an honest user is misclassified as a malicious user).

The data collector discards noisy neighbor lists of users detected as malicious and uses only noisy neighbor lists of the remaining users (i.e., only a noisy graph composed of the remaining users). We denote this defense by Defense(θ).

Proposition 6. *The false positive probability of Defense(θ) is smaller than or equal to θ; i.e., for any honest user $v_i \in V$, we have*

$$\Pr(\tilde{d}_i \geq \tau) \leq \theta,$$

where $\tilde{d}_i$ is the number of 1s in the noisy neighbor list $\tilde{\mathbf{a}}_i$, and τ is a threshold given by (17).

The proof is given in Appendix B. In our experiments, we set $\theta = 0.05$.

6.3 Evaluation

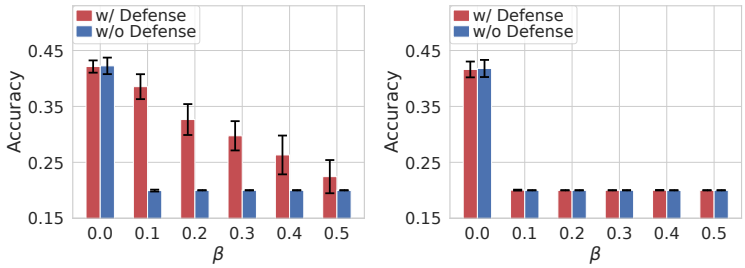
Experimental Set-up. We evaluated the data poisoning attacks and the defense in Sections 6.1 and 6.2, respectively, using the three datasets in Section 5. Specifically, we set the privacy budget ε to $\varepsilon = 1$ and the proportion λ of non-private users to $\lambda = 0$ (i.e., common setting). We set the proportion β of malicious users to $\beta = 0$, 0.1, 0.2, 0.3, 0.4, or 0.5.

For attack algorithms, we evaluated Poison$(1, 1)$ (i.e., all-ones attack) and Poison$(0.5, 0.5)$ (i.e., random attack). In both Poison$(1, 1)$ and Poison$(0.5, 0.5)$, βn malicious users change their noisy neighbor lists in the training graphs. Note that they leave their noisy neighbor lists in the testing graphs unchanged so that the difference between the training and testing graphs is large. We also confirmed that this attack results in lower accuracy than the attack that changes noisy neighbor lists in both the training and testing graphs.

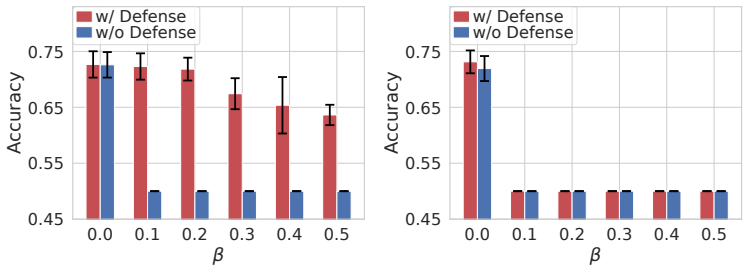
For a defense algorithm, we evaluated Defense(0.05). We used our DPRR with our privacy budget allocation method as a local randomizer, and evaluated the accuracy of this local randomizer with or without Defense(0.05). The other experimental settings are the same as Section 5.1.

Experimental Results. Figure 10 shows the results. The left figures show that the accuracy is significantly degraded by Poison$(1, 1)$ when we do not introduce a defense. This

(a) REDDIT-MULTI-5K (left: Poison$(1, 1)$, right: Poison$(0.5, 0.5)$)



(b) REDDIT-BINARY (left: Poison$(1, 1)$, right: Poison$(0.5, 0.5)$)



(c) Github StarGazers (left: Poison$(1, 1)$, right: Poison$(0.5, 0.5)$)

