Generating Synthetic Decentralized Social Graphs with Local Differential Privacy

Zhan Qin[1,2], Ting Yu[2], Yin Yang[3], Issa Khalil[2], Xiaokui Xiao[4], Kui Ren[1*]

[1]Dept of Computer Science and Engineering
SUNY at Buffalo
USA

[2]Qatar Computing Research Institute
Hamad Bin Khalifa University
Qatar

[3]College of Science and Engineering
Hamad Bin Khalifa University
Qatar

[4]School of Computer Science and Engineering
Nanyang Technological University
Singapore

ABSTRACT

A large amount of valuable information resides in decentralized social graphs, where no entity has access to the complete graph structure. Instead, each user maintains locally a limited view of the graph. For example, in a phone network, each user keeps a contact list locally in her phone, and does not have access to other users' contacts. The contact lists of all users form an implicit social graph that could be very useful to study the interaction patterns among different populations. However, due to privacy concerns, one could not simply collect the unfettered local views from users and reconstruct a decentralized social network.

In this paper, we investigate techniques to ensure local differential privacy of individuals while collecting structural information and generating representative synthetic social graphs. We show that existing local differential privacy and synthetic graph generation techniques are insufficient for preserving important graph properties, due to excessive noise injection, inability to retain important graph structure, or both. Motivated by this, we propose LDPGen, a novel multi-phase technique that incrementally clusters users based on their connections to different partitions of the whole population. Every time a user reports information, LDPGen carefully injects noise to ensure local differential privacy. We derive optimal parameters in this process to cluster structurally-similar users together. Once a good clustering of users is obtained, LDPGen adapts existing social graph generation models to construct a synthetic social graph.

We conduct comprehensive experiments over four real datasets to evaluate the quality of the obtained synthetic graphs, using a variety of metrics, including (i) important graph structural measures; (ii) quality of community discovery; and (iii) applicability in social recommendation. Our experiments show that the proposed technique produces high-quality synthetic graphs that well represent the original decentralized social graphs, and significantly outperform those from baseline approaches.

KEYWORDS

decentralized social networks; synthetic graph generation; local differential privacy; community discovery

1 INTRODUCTION

With the advances of graph analytics, much valuable knowledge can be obtained by mining a social graph, which contains information about the relationships and interactions of people. Such information, however, can be sensitive and private, e.g., not everyone is comfortable to release her contact list to strangers. For online social networks where the whole social graph is available to a single party, there exist solutions that publish graph data [23, 42, 45] or analysis results [24, 32, 36, 46] under certain privacy guarantees such as differential privacy [11]. The problem is far more challenging if the graph is *decentralized*, meaning that no party has access to the whole graph. This happens for many sensitive social graphs in the physical world. For instance, consider distributed social networks, e.g., Synereo [27]. Clearly, for such a graph (i) everyone has a local view (e.g., those with direct relationship with oneself), and (ii) it is infeasible to collect the whole graph unfettered as a single dataset. In fact, even for less sensitive relationships such as face-to-face interactions and phone contacts, collecting a decentralized social graph is difficult as people tend to resist revealing private relationships. Additionally, a social graph (e.g., a phone call network or email communications) is effectively decentralized when the party that possesses the whole graph (the telephone/email service providers) does not cooperate with the researchers trying to analyze the data. In these situations, existing solutions for privacy-preserving graph publication and analysis do not apply, since the data cannot be collected in the first place.

*Majority of this work was conducted while the first author was doing internship at Qatar Computing Research Institute.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134086

Meanwhile, clearly there is valuable knowledge that can be extracted by analyzing a decentralized social graph, and such knowledge might not be easily obtained by analyzing online social networks. For example, relationships in the physical world (e.g., friends that we go out with) can be very different from those online (e.g., IDs that we chat with), and so are communities (e.g., parents associations vs. game fan clubs). To gain knowledge on decentralized social graphs, it is essential to collect sensitive local views with strong privacy guarantees. This study focuses on generating a synthetic social graph from a real, decentralized one, under local differential privacy [25] (explained in Section 2), a strong privacy standard that has been used in notable systems such as Google Chrome [16] and Apple iOS [2]. Such a synthetic graph enables data scientists to draw meaning insights, while protecting the participants involved and the data collector itself against risks of privacy violations.

One major challenge in this study is that the state-of-the-art of local differential privacy research has been limited to collecting simple, statistical information such as counts [25], histograms [16] and heavy hitters [2]. In our problem, a large-scale graph needs to be collected with detailed edge-level information. As will become clear later, collecting data at such a fine granularity (e.g., neighbor lists) requires heavy noise injection in order to satisfy local differential privacy, which may render the graph too distorted to be useful. On the other hand, if we only collect graph statistics (e.g., node degrees) and generate a synthetic graph from only such statistics (e.g., using BTER [43]), the resulting synthetic graph may not preserve important properties of the original graph, other than the statistics from which it is generated, as shown in our experiments.

In this paper we propose LDPGen, a novel, multi-phase approach to generating synthetic decentralized social graphs under local differential privacy. One key idea is that LDPGen captures the structure of the original decentralized graph, by incrementally identifying and refining clusters of connected nodes under local differential privacy. To do so, LDPGen iteratively partitions nodes into groups, collects information on node-to-group connectivity under local differential privacy, and clusters nodes according to such information. After obtaining such node clusters, LDPGen applies a graph generation model that utilizes such clusters to generate a representative synthetic social graph. In addition, we describe techniques to optimize key parameters of LDPGen to improve the utility of the generated synthetic social graph.

To validate the effectiveness of LDPGen, we present an extensive set of experiments using four real social graphs in various domains, and three different use cases: (i) statistical analysis of the social graph structure, (ii) community discovery [5] and (iii) social recommendation [31]. The evaluation results show that synthetic social graphs generated using LDPGen obtains high utility for all use cases and datasets, whereas baseline solutions fail to obtain competitive utility in most settings except for the few that they are specifically optimized for.

Our main contributions are summarized as follows:

- We formulate the problem of synthetic data generation of decentralized social graphs under local differential privacy. To our knowledge, this is the first effort in the literature to define and tackle this problem.
- We describe straw-man approaches that rely on existing local differential privacy and synthetic graph generation techniques, and analyze in detail their limitations.
- We propose LDPGen, a novel and effective multi-phase approach to synthetic decentralized social graph generation, and describe methods for optimizing key parameters.
- We conduct a comprehensive experimental study using several real datasets and use cases, and the results demonstrate that LDPGen is capable of generating high-utility synthetic graphs.

In the following, Section 2 provides background on local differential privacy. Section 3 defines the problem of synthetic decentralized social graph generation under local differential privacy, and discusses two straw-man solutions. Section 4 presents the proposed solution, LDPGen, and proves that it satisfies local differential privacy. Section 5 contains an extensive set of experiments, Section 6 reviews related work, and Section 7 concludes with future directions.

2 LOCAL DIFFERNETIAL PRIVACY

The concept of local different privacy (LDP) [25] was recently proposed as a strong privacy measure such that sensitive information of individuals is kept private even from data collectors. Different from the setting of the classical differential privacy model [10], where privacy is guaranteed in the data analysis and publishing process, local differential privacy focuses on the data collection process. Specifically, under local differential privacy, each data contributor locally perturbs her own data using a randomized mechanism, and then sends the noisy version of her data to a data collector. In the following, we first provide a brief review of differential privacy, and then introduce local differential privacy in the context of social networks.

We say a randomized mechanism $\mathcal{M}$ satisfies ϵ-differential privacy, if and only if for any two neighboring databases D and D' that differ in exactly one record, and any possible $s \in range(\mathcal{M})$, we have $\frac{Pr[\mathcal{M}(D)=s]}{Pr[\mathcal{M}(D')=s]} \leq e^{\epsilon}$. Differential privacy ensures that from any output s of the mechanism $\mathcal{M}$, an attacker cannot infer with high confidence whether the input database is D or D'. The strength of privacy protection is controlled by the system parameter ϵ, which is often referred to as the *privacy budget*. Clearly, the smaller ϵ is, the closer are the distributions of $\mathcal{M}(D)$ and $\mathcal{M}(D')$, and thus the higher privacy is offered by $\mathcal{M}$.

Differential privacy was originally designed for a *centralized* setting, in which a trusted data curator processes a database with the exact data records of multiple users, and publishes perturbed statistics or other data analysis results from the database using a randomized mechanism. In the *local differential privacy* setting, which is the focus of this paper, the data curator is not trusted; instead, each user perturbs her data locally with a differentially private mechanism. In this setting, the input database D would only contain the data of a single user. What constitutes a neighboring database D' of D depends on the type of the user's data and the goal of privacy protection. For example, a user's data could be the homepage of her browser [16]. Since the user would not want the data collector to infer with high confidence her actual homepage, no matter what that homepage is, the neighboring database would then be any other arbitrary website. In this context, for mechanism $\mathcal{M}$ to satisfy local differential privacy, it needs to ensure that, for any two websites w and w' and for any $s \in range(\mathcal{M})$, $\frac{Pr[\mathcal{M}(w)=s]}{Pr[\mathcal{M}(w')=s]} \leq e^{\epsilon}$.

In the context of social graphs, depending on the privacy requirement, a privacy mechanism can be designed to satisfy either *edge differential privacy* [4] or *node differential privacy* [26]. The former ensures that a randomized mechanism does not reveal the inclusion or removal of a particular edge of an individual, while the latter hides the inclusion or removal of a node together with all its adjacent edges. For local differential privacy, we have the following formal definitions. Let $U = \{u_1, \ldots, u_n\}$ be the set of all users in a social network. A user u's neighbor list can be represented as an n-dimensional bit vector $(b_1, \ldots, b_n)$, i.e., $b_i = 1$, $i = 1, \ldots, n$, if and only if there is an edge (u, u_i) in the social graph; otherwise $b_i = 0$.

Definition 2.1 (Node local differential privacy.). A randomized mechanism $\mathcal{M}$ satisfies ϵ-node local differential privacy (ϵ-node LDP) if and only if for any two neighbor lists γ and γ' and any $s \in range(\mathcal{M})$, we have $\frac{Pr[\mathcal{M}(\gamma)=s]}{Pr[\mathcal{M}(\gamma')=s]} \leq e^{\epsilon}$.

