Differentially Private Triangle and 4-Cycle Counting in the Shuffle Model

Jacob Imola\textsuperscript{*}
nimola@eng.ucsd.edu
UC San Diego
USA

Takao Murakami\textsuperscript{*}
takao-murakami@aist.go.jp
AIST
Japan

Kamalika Chaudhuri
kamalika@cs.ucsd.edu
UC San Diego
USA

ABSTRACT

Subgraph counting is fundamental for analyzing connection patterns or clustering tendencies in graph data. Recent studies have applied LDP (Local Differential Privacy) to subgraph counting to protect user privacy even against a data collector in social networks. However, existing local algorithms suffer from extremely large estimation errors or assume multi-round interaction between users and the data collector, which requires a lot of user effort and synchronization.

In this paper, we focus on a one-round of interaction and propose accurate subgraph counting algorithms by introducing a recently studied shuffle model. We first propose a basic technique called wedge shuffling to send wedge information, the main component of several subgraphs, with small noise. Then we apply our wedge shuffling to counting triangles and 4-cycles - basic subgraphs for analyzing clustering tendencies - with several additional techniques. We also show upper bounds on the estimation error for each algorithm. We show through comprehensive experiments that our one-round shuffle algorithms significantly outperform the one-round local algorithms in terms of accuracy and achieve small estimation errors with a reasonable privacy budget, e.g., smaller than 1 in edge DP.

CCS CONCEPTS
- Security and privacy → Privacy-preserving protocols: Social network security and privacy.

KEYWORDS
differential privacy, shuffle model, subgraph counting, wedges

ACM Reference Format:
Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Differentially Private Triangle and 4-Cycle Counting in the Shuffle Model. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560659

1 INTRODUCTION

Graph statistics is useful for finding meaningful connection patterns in network data, and subgraph counting is known as a fundamental task in graph analysis. For example, a triangle is a cycle of size three, and a k-star consists of a central node connected to k other nodes. These subgraphs can be used to calculate a clustering coefficient \( (\frac{3\times \text{triangles}}{\#2\text{-stars}}) \). In a social graph, the clustering coefficient measures the tendency of nodes (users) to form a cluster with each other. It also represents the average probability that a friend’s friend is also a friend [53]. Therefore, the clustering coefficient is useful for analyzing the effectiveness of friend suggestions. Another example of the subgraph is a 4-cycle, a cycle of size four. The 4-cycle count is useful for measuring the clustering ability in bipartite graphs (e.g., online dating networks, mentor-student networks [43]) where a triangle never appears [46, 58, 61]. Figure 1 shows examples of triangles, 2-stars, and 4-cycles. Although these subgraphs are important for analyzing the connection patterns or clustering tendencies, their exact numbers can leak sensitive edges (friendships) [32].

DP (Differential Privacy) [21, 22] – the gold standard of privacy notions – has been widely used to strongly protect edges in graph data [19, 20, 30, 32, 33, 38, 40, 55, 62, 67, 68]. In particular, recent studies [32, 33, 55, 67, 68] have applied LDP (Local DP) [39] to graph data. In the graph LDP model, each user obfuscates her neighbor list (friend list) by herself and sends the obfuscated neighbor list to a data collector. Then, the data collector estimates graph statistics, such as subgraph counts. Compared to central DP where a central server has personal data of all users (i.e., the entire graph), LDP does not have a risk that all personal data are leaked from the server by cyberattacks [31] or insider attacks [41]. Moreover, LDP can be applied to decentralized social networks [54, 60] (e.g., diaspora* [4]) or Mastodon [5]) where no server can access the entire graph; e.g., the entire graph is distributed across many servers, or no server has any original edges. It is reported in [32] that k-star counts can be accurately estimated in this model.

However, it is much more challenging to accurately count more complicated subgraphs such as triangles and 4-cycles under LDP. The root cause of this is its local property – a user cannot see edges between others. For example, user \( v_1 \) cannot count triangles or 4-cycles including \( v_1 \), as she cannot see edges between others, e.g., \((v_2, v_3), (v_2, v_4), \) and \((v_3, v_4)\). Therefore, the existing algorithms [32, 33, 67, 68] obfuscate each bit of the neighbor list rather than the subgraph count by the RR (Randomized Response) [64], which randomly flips 0/1. As a result, their algorithms suffer from extremely
large estimation errors because it makes all edges noisy. Some studies [32, 33] significantly improve the accuracy by introducing an additional round of interaction between users and the data collector. However, multi-round interaction may be impractical in many applications, as it requires a lot of user effort and synchronization; indeed, in [32, 33], every user must respond twice, and the data collector must wait for responses from all users in each round.