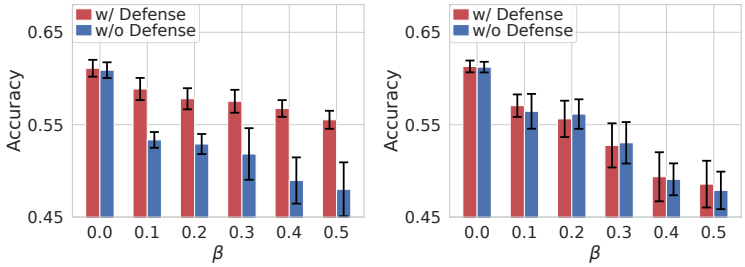


Figure 10: Classification accuracy for different proportions β of malicious users ($\varepsilon = 1$, $\lambda = 0$). "Defense" represents Defense(0.05). An error bar represents the standard deviation.

attack is mitigated by using Defense(0.05). For example, when there is no attack ($\beta = 0$), the accuracy of our DPRR is 0.61 in Github StarGazers. When $\beta = 0.5$, the accuracy of our DPRR with and without Defense(0.05) is 0.56 and 0.48, respectively, which means that Defense(0.05) effectively defends against Poison$(1, 1)$.

However, the right figures show that Defense(0.05) does not mitigate Poison$(0.5, 0.5)$. For example, when $\beta = 0.5$, the accuracy of our DPRR in Github StarGazers is about 0.48, irrespective of the presence or absence of Defense(0.05). This means that Defense(0.05) is not effective for Poison$(0.5, 0.5)$.

Discussions. The reason why Defense(0.05) effectively defends against Poison$(1, 1)$ but not against Poison$(0.5, 0.5)$ lies in the threshold τ in (17). In Defense(0.05), we set $\theta = 0.05$ so that the false positive probability (i.e., the probability that an honest user is misclassified as a malicious user) is smaller than or equal to 0.05. As a result, the threshold τ becomes large. For example, when $n = 100$ and $\varepsilon_2 = 0.9$, the threshold τ in (17) is: $\tau = 92.5$. Thus, Defense(0.05) can detect the all-ones attack Poison$(1, 1)$ that always results in $\tilde{d}_i = 99$ but cannot detect the random attack Poison$(0.5, 0.5)$ that results in $\tilde{d}_i = 49.5$ on average.

In summary, although Defense(θ) in Section 6.2 can be easily applied to both directed and

undirected graphs, it cannot defend against all possible attacks. As described in Section 6.2, the existing defense [30] cannot be applied to our setting, as (i) it assumes Warner's RR for a local randomizer $\mathcal{R}$ and (ii) it cannot be applied to directed graphs. Our experimental evaluation shows that a new defense is needed to prevent all possible attacks more effectively. Developing such a new defense is left for future work.

7 Conclusion

In this paper, we proposed the DPRR and a privacy budget allocation method to provide high accuracy in GNNs with a small privacy budget ε in edge LDP. Through experimental evaluation, we showed that the DPRR outperforms the three baselines (RR, LocalLap, and NonPriv-Part) in terms of accuracy. We also showed that the DPRR is much more efficient than the RR in that it needs much less time for training and classification and much less memory. We also evaluated the robustness of our DPRR against data poisoning attacks.

Although we have focused on unattributed graphs, we can also provide LDP for both edges and feature vectors by combining our DPRR with the algorithm in [52]. Specifically, the authors in [52] propose an algorithm providing LDP for only feature vectors. They assume that original edges are public and use the original edges as input to their algorithms (Algorithms 2 and 3 in [52]). Here, we can use noisy edges output by our DPRR as input to their algorithms. In other words, we can combine our DPRR with the algorithms in [52] by replacing the original edges with the noisy edges output by our DPRR. Since the noisy edges provide LDP, we can provide LDP for both feature vectors and edges. We also note that although the algorithms in [52] focus on node classification, they can be easily applied to graph classification by using the mean readout as a graph pooling method, as in our experiments. For future work, we would like to evaluate the accuracy of the combined algorithms using attributed graphs.

Acknowledgements

This study was supported in part by JSPS KAKENHI 22H00521.

References

[1] AI bridging cloud infrastructure (ABCI). https://abci.ai/, 2023.

[2] The diaspora* project. https://diasporafoundation.org/, 2023.

[3] How powerful are graph neural networks? https://github.com/weihua916/powerful-gnns, 2023.

[4] Linkedin. https://www.linkedin.com/company/social-network, 2023.

[5] Mastodon: Giving social networking back to you. https://joinmastodon.org/, 2023.