Definition 2.2 (Edge local differential privacy.). A randomized mechanism $\mathcal{M}$ satisfies ϵ-edge local differential privacy (ϵ-edge LDP) if and only if for any two neighbor lists γ and γ', such that γ and γ' only differ in one bit, and any $s \in range(\mathcal{M})$, we have $\frac{Pr[\mathcal{M}(\gamma)=s]}{Pr[\mathcal{M}(\gamma')=s]} \leq e^{\epsilon}$.

Local differential privacy is also composable: given t mechanisms $\mathcal{M}_i$, $i = 1, \ldots, t$, each of which satisfies ϵ_i-edge (or node) local differential privacy, the sequence of $\mathcal{M}_i(v)$ satisfies $\left(\sum_{i=1}^{t}\right) \epsilon_t$-edge (node) local differential privacy.

Node LDP is clearly a much stronger privacy guarantee than edge LDP (in fact node LDP implies edge LDP). For example, as we will see later, under edge LDP a user could still reveal her degree with reasonable accuracy, which would be almost impossible under node LDP, even with a fairly large privacy budget. Depending on the application and the nature of a social graph, one privacy model may be more appropriate than the other. For instance, when collecting the contact list of a user, edge LDP could be sufficient, as a user usually does not mind sharing the number of her contacts. What she wants to protect is who exactly are in her contacts. For some other social graphs (e.g., the sexual relationship graph), even a user's degree could be highly sensitive, and thus node LDP should be adopted. We note that the strong privacy guarantee of node LDP comes with a hefty price in terms of utility. Even in the global privacy setting, node differential privacy is hard to achieve without significant negative impacts on the utility of social graph data [26]. It would be even more challenging to do so in the local privacy setting. In this paper, we focus on achieving edge LDP and designing techniques to facilitate social graph analysis.

3 PROBLEM DESCRIPTION AND STRAW-MAN APPROACHES

Consider a decentralized social network G with users $U = \{u_1, \ldots, u_n\}$. Without loss of generality, we assume G is a directed graph; for an undirected graph, we simply view edge undirected edge as two directed edges. Each user u maintains locally a neighbor list, which, as mentioned before, could be modeled as an n-dimensional binary vector. Our problem is to design techniques such that (i) an untrusted data curator is able to collect information from each individual user while satisfying edge local differential privacy; and (ii) from the collected data the data curator is able to construct a representative synthetic graph of G. The representativeness of the synthetic graph could be reflected from different angles, which we discuss in detail in section 5.Note that this work focuses on the weaker edge-LDP definition since the problem is already highly challenging under edge-LDP. Node differential privacy is vastly more difficult, even in the centralized setting, and it might not lead to a meaningful utility in our target applications such as community discovery. Thus, we leave node LDP as our future work.

In the following, we discuss two simple solutions based on adaptations of existing local differential privacy and synthetic social graph generation techniques.

3.1 Randomized Neighbor List Approach

A common methodology for enforcing local differential privacy is randomized response [13]. Our first straw-man approach, namely randomized neighbor list (RNL), directly applies randomized response to collect neighbor lists from users. Specifically, in RNL, given a privacy budget ϵ, each user flips each bit in her neighbor list with probability $p = \frac{1}{1+e^{\epsilon}}$, and sends the perturbed neighbor list to the data curator. The latter then combine the noisy neighbor lists from all users together to form a synthetic social graph.

Theorem 3.1. *The randomized neighbor list approach satisfies ϵ-edge local differential privacy.*

Proof. Denote the randomized neighbor list mechanism as $\mathcal{M}$, and denote $Pr[x \rightarrow y]$ the probability that $x \in \{0, 1\}$ becomes $y \in \{0, 1\}$ after a random bit flipping. Let $q = 1 - p = \frac{e^{\epsilon}}{1+e^{\epsilon}}$. Since $\epsilon > 0$, we have $q > p$.

Let $\gamma = (b_1, \ldots, b_n)$ and $\gamma' = (b'_1, \ldots, b'_n)$ be two neighbor lists that differ in only one bit. Without loss of generality, assume $b_1 \neq b'_1$. Given any output $s = (s_1, \ldots, s_n)$ from $\mathcal{M}$, we have

$$\frac{Pr[\mathcal{M}(v)=s]}{\Pr[\mathcal{M}(v')=s]} = \frac{Pr[\gamma_1 \to s_1] \ldots Pr[\gamma_n \to s_n]}{Pr[\gamma'_1 \to s_1] \ldots Pr[\gamma'_n \to s_n]} = \frac{Pr[\gamma_1 \to s_1]}{Pr[\gamma'_1 \to s_1]} < \frac{q}{p} = e^{\epsilon}$$

□

Although RNL achieves our privacy goal, the resulting synthetic graph it generates does not represent well the original decentralized social graph. One obvious problem is that it tends to return a much denser graph than the original one. In general, a real social graph tends to be sparse: even hub users with numerous connections have degrees much smaller compared to the whole population of the social network. Hence, a real neighbor lists generally has much more 0s than 1s. After random bit flipping, however, even with a relatively large privacy budget, the number of 1s would significantly increase, leading to a significantly denser synthetic graph. In particular, if the density of original graph is r, after applying the randomized neighbor list approach, the expected density of the obtained synthetic graph becomes $(1 - p)r + (1 - r)p$. Consider the Enron email graph (detailed in section 5), whose original density is 0.01%. Even with a fairly small bit flipping probability $p = 0.01$, the expected density of the graph after neighbor list randomization would become 2%, an increase of 200 times.

3.2 Degree-based Graph Generation Approach

In the social computing literature, there have been many existing synthetic social graph generation algorithms, such as Erdos-Renyi [14], Chung-Lu [1] and BTER [43]. Usually, such an algorithm takes as input some graph statistical information such as node degrees, and generates a synthetic graph based on a social graph model. In other words, *the algorithm brings in prior knowledge about social graphs in the form of a graph model*, and adapts such knowledge to the high-level structural properties of the input graph. The idea of degree-based graph generation (DGG) is to apply such a social graph generation module to our problem.

Note that not all synthetic social graph generation algorithms can be applied to our problem. The reason is that in our setting, each user only has a limited local view of the graph in the form of a neighbor list. On the other hand, some graph generation algorithms requires global information of the entire graph, e.g., the submatrix of the adjacency matrix in Kronecker Graph model [28]; thus, such algorithms cannot be used in DGG. Our implementation of DGG is based on an adapted version of BTER [43], described below.

Using DGG, each user calculates her node degree, perturbs the degree under ϵ-edge differential privacy (e.g., using the Laplace mechanism [10]), and sends the resulting noisy degree to the data curator. The latter collects such perturbed degrees from all users, and runs the BTER algorithm to generate a synthetic graph. Specifically, BTER first forms node clusters based on their degrees. In particular, nodes with similar degrees are clustered together. The size of a cluster is also determined by the degrees of nodes in it: the larger the node degrees, the larger the cluster size. After that, for each cluster, BTER generates random intra-cluster edges whose number depends on both the node degrees in the cluster and a connectivity parameter, which is set to a default value in DGG due to the lack of global graph statistics. Finally, BTER generates inter-cluster edges, based on the remaining degrees of each node and the size of each cluster.

DGG clearly satisfies our privacy requirement, since each user only sends her perturbed degree to the curator, which is randomized under ϵ-edge local differential privacy. Meanwhile, for reasonably large values of ϵ, the perturbed degree is expected to be close to its true value, since the noise injected with the Laplace mechanism has a variance of $\frac{1}{\epsilon^2}$ [11]. Hence, the synthetic graph generated with DGG accurately captures node degrees. However, since DGG collects exclusively node degrees, it loses all other information of the underlying graph. For instance, two users who have similar degrees but are far apart in the original graph could be placed in the same cluster in the synthetic graph. Furthermore, DGG often fails to capture other aspects of graph structure besides node degrees, as we show in our experiments in Section 5.

3.3 Observations

We observe that the two straw-man approaches RNL and DGG described above represent two extremes of data collection: RNL collects fine-grained information (i.e., neighbor lists), and pays the price of heavy perturbations required to satisfy local differential privacy. DGG, on the other hand, collects only coarse-grained statistics (i.e., node degrees) accurately since they only require a small amount of noise to satisfy local differential privacy, but it also loses important details of the underlying graph. Another interesting observation is that DGG brings in prior knowledge about social graphs, whereas RNL does not.

The above observation suggests the need to strike a balance between noise added to satisfy differential privacy, and information loss due to collecting information at a coarser granularity. One tricky issue is that this balance itself is data dependent, and, thus, may reveal private information. The proposed approach LDPGen, described next, iteratively finds such a balance under edge local privacy constraints. Furthermore, similar to DGG, it utilizes prior knowledge of social graphs to enhance the synthetic graph.

4 LDPGEN

Section 4.1 overviews the general framework of the proposed solution LDPGen, which involves three phases. Sections 4.2-4.4 instantiate these three phases, respectively. Section 4.5 proves that LDPGen satisfies our privacy requirement. Section 4.7 provides additional discussions on the algorithmic design of LDPGen.

4.1 General Framework

LDPGen is based on the following idea. Suppose that we have a way to partition all users in the social graph into k disjoint groups $\xi = \{U_1, \ldots, U_k\}$. Then for each user u we could define a degree vector $\delta^u = (\delta_1^u, \ldots, \delta_k^u)$, where δ_i^u $(i = 1, \ldots, n)$ is the number of u's neighbors in U_i. For instance, consider that all users are divided into two disjoint groups U_1 and U_2. Suppose that a user u has 5 (resp. 7) connections with nodes in U_1 (resp. U_2). Then, the degree vector for u is (5, 7). Clearly, if the neighbor lists of two users are similar, so would be their degree vectors. Thus, a data curator could collect each user's degree vector using a local differentially private mechanism, and use these vectors to generate a synthetic graph.