In this work, we focus on a one-round of interaction between users and the data collector and propose accurate subgraph counting algorithms by introducing a recently studied privacy model: the shuffle model [24, 25]. In the shuffle model, each user sends her (encrypted) obfuscated data to an intermediate server called the shuffler. Then, the shuffler randomly shuffles the obfuscated data of all users and sends the shuffled data to the data collector (who decrypts them). The shuffling amplifies DP guarantees of the obfuscated data under the assumption that the shuffler and the data collector do not collude with each other. Specifically, it is known that DP strongly protects user privacy when a parameter (a.k.a. privacy budget) $\epsilon$ is small, e.g., $\epsilon \leq 1$ [44]. The shuffling significantly reduces $\epsilon$ and therefore significantly improves utility at the same value of $\epsilon$. To date, the shuffle model has been successfully applied to tabular data [51, 63] and gradients [26, 47] in federated learning. We apply the shuffle model to graph data to accurately count subgraphs within one round.

The main challenge in subgraph counting in the shuffle model is that each user’s neighbor list is high-dimensional data, i.e., $n$-dim binary string where $n$ is the number of users. Consequently, applying the RR to each bit of the neighbor list, as in the existing work [32, 33, 67, 68], results in an extremely large privacy budget $\epsilon$ even after applying the shuffling (see Section 4.1 for more details).

We address this issue by introducing a new, basic technique called wedge shuffling. In graphs, a wedge between $v_i$ and $v_j$ is defined by a 2-hop path with endpoints $v_i$ and $v_j$. For example, in Figure 1, there are two wedges between $v_2$ and $v_3$: $v_2-v_4-v_3$ and $v_2-v_4^*-v_3$. In other words, users $v_1$ and $v_4$ have a wedge between $v_2$ and $v_3$, whereas $v_2, \ldots, v_8$ do not. Each user obfuscates such wedge information by the RR, and the shuffler randomly shuffles them. Because the wedge information (i.e., whether there is a wedge between a specific user-pair) is one-dimensional binary data, it can be sent with small noise and small $\epsilon$. In addition, the wedge is the main component of several subgraphs, such as triangles, 4-cycles, and 3-hop paths [62]. Since the wedge has little noise, we can accurately count these subgraphs based on wedge shuffling.

We apply wedge shuffling to triangle and 4-cycle counting tasks with several additional techniques. For triangles, we first propose an algorithm that counts triangles involving the user-pair at the endpoints of the wedges by locally sending an edge between the user-pair to the data collector. Then we propose an algorithm to count triangles in the entire graph by sampling disjoint user-pairs, which share no common users (i.e., no user falls in two pairs). We also propose a technique to reduce the variance of the estimate by ignoring sparse user-pairs, where either of the two users has a very small degree. For 4-cycles, we propose an algorithm to calculate an unbiased estimate of the 4-cycle count from that of the wedge count via bias correction.

We provide upper bounds on the estimation error for our triangle and 4-cycles counting algorithms. Through comprehensive evaluation, we show that our algorithms accurately estimate these subgraph counts within one round under the shuffle model.

Our Contributions. Our contributions are as follows:

- We propose a wedge shuffle technique to enable privacy amplification of graph data. To our knowledge, we are the first to shuffle graph data (see Section 2 for more details).
- We propose one-round triangle and 4-cycle counting algorithms based on our wedge shuffle technique. For triangles, we propose three additional techniques: sending local edges, sampling disjoint user-pairs, and variance reduction by ignoring sparse user-pairs. For 4-cycles, we propose a bias correction technique. We show upper bounds on the estimation error for each algorithm.
- We evaluate our algorithms using two real graph datasets. Our experimental results show that our one-round shuffle algorithms significantly outperform one-round local algorithms in terms of accuracy and achieve a small estimation error (relative error $\ll 1$) with a reasonable privacy budget, e.g., smaller than 1 in edge DP.

In the full version of this paper [34], we show that our triangle algorithm is also useful for accurately estimating the clustering coefficient within one round. We can use our algorithms to analyze the clustering tendency or the effectiveness of friend suggestions in decentralized social networks by introducing a shuffler. We implemented our algorithms in C/C++. Our code is available on GitHub [6]. The proofs of all statements in the main body are given in [34].

2 RELATED WORK

Non-private Subgraph Counting. Subgraph counting has been extensively studied in a non-private setting (see [57] for a recent survey). Examples of subgraphs include triangles [13, 23, 42, 65], 4-cycles [12, 36, 48, 50], $k$-stars [7, 28], and $k$-hop paths [15, 37].

Here, the main challenge is to reduce the computational time of counting these subgraphs in large-scale graph data. One of the simplest approaches is edge sampling [13, 23, 65], which randomly samples edges in a graph. Edge sampling outperforms other sampling methods (e.g., node sampling, triangle sampling) [65] and is also adopted in [33] for private triangle counting.