[6] Reddit. https://www.reddit.com/, 2023.

[7] Smule. https://www.smule.com/, 2023.

[8] Tools: DPRR-GNN. https://github.com/DPRR-GNN/DPRR-GNN, 2023.

[9] Jayadev Acharya, Kallista Bonawitz, Peter Kairouz, Daniel Ramage, and Ziteng Sun. Context-aware local differential privacy. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, pages 52–62, 2020.

[10] Jayadev Acharya, Ziteng Sun, and Huanyu Zhang. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS'19)*, pages 1120–1129, 2019.

[11] Albert-László Barabási. *Network Science*. Cambridge University Press, 2016.

[12] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the 47th annual ACM Symposium on Theory of Computing (STOC'15)*, pages 127–135, 2015.

[13] Suman K. Bera and C. Seshadhri. How to count triangles, without seeing the whole graph. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'20)*, pages 306–316, 2020.

[14] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. In *Proceedings of the 30th Usenix Security Symposium (Usenix Security'21)*, pages 947–964, 2021.

[15] Rosalie Chan. The cambridge analytica whistleblower explains how the firm used facebook data to sway elections. https://www.businessinsider.com/cambridge-analytica-whistleblower-christopher-wylie-facebook-data-2019-10, 2019.

[16] Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*, pages 1–24, 2019.

[17] Albert Cheu, Adam Smith, and Jonathan Ullman. Manipulation attacks in local differential privacy. In *Proceedings of the 2021 IEEE Symposium on Security and Privacy (S&P'21)*, pages 883–900, 2021.

[18] Ben Coleman. Releasing counts with differential privacy. https://randorithms.com/2020/10/09/geometric-mechanism.html, 2020.

[19] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS'17)*, pages 3574–3583, 2017.

[20] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[21] Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS'15)*, pages 614–633, 2015.

[22] Ulfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*, pages 1054–1067, 2014.

[23] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR'20)*, pages 1–16, 2020.

[24] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pages 1263–1272, 2017.

[25] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference (SciPy'08)*, pages 11–15, 2008.

[26] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS'17)*, pages 1024–1034, 2017.

[27] Kaveh Hassani. Cross-domain few-shot graph classification. In *Proceedings of the 36th AAAI*

Conference on Artificial Intelligence (AAAI'22), 2022.

[28] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing links from graph neural networks. In *Proceedings of the 30th Usenix Security Symposium (Usenix Security'21)*, pages 2669–2686, 2021.

[29] Nobuaki Hoshino. A firm foundation for statistical disclosure control. *Japanese Journal of Statistics and Data Science*, 3:721–746, 2020.

[30] Jacob Imola, Amrita Roy Chowdhury, and Kamalika Chaudhuri. Robustness of locally differentially private graph analysis against poisoning. *CoRR*, 2210.14376, 2022.

[31] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Locally differentially private analysis of graph statistics. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security'21)*, pages 983–1000, 2021.

[32] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Communication-efficient triangle counting under local differential privacy. In *Proceedings of the 31th USENIX Security Symposium (USENIX Security'22)*, 2022.

[33] Xun Jian, Yue Wang, and Lei Chen. Publishing graphs under node differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4164–4177, 2023.

[34] Hongwei Jin and Xun Chen. Gromov-wasserstein discrepancy with local differential privacy for distributed structural raphs. *CoRR*, 2202.00808, 2022.

[35] Dave Johnson. How to hide your friends list on facebook, from everyone or only certain people. https://www.businessinsider.com/how-to-hide-friends-on-facebook, 2019.

[36] Zach Jorgensen, Ting Yu, and Graham Cormode. Conservative or liberal? Personalized differential privacy. In *Proceedings of IEEE 31st International Conference on Data Engineering (ICDE'15)*, pages 1023–1034, 2015.

[37] Peter Kairouz, H. Brendan McMahan, and Brendan Avent *et al.* Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.

[38] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*, pages 1–14, 2017.

[39] Ninghui Li, Min Lyu, and Dong Su. *Differential Privacy: From Theory to Practice*. Morgan & Claypool Publishers, 2016.

[40] Wanyu Lin, Baochun Li, and Cong Wang. Towards private learning on decentralized graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security*, 17:2936–2946, 2022.