In fact, both straw-man approaches RNL and DGG described in Section 3 can be viewed as extreme cases of the above general framework. In particular, RNL lies at one extreme when each partition contains exactly one user (i.e., when $k = n$, where k and n are the number of partitions and users in the graph, respectively). Meanwhile, DGG lies at the opposite extreme, when the entire graph is considered as a single partition (i.e., $k = 1$), and the degree vector for each node degenerates into a single node degree. As explained in Section 3, neither approach is ideal: RNL incurs excessive noise whereas DGG incurs excessive information loss. LDPGen follows the same general framework, and strike a good balance between noise and information loss by choosing an appropriate user partitioning scheme.

The main challenge for choosing an appropriate user partitioning scheme, however, is that the best user partitioning scheme depends on the dataset: intuitively, similar users should be grouped together. This leads to a circular dependency: (i) to collect data, we must know the value of k and the partitioning scheme; (ii) to decide on the best value of k, we must first collect data. LDPGen addresses this issue with a multi-phase design. In particular, in LDPGen the data curator collects data from the users in multiple rounds; each round refines the user partitioning scheme, and then collects data from users again with higher accuracy to be used in the next round, e.g., by using a larger portion of the privacy budget. Through this process, users who are structurally close would be gradually grouped together.

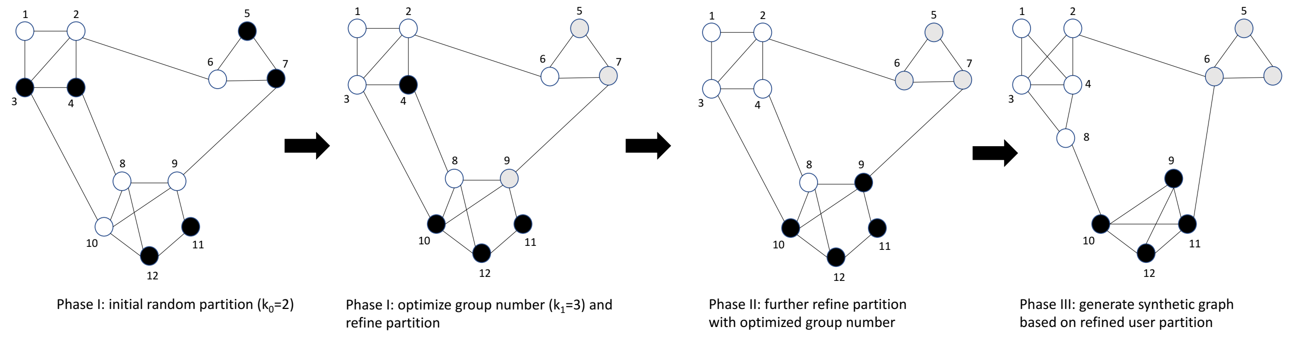

1
2
3
4
5
6
7
8
9
10
11
12
Phase I: initial random partition (k_0=2)
Phase I: optimize group number (k_1=3) and refine partition
Phase II: further refine partition with optimized group number
Phase III: generate synthetic graph based on refined user partition

Figure 1: Illustrative example of LDPGen

Once a good grouping is obtained, LDPGen then applies a social graph generation model to produce a synthetic social network. Figure 1 illustrates an example of LDPGen, in which the users are partitioned into three groups, with colors black, white and gray, respectively. Initially, the grouping is rather random, and there are only two groups whereas three user clusters can be observed in the graph. Later on, the grouping is refined, in the sense that (i) the users are partitioned into 3 groups instead of 2 and (ii) similar nodes are gradually moved to the same cluster.

Specifically, our design of LDPGen is composed of three phases: *initial grouping*, *grouping refinement* and *graph generation*. The first two phases involve data collection from users. We allocate privacy budgets ϵ_1 and ϵ_2 respectively for each phase, such that $\epsilon_1 + \epsilon_2 = \epsilon$. Due to the sequential composition property of differential privacy [11], these two phases together satisfy ϵ-edge LDP. The choice of ϵ_1 and ϵ_2 is discussed later in Section 4.7. The graph generation phase is a post processing of the output of an edge-LDP mechanism, and thus does not consume any privacy budget.

Phase I: initial grouping. At this stage, the data curator does not have any information about the graph structure. Thus, it randomly partitions users into k_0 equal-size groups (where k_0 is a system parameter, discussed further in Section 4.7), and communicates this grouping scheme ξ_0 to all users. Efficient communications between the data curator and the users are further discussed in Section 4.7. After that, each user forms her degree vector based on ξ_0, adds noise according to the allocated privacy budget ϵ_1 for Phase I, and sends the noisy degree vector to the data curator. Once the data curator receives the noisy degree vectors from all users, it computes a new grouping scheme ξ_1 with k_1 partitions, using (i) an estimated degree distribution of all users, obtained from the collected degree vectors, and (ii) the privacy budget ϵ_2 allocated to Phase II. Note that unlike the initial grouping, the new partitions may be of different sizes. This new partition ξ_1 is communicated back to users.

Phase II: grouping refinement. In this phase, users now report again noisy degree vectors based on ξ_1 and the allocated privacy budget ϵ_2 for Phase II. The data curator conducts another round of user grouping, and partition users into k_1 clusters based on their second-round noisy degree vectors. Note that Phase II could be repeated multiple times, each incrementally refines the grouping based on the degree vectors collected in the previous rounds. This issue is discussed further in Section 4.7.

Phase III: graph generation. In this step, the data curator adapts the BTER model to generate a synthetic social graph. Unlike in DGG where users are clustered based solely on their degrees, in LDPGen the data curator starts with the user partitions obtained in Phase II, and generates edges using the collected degree vectors. In the example of Figure 1, LDPGen starts with a random partitioning of users into $k_0 = 2$ groups in Phase I (leftmost). Then, the data curator collects degree vectors from the users, and refine the user grouping according (second left). After that, in Phase II, the user grouping is refined again with another round of data collection. Finally, in Phase III (rightmost), a synthetic graph is generated based on the user grouping and degree vectors. Next, we present in detail the design of each phase.

4.2 Design of Phase I

As mentioned above, the data curator first randomly partitions all users into k_0 equal-sized groups. We call this partition ξ_0. Each user u then computes her degree vector $\delta^u = (\delta^u_1, \ldots, \delta^u_{k_0})$ based on ξ_0. To ensure edge LDP, u adds to each δ^u_i, $i = 1, \ldots, k_0$ a random noise drawn from the Laplace distribution $Lap(0, \frac{1}{\epsilon_1})$, where ϵ_1 is the privacy budget allocated to Phase I. Let the resulting noisy degree vector be $(\tilde{\delta}^u_1, \ldots, \tilde{\delta}^u_{k_0})$, which is shared with the data curator. Intuitively, since these k_0 groups are non-overlapping, adding or removing one edge from a user would change the value of exactly one δ^u_i by 1 in her degree vector. By the parallel composition property of DP [11], sharing $(\tilde{\delta}^u_1, \ldots, \tilde{\delta}^u_{k_0})$ still satisfies ϵ_1-edge LDP. We provide a formal proof of this privacy guarantee in Section 4.5.

From the noisy degree vector, the data curator derives an unbiased degree estimator $\hat{\eta}$ for u:

$$\hat{\eta}^u = \left|\tilde{\delta}^u\right| = \sum_{i=1}^{k} \tilde{\delta}^u_i \tag{1}$$

The main task in Phase I is to come up with a good user grouping for the next phase. To do so, we need to determine (1) an appropriate number k_1 of groups; (2) how to partition users into k_1 groups. To solve the first problem, we need to define a target function, the optimization of which would lead to the preservation of a graph's structural information. Recall that one of the most important information of a social network is its community structure. It is thus desirable to ensure that nodes with similar (dissimilar) neighbor lists are still likely to have similar (dissimilar) degree vectors after the partition. There are many ways to measure the similarity of two vectors. In our design, we choose $\mathcal{L}_1$ distance as the similarity function.

For ease of analysis, consider two users u and v with the same degree (i.e., $\eta^u = \eta^v$), and let $d_{u,v}$, $\hat{d}_{u,v}$, and $\tilde{d}_{u,v}$ denote the $\mathcal{L}_1$ distances between their neighbor lists, degree vectors, and noisy degree vectors, respectively (and d, $\hat{d}$, and $\tilde{d}$ for simplified notations when the context is clear). To ensure the similarity (dissimilarity) of two users to be preserved after the partition and noise perturbation, the expected $\mathcal{L}_1$ distance between noisy degree vectors $\tilde{d}$ and the expected $\mathcal{L}_1$ distance between neighbor lists d should be close. Therefore, we define the objective function as $E[\frac{|\tilde{d}-d|}{d}]$, i.e., the expectation of the relative error between the $\tilde{d}$ and d.

In the remaining part of this subsection, we present an analysis that connects the number of groups k_1 with the objective function, followed by a deduction of an approximated optimal k_1 value that minimizes the objective function.

Objective Function Analysis. Before showing our analysis on the objective function, we elaborate a property of degree vectors: it is easy to see that if two different neighbors of u and v were partitioned into the same user group, their contributions to the $\mathcal{L}_1$ distance of degree vectors would be canceled by each other. Hence, the $\mathcal{L}_1$ distance between degree vectors is always smaller than the $\mathcal{L}_1$ distance of the original neighbor lists, i.e., $\hat{d} \leq d$, thus $|\hat{d} - d| = d - \hat{d}$. Based on this property, we have:

$$E\left[\frac{\left|\tilde{d}-d\right|}{d}\right] = E\left[\frac{\left|\tilde{d}-\hat{d}+\hat{d}-d\right|}{d}\right] \tag{2}$$

$$\leq \frac{1}{d}\left(E\left[\left|\tilde{d}-\hat{d}\right|\right] + E\left[\left|\hat{d}-d\right|\right]\right) \tag{3}$$

$$= \frac{1}{d}\left(E\left[\left|\tilde{d}-\hat{d}\right|\right] + d - E\left[\hat{d}\right]\right) \tag{4}$$

Deduction from Eq. 2 to 3 is based on the triangle inequality.

By Eq. 4, the objective function is bounded from above by Eq.4. In the following, we will first analyze the relationship between k_1 and $E\left[\left|\tilde{d}-\hat{d}\right|\right]$, i.e., which represents the error introduced by Laplace noise perturbation.

Laplace Noise Perturbation Analysis. By the triangle inequality, we have

$$E\left[\left|\tilde{d}-\hat{d}\right|\right] = E\left[\left|\sum_{i=1}^{k_1}\left|\tilde{\delta}_i^u - \tilde{\delta}_i^v\right| - \sum_{i=1}^{k_1}\left|\hat{\delta}_i^u - \hat{\delta}_i^v\right|\right|\right] \tag{5}$$

$$\leq E\left[\sum_{i=1}^{k_1}\left|\tilde{\delta}_i^u - \tilde{\delta}_i^v - \hat{\delta}_i^u + \hat{\delta}_i^v\right|\right] \tag{6}$$

$$\leq E\left[\left|\sum_{i=1}^{k_1}\left(\tilde{\delta}_i^u - \hat{\delta}_i^u\right)\right|\right] + E\left[\left|\sum_{i=1}^{k_1}\left(\tilde{\delta}_i^v - \hat{\delta}_i^v\right)\right|\right] \tag{7}$$

$$= \sum_{i=1}^{k_1}\left(E\left[\left|\tilde{\delta}_i^u - \hat{\delta}_i^u\right|\right] + E\left[\left|\tilde{\delta}_i^v - \hat{\delta}_i^v\right|\right]\right) \tag{8}$$

Observe that $\tilde{\delta}_i^u$ follows a Laplace distribution $Lap(\hat{\delta}_i^u, \frac{1}{\epsilon_2})$. Accordingly, its mean absolute deviation is:

$$E\left[\left|\tilde{\delta}_i^u - \hat{\delta}_i^u\right|\right] = 1/\epsilon_2 \tag{9}$$

Therefore, the right hand side of Eq. 8 equals $\frac{2k_1}{\epsilon_2}$. To sum up, given the privacy budget ϵ_2 for the corresponding noise injection procedure, the expected error introduced to the degree vector (or the sum of Laplace noise variables) is:

$$E\left[\left|\tilde{d}-\hat{d}\right|\right] \leq k_1 \cdot 2\lambda = \frac{2k_1}{\epsilon_2} \tag{10}$$

Degree Vector Transformation Analysis. Next, we analyze $E\left[\left|d-\hat{d}\right|\right]$, namely, the error introduced by transforming the neighbor list to degree vector. Observe that the common neighbors of u and v do not have any impact on the $\mathcal{L}_1$ distances between their degree vectors. Therefore, we will only focus on the different neighbors of u and v in our analysis of $E\left[\left|d-\hat{d}\right|\right]$. It can be verified that the number of such neighbors should equal d.

Suppose that all users are divided into k_1 groups randomly. Let θ_d denote the number of ways to partition d users. We have

$$\theta_d = \sum_{i=1}^{k_1} S(d,i), \tag{11}$$

where $S(n,k)$ is the Stirling number of the second kind (or the Stirling partition number).

Let $\theta_{\hat{d}=t}$ denote the number of partitions where $\hat{d} = t$. It is easy to see that the $\mathcal{L}_1$ distance between the two degree vectors equals t with a probability $\frac{\theta_{\hat{d}=t}}{\theta_d}$, and the expectation of the $\mathcal{L}_1$ distance between degree vectors is:

$$E[\hat{d}] = \sum_{t=0}^{d} t \cdot \frac{\theta_{\hat{d}=t}}{\theta_d} \tag{12}$$

We now focus on the derivation of $\theta_{\hat{d}=t}$. Let's first consider two different neighbors that are connected to u and v. If these two neighbors are allocated into a same user group, their contribution to the degree vectors' L-1 distance would be 0, i.e., they "cancel out" each other. Given d and t as the L-1 distance between neighbor lists and degree vectors, the number of "cancelling" neighbor pairs is $\frac{d-t}{2}$, and the number of ways to select $\frac{d-t}{2}$ neighbors from each user for the further pairing is $\binom{d/2}{t/2}^2$. Furthermore, by enumerating all possible combinations, we can derive that the number of ways to pair these cancelling neighbors from two users is $\cdot\frac{d-t}{2}!$, and there are $k_1^{\frac{d-t}{2}}$ ways to allocate them among k_1 groups.

Meanwhile, if two different neighbors were allocated to different groups, their contribution to the $\mathcal{L}_1$ distance would be 2, and there are $t/2$ such node pairs. Similar to the derivation procedures above, given the $\mathcal{L}_1$ distance d between neighbor lists as and the $\mathcal{L}_1$ distance t between degree vectors, there are $\frac{t}{2}!$ ways to select the corresponding "non-cancelling" pairs. In addition, for any distance t that is generated by k_1-partitions, there are $\binom{k_1}{2}^{t/2}$ ways to allocate these pairs among k_1 groups.

Combining the above analysis, we have the number of partitions that lead to degree vector distance t as:

$$\theta_{\hat{d}=t} = \left(\frac{d-t}{2}!\right) \cdot \left(\frac{t}{2}!\right) \cdot k_1^{\frac{d-t}{2}} \cdot \binom{d/2}{t/2}^2 \cdot \binom{k_1}{2}^{t/2} \tag{13}$$

Substitute Eq. 11 and 13 into Eq. 12, we can derive that the expected $\mathcal{L}_1$ distance between their degree vectors is:

$$E[d - \hat{d}] = d - \sum_{t=0}^{d} t \frac{(\frac{d-t}{2}!)(\frac{t}{2}!)k_1^{\frac{d-t}{2}} \binom{d/2}{t/2}^2 \binom{k_1}{2}^{t/2}}{\sum_{i=1}^{k_1} S(d,i)} \tag{14}$$

Optimal Group Number Deduction: Finally, we put the above results together to derive an approximation of the optimal value of the group number k_1.

First, we substitute Eq. 10 and 14 into Eq. 4 and obtain:

$$= \frac{1}{d}\left(\frac{\sqrt{2}k_1}{\epsilon_2} + d - \frac{\sum_{t=0}^{d} t(\frac{d-t}{2}!)(\frac{t}{2}!)k_1^{\frac{d-t}{2}} \binom{d/2}{t/2}^2 \binom{k_1}{2}^{t/2}}{\sum_{i=1}^{k_1} S(d,i)}\right) \tag{15}$$

The last item in the Eq. 15 has no closed form. Fortunately, by limiting $d_{x,y}$ and k_1 to a range (e.g., between 0 and 50, which is suitable for most social networks), we get a good approximated closed form as follows:

$$-2(k_1^2+1)^2 - 2k_1(d+1) + (d-1)(k_1^2+1) + 2 - d \tag{16}$$

By substituting Eq. 16 into Eq. 15, we can get a closed form target function:

$$Eq.(2) \approx -2k_1^4 + (d-5)k_1^2 + (\frac{2}{\epsilon_2} - 2 - \frac{2}{d})k_1 + d - 2 \tag{17}$$

Eq. 17 is minimized when

$$k_1(d) = (d + \frac{d^2 - 2(1+\sqrt{5})d + 1}{\epsilon_2}) \tag{18}$$

Moreover, for nodes with the same degree η, we estimate their underlying $\mathcal{L}_1$ distance d heuristically using as:

$$d = \frac{1}{2}\eta. \tag{19}$$

Given the degree histogram $\mathbf{H} = \{p_\eta\}$ of graph G, where p_η is the percentage of nodes with degree η, we can derive a general approximation of the k_1 value for G as:

$$k_1 \approx \lceil \sum_{\eta=1}^{\eta_{max}} p_\eta \cdot k_1(\frac{1}{2}\eta) \rceil, \tag{20}$$

where η_{max} is the maximum node degree in $\mathbf{H}$. Note that our analysis can be adapted to the cases when the above two parameters are larger than 50, though the formula will include many additional terms. We omitted the discussions of these cases for the readability of the paper.

After deriving an appropriate k_1, the data curator could simply partition all users randomly into k_1 equal-size groups. However, we observe that if two users have similar noisy degree vectors reported in Phase I, they are more likely to be structurally similar as well. Thus, if we group nodes based on their noisy degree vectors instead of randomly, it could help further identify structurally close nodes in Phase II. Specifically, in our design the data curator runs the standard k-mean algorithm to cluster users into k_1 groups, based on the noisy degree vectors of all users. The resulting clustering forms the partition ξ_1 for the next phase.

4.3 Design of Phase II

Given the user grouping ξ_1 from Phase I, each user reports again a new noisy degree vector to the data curator, using privacy budget ϵ_2. The task of the data curator in Phase II is to refine the clustering of users. Intuitively, since the partition ξ_1 is of an optimal size and is more likely to group users with similar structures together (through k-mean clustering), the noisy degree vectors reported in this phase would help further reveal the structure of a decentralized social network. This is done by performing another round of k-mean clustering, based on the newly reported noisy degree vectors from all users at the beginning of Phase II. Note that the number of target clusters is still k_1, i.e., the near-optimal value derived in the first phase. We call the resulting new partition ξ_2.

4.4 Design of Phase III

To generate synthetic graphs based on the user clusters in ξ_2, the data curator needs a way to estimate the intra-cluster edges inside each cluster and the inter-cluster edges between different clusters. This could be done by asking users to report their noisy degree vectors given ξ_2. However, as mentioned before, this would cause the privacy budget to further split among each phase. Instead, our design uses the noisy degree vector of a user for partition ξ_1 from Phase II to estimate a degree vector for ξ_2. Specifically, let $\tilde{\delta}^u = (\tilde{\delta_1^u}, \ldots, \tilde{\delta_{k_1}^u})$ be a user u's the noisy degree vector for $\xi_1 = \{U_1^1, \ldots, U_{k_1}^1\}$. Given $\xi_2 = \{U_1^2, \ldots, U_{k_1}^2\}$, we estimate u's degree to a group U_i^2 in ξ_2 as:

$$\sum_{j=1}^{k_1} \frac{|U_j^1 \cap U_i^2|}{|U_j^1|} \cdot \tilde{\delta}_j$$

Essentially, for each group U_j^1, we check what portion of it also appears in U_i^2, and we estimate u's noisy degree to U_j^1 proportionally to that to U_i^2.