[41] Yao Ma and Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021.

[42] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017.

[43] Chris Morris. The number of data breaches in 2021 has already surpassed last year's total. `https://fortune.com/2021/10/06/data-breach-2021-2020-total-hacks/`, 2021.

[44] Tamara T. Mueller, Johannes C. Paetzold, Chinmay Prabhakar, Dmitrii Usynin, Daniel Rueckert, and Georgios Kaissis. Differentially private graph classification with GNNs. *CoRR*, 2202.02575, 2022.

[45] Hiep H. Nguyen, Abdessamad Imine, and Michaël Rusinowitch. Network structure release under differential privacy. *Transactions on Data Privacy*, 9(3):215–214, 2016.

[46] Iyiola E. Olatunji, Thorben Funke, and Megha Khosla. Releasing graph neural networks with differential privacy guarantees. *CoRR*, 2109.08907, 2021.

[47] Thomas Paul, Antonino Famulari, and Thorsten Strufe. A survey on decentralized online social networks. *Computer Networks*, 75:437–452, 2014.

[48] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pages 192–203, 2016.

[49] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 425–438, 2017.

[50] Sofya Raskhodnikova and Adam Smith. *Differentially Private Analysis of Graphs*, pages 543–547. Springer, 2016.

[51] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate club: An api oriented open-source python framework for unsupervised learning on graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM'20)*, page 3125–3132, 2020.

[52] Sina Sajadmanesh and Daniel Gatica-Perez. Locally private graph neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*, pages 2130–2145, 2021.

[53] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez. GAP: Differentially private graph neural networks with aggregation perturbation. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security'23)*, 2023.

[54] Julián Salas, Vladimiro González-Zelaya, Vicenç Torra, and David Megías. Differentially private graph publishing through noise-graph addition. In *Proceedings of the 20th International Conference on Modeling Decisions for Artificial Intelligence (MDAI'23)*, pages 253–264, 2023.

[55] Comandur Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of erdös-rényi graphs. *Physical Review E*, 85(5):1–9, 2012.

[56] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*, pages 1310–1321, 2015.

[57] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security'17)*, pages 729–745, 2017.

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

[59] Bin Wu, Ke Yi, and Zhenguo Li. Counting triangles in large graphs by random sampling. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2013–2026, 2016.

[60] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. FedGNN: Federated graph neural network for privacy-preserving recommendation. *CoRR*, 2102.04925, 2021.

[61] Fan Wu, Yunhui Long, Ce Zhang, and Bo Li. LinkTeller: Recovering private edges from graph neural networks via influence analysis. In *Proceedings of the 2022 IEEE Symposium on Security and Privacy (S&P'22)*, pages 522–541, 2022.

[62] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, 1901.00596, 2019.

[63] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*, pages 1–17, 2019.

[64] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*, pages 1365–1374, 2015.

[65] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the 2012 Ninth IEEE International Conference on Data Mining (ICDM'12)*, pages 745–754, 2012.

[66] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering (Early Access)*, pages 1–16, 2021.

[67] Jiaxin Ying, Jiaqi Ma, and Qiaozhu Mei. A simple yet effective method improving graph fingerprints for graph-level prediction. In *Proceedings of the 2021 Workshop on Graph Learning Benchmarks (GLB'21)*, pages 1–9, 2021.

[68] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*, pages 4805–4815, 2018.

[69] Quan Yuan, Zhikun Zhang, Linkang Du, Min Chen, Peng Cheng, and Mingyang Sun. PrivGraph: Differentially private graph data publication by exploiting community information. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security'23)*, 2023.

[70] Zaixi Zhang, Qi Liu, Zhenya Huang, Hao Wang, Chengqiang Lu, Chuanren Liu, and Enhong Chen. GraphMI: Extracting private graph data from graph neural networks. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 3749–3755, 2021.

[71] Zhikun Zhang, Min Chen, Michael Backes, Yun Shen, and Yang Zhang. Inference attacks against graph neural networks. In *Proceedings of the 31th USENIX Security Symposium (USENIX Security'22)*, 2022.