Based on the estimated degree vector of each user, we generate edges between nodes in the synthetic graph. Denote δ_i^u as u's estimated degree to U_i^2. Similar to the well-known Chung-Lu model [1], we compute the probability to connect two nodes u and v that belong to group U_i^2 and U_j^2 respectively as follows:

$$\frac{\delta_j^u \cdot \sum_{v \in G} \frac{\delta_j^v}{|U_j^2|}}{\sum_{u \in U_i^2} \delta_j^u + \sum_{v \in G} \delta_j^v},$$

where the denominator is the total number of edges between groups i and j by aggregating all the elements in the nodes' corresponding degree vectors in two groups.

4.5 Privacy Analysis

Here we show that LDPGen satisfies ϵ-edge local differential privacy. Let $(\delta_1, \ldots, \delta_k)$ be u's degree vector for a given partition ξ. Recall that the noisy degree vector reported to the data curator is $(\tilde{\delta}_1, \ldots, \tilde{\delta}_k)$, where $\tilde{\delta}_i = \delta_i + N$ for $i = 1, \ldots, k$, and N is a random variable following the Laplacian distribution $Lap(0, \frac{1}{\epsilon})$. Denote this mechanism as $\mathcal{M}$.

Theorem 4.1. *The above noisy degree vector mechanism $\mathcal{M}$ satisfies ϵ-edge local differential privacy.*

Proof. Given two neighbor lists γ_u and γ_v, let $\delta^u = (\delta_1^u, \ldots, \delta_k^u)$ and $\delta_v = (\delta_1^v, \ldots, \delta_k^v)$ be their corresponding degree vectors for a partition ξ. If γ_u and γ_v differ at exactly one bit, since ξ is a partition, it is easy to see that δ_u and δ_v differ at exact one element. Without loss of generality, assume $\delta_1^u \neq \delta_1^v$. Further, we have $|\delta_1^u - \delta_1^v| = 1$.

Given an arbitrary vector $s = (s_1, \ldots, s_k) \in range(\mathcal{M})$, we have:

$$\frac{Pr[\mathcal{M}(\gamma_u) = s]}{Pr[\mathcal{M}(\gamma_v) = s]} = \frac{Pr[\tilde{\delta}_1^u = s_1] \ldots Pr[\tilde{\delta}_k^u = s_k]}{Pr[\tilde{\delta}_1^v = s_1] \ldots Pr[\tilde{\delta}_k^v = s_k]} = \frac{Pr[\tilde{\delta}_1^u = s_1]}{Pr[\tilde{\delta}_1^v = s_1]}$$

Since $|\delta_1^u - \delta_1^v| = 1$, according to the PDF of $Lap(0, \frac{1}{\epsilon})$, we have $\frac{Pr[\tilde{\delta}_1^u = s_1]}{Pr[\tilde{\delta}_1^v = s_1]} \leq e^{\epsilon}$. □

In LDPGen, a user adds noise to her degree vector following $Lap(0, \frac{1}{\epsilon_1})$ and $Lap(0, \frac{1}{\epsilon_2})$ in Phase I and Phase II respectively. Due to the composability property of edge local differential privacy, since $\epsilon_1 + \epsilon_2 = \epsilon$, the overall process of LDPGen satisfies ϵ-edge local differential privacy.

4.6 Complexity Analysis

Regarding the communication complexity, both straw man approaches need only a single round of communication between the users and the data curator. The first straw man approach RNL has the highest costs on the user end, i.e., $O(n)$, since each user needs to transmit a potentially large neighbor list to the curator. The second straw man approach DGG has much lower costs, i.e., $O(1)$, since each user only need to report his/her degree to the data curator. The proposed approach LDPGen involves two rounds of communications between users and the data curator: In the first phase, each user needs to send a degree vector with size k_0 to the data curator; In the second phase, the data curator first broadcasts the user partition to each user, which is $O(n)$, and then each user sends a degree vector with size k_1 to the data curator. Obviously, LDPGen inevitably incurs higher costs than the straw man methods that only require single round communication, and the high accuracy of LDPGen compensates for such cost.

Regarding the computation complexity, both straw man approaches and the proposed LDPGen approach have low computational overhead on user end, which is also an important design objective of such survey applications. Meanwhile, on the data curator side, since the probability of each edge between nodes needs to be considered by the data curator once, both straw man methods have a computation overhead of $O(n^2)$ on generating the synthetic graph, while the LDPGen approach has a cost of $O(n^2 + n(k_0 + k_1))$, which is just slightly higher than straw man methods.

4.7 Discussions

Recall that Phase II of LDPGen refines the partition from Phase I by re-clustering users based on their newly reported noisy degree vectors. Technically, we could continue this process and have more rounds of grouping refinement instead of stopping at the second phase. However, the more rounds we have, the less privacy budget each round could use, which means much more noise needs to be added to degree vectors. This could in fact reduce rather than improve the quality of groupings. In Figure 10 from Section 5, we experimentally show the impact of multiple rounds of partition refinements on the utility of the generated synthetic graph.

Another issue is the allocation of privacy budget in the first two phases. In our implementation, we choose to evenly split ϵ, i.e., $\epsilon_1 = \epsilon_2$, based on the intuition that in both phases users report the same type of information, and tend to have equal influence on the final user clustering.

In Phase I, before collecting any information from users, the data curator partitions all the users into k_0 equal-size groups randomly. Recall that a user's noisy degree vector based on this random partition serves two purposes. First, the data curator uses it to estimate the user's degree (recall equation 1). Second, it is also used as the feature vector later to cluster all users. To get the best degree estimation, k_0 should be set to 1. However, that means the later clustering is essential only based on degrees, which is undesirable as discussed before (section 3.2). In our implementation, we set $k_0 = 2$ to strike an intuitive balance.

In our discussion so far, we implicitly assume that the data curator is aware of all the users in the decentralized social network. Whether this assumption holds depends on the specifics of the network. For example, if the social network is about the communications between different IPs, then the domain of all IPs is publicly known. The data curator could also first inquiry their social network IDs when users opt in for a study, when such IDs are not sensitive (e.g., their email addresses for the contact-list network). When users' social network IDs could not be revealed directly, they could instead use the hashes of their IDs to communicate with the data curator. By doing so, an unknown user ID space is mapped to a well-defined hash space. Our scheme could then be applied over this hash space. In addition, the curator could broadcast the user group partitions using compression techniques such as Bloom filter to reduce communication cost.

5 EXPERIMENTAL EVALUATION

We have implemented LDPGen and run a comprehensive set of experiments to study its effectiveness. In particular, our experiments compare LDPGen with the two straw-man approaches RNL and DGG described in Section 2. Meanwhile, the experiments also investigate the effect of number of groups k_1 in Phase 1 (described in Section 4.2), and whether LDPGen obtains an appropriate value for this parameter. For each experiment, we report the average results over 10 executions. The experiments involve four real social graphs that have been used as benchmark datasets for community detection and recommendation tasks. They are:

- *Facebook* [30] is an undirected social graph consisting of 4, 039 nodes (i.e., users) and 88, 234 edges (connections on Facebook), collected from survey participants using a Facebook app.
- *Enron* [29] is an undirected email graph consisting of 36, 692 nodes (i.e., email accounts in Enron) connected by 183, 831 edges (emails).

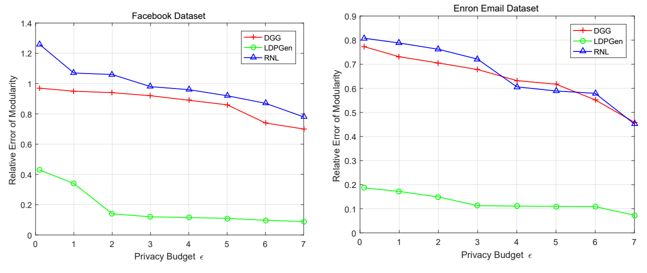
- *Last.fm* [3] contains both a *social graph* and a *preference graph.* The social graph consists of 1, 892 user nodes and 12, 717 undirected edges that represent friend relationships. The preference graph contains the same set of user nodes, as well as 17, 632 item nodes, each of which corresponds to a song. Each edge in the preference graph connects a user and a song, with a weight that corresponds to the number of times that the user has listened to the song. In total, there are 92, 198 such directed user-to-song edges.
- *Flixster* [21] also contains a social graph and a preference graph, similar to *Last.fm.* After some pre-processing (explained below), the social graph contains 137, 372 user nodes connected by 1, 269, 076 undirected edges, and the preference graph contains the same set of users, 48, 756 item nodes, and over 7 million directed user-to-item edges. Here, each item node represents a movie, and each user-to-item edge corresponds to a movie rating by the corresponding user, associated with a weight, i.e, the rating in the range of [0, 5].

The experiments mostly use centralized social graphs since we need ground truth to validate the effectiveness of LDPGen. Among these datasets, Enron contains email communications which are usually modeled as decentralized data, as typically no server has all the email communications between all the users.

The utility of a synthetic social network depends on for what it is used in different applications. In our experiments, we validate the effectiveness of LDPGen using three use cases, as follows.

- *Social graph statistics.* As the simplest use case, we use datasets *Facebook* and *Enron* to investigate how well the synthetic social graphs generated under local differential privacy preserve important statistics (detailed in Section 5.1) of the original graph structure.
- *Community discovery.* In the second use case, we perform community discovery on the original and generated synthetic graphs, and compare the results, using datasets *Facebook* and *Enron.* In particular, our implementation uses the Louvain method [5] for community discovery, which outputs a list of user clusters. We then compare the user clusters obtained from the real and synthetic graphs on two common metrics ARI and AMI, explained in Section 5.1.
- *Social recommendation.* In the third use case, we use *Last.fm* and *Flixster* datasets to study how the generated synthetic graphs can be effectively used in a social recommendation application. Specifically, we assume that the social graph is private, and the preference graph is publicly available (e.g., when a user rates a movie, the rating is publicly visible). Then, for each user, we recommend a list of top-k items based on item preferences of her friends. The evaluation compares the recommended item lists computed from the real and synthetic graphs. The detailed reasoning behind the social recommendation algorithm can be found in [31].