A Proof of Proposition 5

Adding or removing one bit of $\mathbf{a}_i$ will change a degree d_i of user v_i by one. Thus, the global sensitivity of the degree is 1, and adding $\mathrm{Lap}(\frac{1}{\varepsilon_1})$ to d_i (line 2 in Algorithm 5) provides ε_1-edge LDP. The subsequent sampling probability tuning (lines 4-5) is a post-processing on the noisy degree d_i^* ($= d_i + \mathrm{Lap}(\frac{1}{\varepsilon_1})$). In addition, Warner's RR with the flipping probability $1-p = \frac{1}{e^{\varepsilon_2}+1}$ (line 6) provides ε_2-edge LDP, as described in Section 3.2. The subsequent edge sampling (line 7) is a post-processing on the noisy neighbor list $\tilde{\mathbf{a}}_i$.

Finally, we use the (general) sequential composition [39] of edge LDP, which is proved in [32]:

Lemma 7 (Sequential composition of edge LDP [32])**.** *For $i \in [n]$, let $\mathcal{R}_i^1$ be a local randomizer of user v_i that takes $\mathbf{a}_i \in \{0,1\}^n$ as input. Let $\mathcal{R}_i^2$ be a local randomizer of v_i that depends on the output $\mathcal{R}_i^1(\mathbf{a}_i)$ of $\mathcal{R}_i^1$. If $\mathcal{R}_i^1$ provides ε_1-edge LDP and for any $\mathcal{R}_i^1(\mathbf{a}_i)$, $\mathcal{R}_i^2(\mathcal{R}_i^1(\mathbf{a}_i))$ provides ε_2-edge LDP, then the sequential composition $(\mathcal{R}_i^1(\mathbf{a}_i), \mathcal{R}_i^2(\mathcal{R}_i^1(\mathbf{a}_i))(\mathbf{a}_i))$ provides $(\varepsilon_1+\varepsilon_2)$-edge LDP.*

In our case, $\mathcal{R}_i^1$ is the Laplacian mechanism followed by the sampling probability tuning, and $\mathcal{R}_i^2$ is Warner's RR followed by the edge sampling. $\mathcal{R}_i^2$ depends on the output q_i of $\mathcal{R}_i^1$. Thus, by Lemma 7 (and the post-processing invariance), the DPRR provides $(\varepsilon_1+\varepsilon_2)$-edge LDP. □

B Proof of Proposition 6

Assume that user v_i is honest. The number $\tilde{d}_i$ ($= ||\tilde{\mathbf{a}}_i||_1$) of 1s in the noisy neighbor list $\tilde{\mathbf{a}}_i$ is maximized when the original neighbor list $\mathbf{a}_i$ is an all-ones vector $\mathbf{a}_i = (1, 1, \ldots, 1)$ (except for the i-th element $a_{i,i} = 0$) and the sampling probability q_i is $q_i = 1$. In this case, each element of the noisy neighbor list $\tilde{\mathbf{a}}_i$ (except for the i-th one) follows the Bernoulli

distribution with parameter $p = \frac{e^{\varepsilon_2}}{e^{\varepsilon_2}+1}$. Thus, we can use the multiplicative Chernoff bound [42] as follows:

$$\Pr(\tilde{d}_i \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2+\delta}}, \tag{18}$$

where $\mu = (n-1)p$ and $\delta \geq 0$.

Let $\theta = e^{-\frac{\delta^2\mu}{2+\delta}}$. Then, we have

$$\begin{aligned} \theta = e^{-\frac{\delta^2\mu}{2+\delta}} &\iff \log\frac{1}{\theta} = \frac{\delta^2\mu}{2+\delta} \\ &\iff \mu\delta^2 - \left(\log\frac{1}{\theta}\right)\delta - 2\left(\log\frac{1}{\theta}\right) = 0 \\ &\iff \delta = \frac{\log\frac{1}{\theta} + \sqrt{(\log\frac{1}{\theta})^2 + 8\mu\log\frac{1}{\theta}}}{2\mu} \quad \text{(as } \delta \geq 0\text{)}. \end{aligned} \tag{19}$$

By (18) and (19), we have

$$\Pr(\tilde{d}_i \geq \tau) \leq \theta,$$

where τ is given by (17). □