In the following Section 5.1 describes the evaluation metrics; Section 5.2 presents the main evaluation results; and Section 5.3 discusses parameter selection in LDPGen, particularly whether LDPGen is able to select an appropriate value for the number of groups k_1 in Phase I.



(a) Facebook **(b) Enron**

Figure 2: Effect of ϵ on modularity

5.1 Evaluation Metrics

As explained above, the experiments focus on three aspects of utility: preservation of graph statistics, accuracy in community discovery, and effectiveness in a recommendation system. First, in terms of graph statistics, we evaluate LDPGen and its competitors in terms of graph *modularity* [19], *clustering coefficient* [48] and *assortativity coefficient* [18]. Each of the three reflects the structure of the graph, and has numerous applications in practice [7]. We omit their detailed definitions for brevity.

Regarding community detection, the utility metric focuses on the similarity of the communities obtained from the generated synthetic graph and those from the original graph. For this purpose, we follow the standard metrics Adjusted Random Index (ARI) [40] and Adjusted Mutual Information (AMI) [44], as follows.

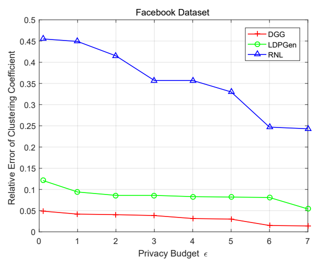
Definition 5.1 (ARI and AMI). Given a set of N elements $G = \{n_1, ..., n_n\}$ and two partitioning schemes of G, $X = \{x_1, ..., x_r\}$ (which partitions G into r subsets) and $Y = \{y_1, ..., y_s\}$ (which partitions of G into s subsets), the overlap between X and Y can be summarized in a contingency table where each entry n_{ij} denotes the number of objects in common between x_i and y_j: $n_{ij} = |X_i \cap Y_j|, \forall 1 \le i \le r, 1 \le j \le s$. Let a_i (resp. b_j) be the sum of the i-th row (resp. j-th column) in the contingency table, i.e., $a_i = \sum_j n_{ij}$ and $b_j = \sum_i n_{ij}$. Define ARI and AMI as follows:

$$ARI(X, Y) = \frac{\sum_{ij} \binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \quad (21)$$

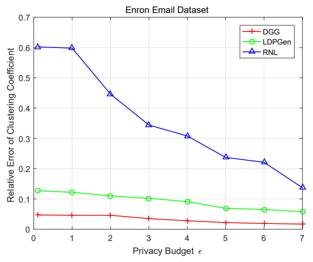
$$AMI(X, Y) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{n_{ij}=(a_i+b_j-N)^+}^{\min(a_i, b_j)} \frac{n_{ij}}{N} \log\left(\frac{N \cdot n_{ij}}{a_i b_j}\right) \times \frac{a_i! b_j! (N - a_i)! (N - b_j)!}{N! n_{ij}! (a_i - n_{ij})! (b_j - n_{ij})! (N - a_i - b_j + n_{ij})!} \quad (22)$$

where $(a_i + b_j - N)^+$ denotes $\max(1, a_i + b_j - N)$.

Intuitively, ARI quantifies the frequency of agreements between the two obtained clusterings over all element pairs, and AMI quantifies the information shared by the two clusterings. Both ARI and AMI discount the measurement between two completely random clusterings [6, 35]. Larger values of ARI and AMI indicate that the underlying clusterings are more similar; in our experiments that compare the results obtained on real and synthetic graphs, higher ARI/AMI values signify higher accuracy.



(a) Facebook



(b) Enron

Figure 3: Effect of ϵ on clustering coefficient

Finally, in terms of effectiveness in social recommendation, we measure utility as the similar of the recommended items computed from the original and those obtained from the synthetic graph. Specifically, the score μ_u^i for recommending an item i to a user u is defined as:

$$\mu_u^i = \sum_v |\Gamma(v) \cap \Gamma(u)| \cdot w(v, i) \tag{23}$$

where v is another arbitrary user, $\Gamma(u)$ denotes the set of nodes directly connected to u in the social graph, and $w(v, i)$ is the weight in the preference graph connecting user u and item i.

The recommendation system then recommends to each user u a list of k items with the highest scores with respect to u. To compare two different top-k item lists, we use Normalized Discounted Cumulative Gain (NDCG) [22] as the accuracy metric, which is a commonly used measure for the effectiveness of recommendation systems [34]. In particular, the NDCG score between to ranked lists is defined as follows.

Definition 5.2 (NDCG). Given two ranked lists $rank_{act}$ and $rank_{est}$. Given an item v_i in the ranked lists, we define its relevance score rel_i as:

$$rel_{v_i} = \log_2 |d - |rank_{act}(v_i) - rank_{est}(v_i)||$$

where $rank_{act}(v_i)$ (resp. $rank_{est}(v_i)$) returns the rank of item v_i in the ranked list $rank_{act}$ (resp. $rank_{est}$. Intuitively, the closer v_i's estimated rank is to its actual rank, the higher the relevance score. Given the actual top k items $v_1, \ldots, v_k$, the discounted cumulative gain (DCG) of an estimated rank list $rank_{est}$ is computed as:

$$DCG_k = rel_{v_1} + \sum_{i=2}^{k} \frac{rel_{v_i}}{\log_2 i}$$

The discount factor $\log_2(i)$ is to give more weight to the gain of higher ranked items. Essentially, we care more about the correct ranking of important items (those with high ranks). Finally, we normalize DCG of an estimated ranking list by comparing it with the ideal DCG (IDCG), which is DCG when the estimated ranking list is exactly the same as the actual one (i.e., no estimation error):

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

The overall utility of the synthetic graph with respect to the social recommendation use case is then calculated as the average NDCG score over the set of all users in the dataset.

5.2 Utility of LDPGen

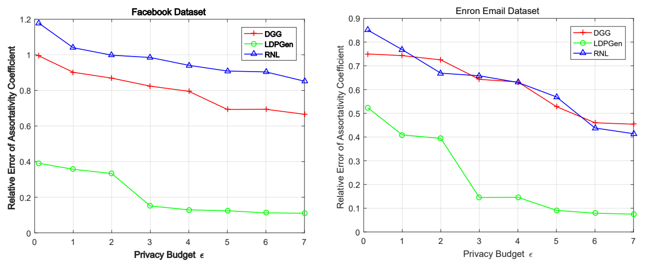
In this subsection, we study the utility of LDPGen in comparison with the two straw-man approaches RNL and DGG. In all experiments, the optimal number of groups k_1 is computed by the data curator as described in 4.2. The effects of k_1, and whether LDPGen's choice is appropriate, are further investigated in Section 5.3.

Results on graph structure statistics. Figure 2a plots the relative errors of the proposed solution LDPGen and two straw-man approaches RNL and DGG, as functions of the privacy budget ϵ, using the *Facebook* dataset. LDPGen demonstrates clear advantages over RNL and DGG on all settings. In particular, unless we use a very high value of ϵ, i.e., $\epsilon > 5$, the relative errors of RNL and DGG are close to 100%, meaning that their modularity results are rather useless under these settings. LDPGen, on the other hand, achieves low relative error (< 20%) when $\epsilon \geq 2$. Note that such values of ϵ have been commonly used in the local differential privacy literature, e.g., [16, 39]. This indicates that LDPGen obtains practically meaningful results in terms of modularity under local differential privacy. Meanwhile, LDP also achieves dramatic performance improvements over RNL and DGG on extreme values of ϵ (i.e., when ϵ approaches 0 and reaches 7, respectively). This demonstrates that the LDPGen has inherent utility advantages over its competitors on a broad range of ϵ values.

Figure 2b repeats the above experiment on the *Enron* dataset. Again, LDPGen outperforms RNL and DGG by large margins on all settings, and the difference is between practical accuracy (< 20% relative error for LDPGen on all settings) and useless results (> 60% relative error for RNL and DGG, when $\epsilon < 5$). This confirms that the performance advantage of LDPGen over RNL and DGG is inherent, rather than data-dependent. Comparing the results on *Facebook* and *Enron*, we observe that (i) all three methods obtain higher utility on *Enron* than *Facebook*. one reason is that *Enron* is a much bigger dataset than *Facebook*; consequently, the noise injected to satisfy local differential privacy is more pronounced on the latter dataset. (ii) Although the RNL performs slightly better than DGG on *Facebook*, its performance on *Enron* is comparable to that of RNL, which suggest that the relative performance of the two straw-man approaches depends on the dataset.

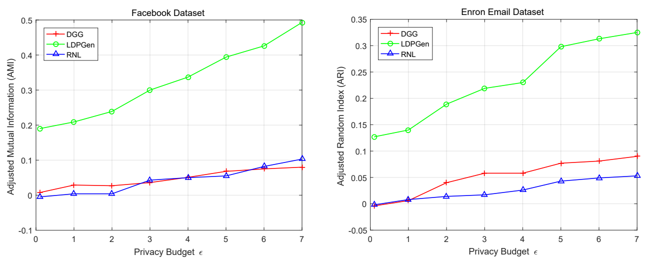
Figures 3a and 3b evaluate the utility of LDPGen, RNL and DGG in terms of relative error in the clustering coefficient of the generated synthetic graph, using the *Facebook* and *Enron* datasets, respectively. DGG obtains the best accuracy in all settings. This is expected, since DGG is based on the BTER algorithm (sketched in Section 6), which is specifically optimized for returning accurate clustering coefficients [33]. Notably, the accuracy of LDPGen is close to that of DGG in all settings, whereas RNL leads to far lower utility than the other two methods.

Figures 4a and 4b exhibit the utility results in terms of relative error of the graph assortativity coefficient in the generated synthetic graph. The results lead to similar observations as those for modularity (Figures 2a and 2b), i.e., LDPGen achieves practical accuracy whereas RNL and DGG lead to useless results. This indicates that the two straw-man solutions have very low utility, except for the statistics that they are specifically optimized for (e.g., DGG for clustering coefficient). The performance of LDPGen, on the other



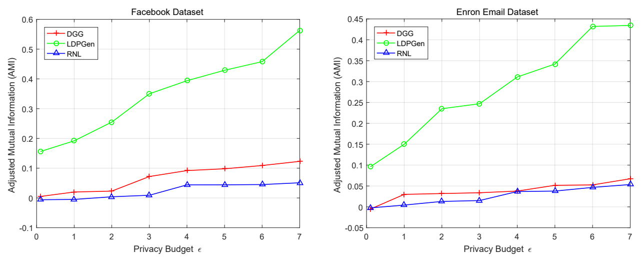
(a) Facebook (b) Enron

Figure 4: Effect of ϵ: on assotativity coefficient



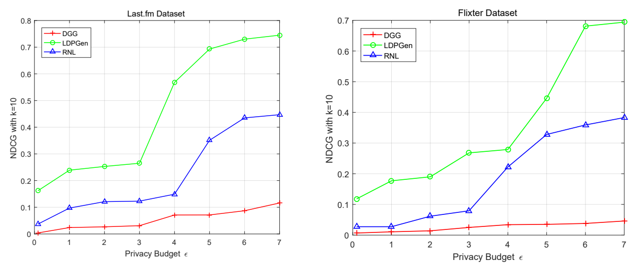
(a) Facebook (b) Enron

Figure 5: Effect of ϵ on Adjusted Random Index



(a) Facebook (b) Enron

Figure 6: Effect of ϵ on Adjusted Mutual Information



(a) Last.fm (b) Flixter

Figure 7: Effect of ϵ on NDCG

hand, is consistently high in all metrics, on both datasets, and under all values of the privacy budget ϵ.

Community preservation results. In our second set of experiments, we evaluate how well the synthetic graphs generated by LDPGen, RNL and DGG preserve the community information in the original graph, using two metrics ARI and AMI defined in Section 5.1. Figures 5a and 5b show the evaluation results using the ARI metric, on datasets *Facebook* and *Enron* respectively. *Note that higher values of ARI and AMI correspond to higher accuracy.* Clearly, on both datasets, the proposed approach LDPGen significantly outperforms the straw-man methods RNL and DGG. Further, the ARI values of LDPGen grows rapidly with the privacy budget ϵ, whereas the ARI lines for RNL and DGG stay almost flat. These observations indicate that LDPGen is more effective in collecting private information under local differential privacy. Comparing the two straw-man approaches, DGG slightly outperforms RNL on the larger *Enron* dataset; yet, the ARI values of DGG remain very low in comparison with LDPGen, in all settings.

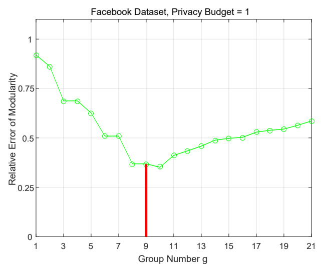
Figures 6a and 6b exhibit community information preservation results using the AMI metric, also defined in Section 5.1. These results lead to similar conclusions as the ARI results described above, i.e., LDPGen beats its competitors on every setting, and the performance gap expands with increasing values of the privacy budget ϵ. Comparing ARI and AMI results, LDPGen performs consistently better in terms of AMI than ARI, which suggests that LDPGen might be more suitable for applications where high AMI values are more important.

Utility results in recommendation systems. Next we evaluate the effectiveness of LDPGen, RNL and DGG in the social recommendation use case, using the NDCG metric explained in Section 5.1, which measures the similarly of recommended lists of items using the real and synthetic data, respectively. *Larger NDCG values correspond to higher accuracy.*

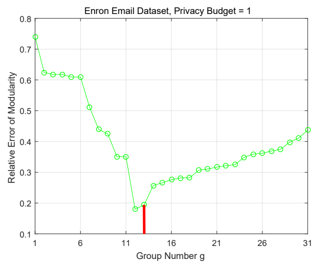
Figures 7a and 7b demonstrate the NDCG scores of all three methods using the *Last.fm* and *Flixter* datasets, respectively. Once again, LDPGen clearly outperforms RNL and DGG on all settings and both datasets. Comparing RNL and DGG, the latter completely fails, yielding NDCG scores no more than 0.2, meaning that the items it recommends has few in common with the recommendations computed using the exact social graph. RNL performs much better than DGG; the reason is that in the recommendation application, only immediate neighbors are important, and RNL directly collects this information. Nevertheless, LDPGen still obtains superior NDCG scores compared to RNL.

An interesting observation is that the NDCG result of LDPGen does not increase linearly with the privacy budget ϵ; instead, the NDCG score remains low for small values of ϵ, increases rapidly when $3 \le \epsilon \le 6$, and plateaus afterwards. The reason is that for each user u, the recommended list of items consists of common items rated high among its neighbors. When ϵ is low, the social graph is very noisy; yet, there are items that are rated high by most users, not just neighbors of u. Hence, the NDCG score mainly reflects such items, which are not sensitive to how noisy u's neighbor list is. Once ϵ reaches a certain level, items that are rated high among u's neighbors but not globally gradually appear in the recommended item list for u, boosting the NDCG score. Finally, when the majority of such items are included in the recommended item list, the NDCG score of LDPGen becomes stable; in other words, at this stage, the recommended items computed using the synthetic and real graphs differ mainly in details, i.e., items that are not consistently rated high by the user's neighbors.

Summarizing the utility evaluations, LDPGen is the clear winner, which obtains high accuracy on all metrics and over all datasets.

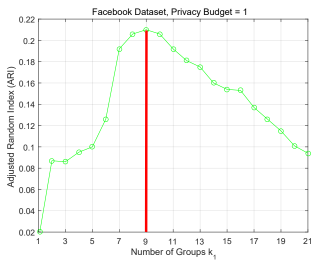


(a) Facebook

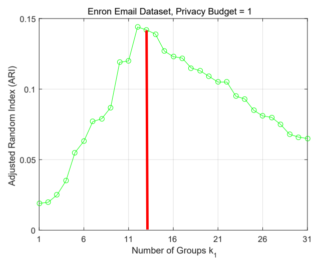


(b) Enron

Figure 8: Effect of number of groups k_1 on modularity



(a) Facebook



(b) Enron

Figure 9: Effect of number of groups k_1 on ARI

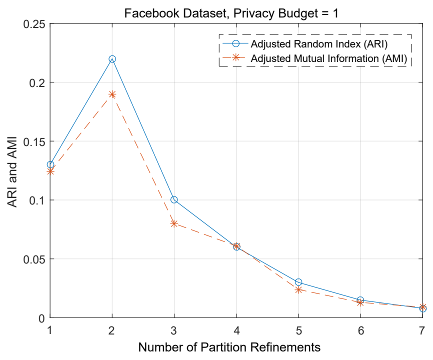


Figure 10: ARI and AMI vs. Number of rounds for grouping refinements (Facebook)

RNL and DGG are only competitive for metrics that they are specifically optimized for (e.g., DGG for clustering coefficient) or when the application uses mainly the information that they directly collects (e.g., RNL for the recommendation application). Hence, LDPGen is the method of choice for synthetic decentralized graph generation in practical applications.

5.3 Parameter Study for LDPGen

Having established the superiority of LDPGen over its competitors, we now investigate the impact of an appropriate value for the parameter k_1 in LDPGen, which is the number of groups in Phase I, as described in Section 4.2. Intuitively, a smaller k_1 leads to a coarser granularity in the reported neighbor lists of each user, and at the same time a smaller amount of random perturbations is needed for satisfying local differential privacy. The reverse is true for a larger value of k_1. Hence, choosing an appropriate k_1 is crucial for the overall performance of LDPGen. As described in Section 4.2, LDPGen chooses a value for k_1 based on the collected degree vectors. Hence, the experiments also evaluate the quality of LDPGen's choice for k_1.

Figure 8 presents the modularity results for LDPGen on *Facebook* and *Enron* datasets with varying values of k_1, after fixing the privacy budget $\epsilon = 1$. The same figure also shows the value chosen by LDPGen, shown in the vertical line. For both datasets, the utility of LDPGen first increases with k_1, and then decreases with k_1 after the latter reaches its optimal value, which is very close to what LDPGen chooses. This confirms the effectiveness of the method in Section 4.2 for choosing k_1. Similarly, Figures 9 show the effect of k_1 on the utility of community discovery for LDPGen in terms of the ARI metric, as well as the values of k_1 chosen by LDPGen. Clearly, these results are consistent with the ones for modularity. We omit further experimental results on k_1, since they all lead to the same conclusions.

Finally, Figure 10 shows the effect of number of rounds for refining the user grouping in Phase II of LDPGen, using the ARI/AMI metrics and the *Facebook* dataset. The privacy budget ϵ is fixed to 1 in these experiments. The results show that two rounds of refinement work best. This because more rounds of grouping refinement lead to diminishing benefits in terms of improving user grouping, which do not compensate for the increased privacy budget consumption. Results on other metrics and datasets lead to the same conclusion. Hence, LDPGen always performs rounds of refinement for user grouping, one at the end of Phase I and one in Phase II, as described in Section 4.

6 RELATED WORK

There is a large body of work on formal models for graph generation. The earliest model is the Erdos-Renyi (ER) model [14], which assumes that every pair of nodes in a graph is connected by an edge with an independent and identical probability. Chung and Lu proposed the CL model [1], which is similar in spirit to the ER model but allows each edge to exist with a different probability, so as to ensure that the node degrees follow a given distribution. Based on the ER and CL models, Seshadri et al. propose the Block Two-Level Erdos-Renyi Graph Model (BTER) [43], which uses the standard ER model to form relatively dense subgraphs (which correspond to communities), and then utilizes the CL model to add additional edges so as to ensure that the resulting graph follows a given edge distribution. Subsequently, there are a few variations of the BTER models with increased complexities for more accurate graph modelling. In addition, there exist several other graph models, such as the Kronecker graph model [28], the preferential attachment model [38], the exponential random graph model [41].

The aforementioned graph models do not take into account the protection of individual privacy in the generation of synthetic graphs. To address this issue, recent work [8, 23, 24, 32, 36, 42, 45, 46, 50] has developed graph generation methods that ensure differential privacy in the *centralized* setting, based on same basic idea as follows. Given an input graph G, they first derive some noisy statistics of G (e.g., degree distributions), in a manner that satisfies differential privacy. After that, they feed the noisy statistics into a graph model, and return the synthetic graph G^* thus obtained. Since G^* depends only on the noisy statistics (which are differentially private) instead of the input graph, G^* also achieves differential privacy. These methods, however, require that the data publisher know the whole input graph *G*. As a consequence, they are inapplicable in our setting, where the data publisher does not have direct access to *G* and needs information from each user (i.e., each vertex) while ensuring local differential privacy. Note that, our approach could also use existing models after user groups are refined in Phase II, as long as the model does not require more than one-hop information of users.

Local differential privacy (LDP) is a privacy framework adapted from the traditional centralized differential privacy (DP) model to address cases in which data is distributed among different users and hence no collector has access to the complete dataset, or the collector does not want to be liable for collecting the original data from end users. Most of the existing LDP approaches, however, are limited to cases where end users data is a tuple of numerical or categorical values, and the collector computes simple statistics such as as counts and histograms. A long line of research has been introduced for frequency estimation under LDP including [2, 9, 15, 17, 20, 25, 47, 49]. Specifically, the pioneer study on randomized response method by Warner [47] has been followed by a study which formalized the LDP model [25]. Duchi et al. [9] provide a minmax-error bound on convex statistical estimation by utilizing information theoretic converse techniques, while Hsu et al. [20] uses random projection and concentration of measure to estimate heavy hitters. Then, Bassily et al. [2] propose an algorithm for succinct histogram estimation with information-theoretical optimal error. Erlingsson et al. [15] propose an extension of randomized response for collecting the frequencies of categorical values, while Fanti et al. [17] further extends it to handle the case when the domain of the categorical values are unknown. After that, an interesting work has been proposed by Xiong et al. [49], which offers LDP by providing optimization over discrete memoryless channels between the sensor observations and their compressed version.

A more recent work by Qin et al. [39] proposes a two-phase LDP mechanism, dubbed LDPMiner, which tackles more complex data mining tasks. Specifically, LDPMiner is shown to provide accurate heavy hitters estimation over set-valued data. Private estimation of frequent element from stream data has also been studied by [12]. Nguyen et al. [37] proposes an LDP mechanism, dubbed Harmony, as a practical, accurate and efficient system for collecting and analyzing data from users of smart device. Harmony can handle multi-dimensional data containing both numerical and categorical attributes, and supports both basic statistics (e.g., mean and frequency estimates), and complex machine learning tasks (e.g., linear regression, logistic regression and SVM classification).

The common theme of all above LDP research is that it is limited to collecting simple, statistical information such as counts, histograms, and heavy hitters. Additionally, as pointed out earlier, quite some has been done on graph generation under the global differential privacy framework. To the best of our knowledge, we are the first to propose a mechanism for generating synthetic decentralized social graphs with local differential privacy.

7 CONCLUSION

Rich and valuable information exists in decentralized social networks, where graph information is only stored by individual users in their limited local views. In this paper, we propose LDPGen, a multi-phase technique to gradually extract information from users and construct a representative graph of the underlying social network, while guaranteeing edge local differential privacy. The advantages of LDPGen over straightforward adaption of existing LDP and synthetic graph generation techniques are validated through comprehensive experiments over several real datasets.

To the best of our knowledge, this work is the first effort towards privacy-preserving graph analysis over decentralized social networks with local differential privacy. Many challenging issues remain, including incorporating stronger privacy models (e.g., node-local differential privacy), handling graphs with edge weights and node/edge attributes, and developing techniques for specific graph mining tasks, e.g, triangle counting and frequent subgraph structure mining, which we plan to address in our future work.

8 ACKNOWLEDGMENTS

Xiaokui Xiao was supported by grant MOE2015-T2-2-069 from MOE, Singapore and by the DSAIR center at NTU, Singapore. Yin Yang was supported by QNRF grant NPRP10-0708-170408.

REFERENCES

[1] William Aiello, Fan Chung, and Linyuan Lu. 2000. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. Acm, 171–180.

[2] Raef Bassily and Adam D. Smith. 2015. Local, Private, Efficient Protocols for Succinct Histograms. In *STOC*. 127–135.

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

[4] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2012. The johnson-lindenstrauss transform itself preserves differential privacy. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. IEEE, 410–419.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[6] Christian Braune, Christian Borgelt, and Rudolf Kruse. 2013. Behavioral clustering for point processes. In *International Symposium on Intelligent Data Analysis*. Springer, 127–137.

[7] L da F Costa, Francisco A Rodrigues, Gonzalo Travieso, and Paulino Ribeiro Villas Boas. 2007. Characterization of complex networks: A survey of measurements. *Advances in physics* 56, 1 (2007), 167–242.

[8] Wei-Yen Day, Ninghui Li, and Min Lyu. 2016. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 123–138.

[9] John C Duchi, Michael I Jordan, and Martin J Wainwright. 2013. Local privacy and statistical minimax rates. In *FOCS*. 429–438.

[10] Cynthia Dwork. 2006. Differential privacy. In *Automata, languages and programming*. 1–12.

[11] Cynthia Dwork and Jing Lei. 2009. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. ACM, 371–380.

[12] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N Rothblum, and Sergey Yekhanin. 2010. Pan-Private Streaming Algorithms.. In *ICS*. 66–80.

[13] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[14] Paul Erdös and Alfréd Rényi. 1959. On random graphs, I. *Publicationes Mathematicae (Debrecen)* 6 (1959), 290–297.

[15] Úlfar Erlingsson et al. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*. 1054–1067.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 1054–1067.

[17] Giulia Fanti et al. 2016. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. *PoPETS* issue 3, 2016 (2016).

[18] Neil Zhenqiang Gong, Wenchang Xu, Ling Huang, Prateek Mittal, Emil Stefanov, Vyas Sekar, and Dawn Song. 2012. Evolution of social-attribute networks: measurements, modeling, and implications using google+. In *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 131–144.

[19] Roger Guimera, Marta Sales-Pardo, and Luís A Nunes Amaral. 2004. Modularity from fluctuations in random graphs and complex networks. *Physical Review E* 70, 2 (2004), 025101.

[20] Justin Hsu, Sanjeev Khanna, and Aaron Roth. 2012. Distributed private heavy hitters. In *Automata, Languages, and Programming*. 461–472.

[21] Mohsen Jamali and Martin Ester. 2010. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 135–142.

[22] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[23] Zach Jorgensen, Ting Yu, and Graham Cormode. 2016. Publishing Attributed Social Graphs with Formal Privacy Guarantees. In *SIGMOD*. 107–122.

[24] Vishesh Karwa and Aleksandra B Slavković. 2012. Differentially private graphical degree sequences and synthetic graphs. In *Privacy in Statistical Databases*. Springer, 273–285.

[25] Shiva Prasad Kasiviswanathan et al. 2011. What can we learn privately? *SICOMP* 40, 3 (2011), 793–826.

[26] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2013. Analyzing graphs with node differential privacy. In *Theory of Cryptography*. Springer, 457–476.

[27] Dor Konforty, Yuval Adam, Daniel Estrada, and Lucius Gregory Meredith. 2015. Synereo: The Decentralized and Distributed Social Network. (2015).

[28] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research* 11, Feb (2010), 985–1042.

[29] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[30] Jure Leskovec and Julian J Mcauley. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*. 539–547.

[31] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150–1170.

[32] Wentian Lu and Gerome Miklau. 2014. Exponential Random Graph Estimation under Differential Privacy. In *KDD*. 921–930.

[33] R Duncan Luce and Albert D Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (1949), 95–116.

[34] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press Cambridge.

[35] Marina Meilă. 2003. Comparing clusterings by the variation of information. In *Learning theory and kernel machines*. Springer, 173–187.

[36] Darakhshan Mir and Rebecca N Wright. 2012. A differentially private estimator for the stochastic kronecker graph model. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. ACM, 167–176.

[37] Thông T. Nguyên, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. 2016. Collecting and Analyzing Data from Smart Device Users with Local Differential Privacy. *CoRR* abs/1606.05053 (2016). http://arxiv.org/abs/1606.05053

[38] Derek de Solla Price. 1976. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science* 27, 5 (1976), 292–306.

[39] Zhan Qin, Yin Yang, Ting Yu, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2016. 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)*. ACM, New York, NY, USA, 192–203. https://doi.org/10.1145/2976749.2978409

[40] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.

[41] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. 2007. An introduction to exponential random graph (p*) models for social networks. *Social networks* 29, 2 (2007), 173–191.

[42] Alessandra Sala, Xiaohan Zhao, Christo Wilson, Haitao Zheng, and Ben Y Zhao. 2011. Sharing graphs using differentially private graph models. In *IMC*. ACM, 81–98.

[43] Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. 2012. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E* 85, 5 (2012), 056109.

[44] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research* 11, Oct (2010), 2837–2854.

[45] Yue Wang and Xintao Wu. 2013. Preserving Differential Privacy in Degree-Correlation Based Graph Generation. *Trans. Data Privacy* 6, 2 (Aug. 2013).

[46] Yue Wang, Xintao Wu, and Leting Wu. 2013. Differential Privacy Preserving Spectral Graph Analysis. In *PAKDD*. 329–340.

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

[48] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.

[49] Sijie Xiong, Anand D. Sarwate, and Narayan B. Mandayam. 2016. Randomized requantization with local differential privacy. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*. 2189–2193. https://doi.org/10.1109/ICASSP.2016.7472065

[50] Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2015. Private Release of Graph Statistics using Ladder Functions. In *SIGMOD*. 731–745.