Although our triangle algorithm also samples user-pairs, ours is different from edge sampling in two ways. First, our algorithm does not sample an edge but samples a pair of users who may or may not be a friend. Second, our algorithm samples user-pairs that share no common users to avoid the increase of the privacy budget $\epsilon$ as well as to reduce the time complexity (see Section 5 for details).

Private Subgraph Counting. Differentially private subgraph counting has been widely studied, and the previous work assumes either the central [20, 38, 40] or local [32, 33, 62, 67, 68] models. The central model assumes a centralized social network and has a data breach issue, as explained in Section 1.

Subgraph counting in the local model has recently attracted attention. Sun et al. [62] propose subgraph counting algorithms assuming that each user knows all friends’ friends. However, this assumption does not hold in many social networks; e.g., Facebook users can change their settings so that anyone cannot see their friend lists. Therefore, we make a minimal assumption – each user knows only her friends.
In this setting, recent studies propose triangle [32, 33, 67, 68] and k-star [32] counting algorithms. For k-stars, Imola et al. [32] propose a one-round algorithm that is order optimal and show that it provides a very small estimation error. For triangles, they propose a one-round algorithm that applies the RR to each bit of the neighbor list and then calculates an unbiased estimate of triangles from the noisy graph. We call this algorithm RR\(_\Delta\). Imola et al. [33] show that RR\(_\Delta\) provides a much smaller estimation error than the one-round triangle algorithms in [67, 68]. In [33], they also reduce the time complexity of RR\(_\Delta\) by using the ARR (Asymmetric RR), which samples each 1 (edge) after applying the RR. We call this algorithm ARR\(_\Delta\). In this paper, we use RR\(_\Delta\) and ARR\(_\Delta\) as baselines in triangle counting. For 4-cycles, there is no existing algorithm under LDP, to our knowledge. Thus, we compare our shuffle algorithm with its local version, which does not shuffle the obfuscated data.

For triangles, Imola et al. also propose a two-round local algorithm in [32] and significantly reduce its download cost in [33]. Although we focus on one-round algorithms, we show in the full algorithm in [32] and significantly reduce its download cost in [33].

### 3.1 Notation

Let \(a\in\mathbb{R}\) be the set of real numbers, \(\mathbb{N}\) be the set of natural numbers, and \(\mathbb{N}_0\) be the set of non-negative integers, respectively. For \(a\in\mathbb{N}\), let \([a]\) be the set of natural numbers that do not exceed \(a\), i.e., \([a]=\{1,2,\ldots,a\}\).

We consider an undirected social graph \(G=(V,E)\), where \(V\) represents a set of nodes (users) and \(E\subseteq V\times V\) represents a set of edges (friendships). Let \(n\in\mathbb{N}\) be the number of nodes in \(V\), and \(v_i\in V\) be the \(i\)-th node, i.e., \(V=\{v_1,\ldots,v_n\}\). Let \(L_{(i,j)}\) be the set of indices of users other than \(v_i\) and \(v_j\), i.e., \(L_{(i,j)}=[n]\setminus\{i,j\}\).

Let \(d_i\in\mathbb{Z}_{\geq0}\) be a degree of \(v_i\), \(d_{max}\in\mathbb{N}\) be the maximum degree of \(G\). Let \(\triangle^G:G\rightarrow\mathbb{Z}_{\geq0}\) and \(4\text{-cycle}^G:G\rightarrow\mathbb{Z}_{\geq0}\) be triangle and 4-cycle count functions, respectively. The triangle count function \(\triangle^G\) takes \(G\) as input and outputs the number \(\triangle^G(G)\) of triangles in \(G\), whereas the 4-cycle count function takes \(G\) as input and outputs the number \(4\text{-cycle}^G(G)\) of 4-cycles.

Let \(A=(a_{i,j})\in\{0,1\}^{n\times n}\) be an adjacency matrix corresponding to \(G\). If \((v_i,v_j)\in E\), then \(a_{i,j}=1\); otherwise, \(a_{i,j}=0\). We call \(a_{i,j}\) an edge indicator. Let \(a_i\in\{0,1\}^n\) be a neighbor list of user \(v_i\), i.e., \(a_i\) is the \(i\)-th row of \(A\).

### 3.2 Differential Privacy

**DP and LDP.** We use differential privacy, and more specifically \((\varepsilon,\delta)\)-DP [22], as a privacy metric:

\[
(\varepsilon,\delta)\text{-DP guarantees that two neighboring datasets } D \text{ and } D' \text{ are almost equally likely when } \varepsilon \text{ and } \delta \text{ are close to 0. The parameter } \varepsilon \text{ is called the privacy budget. It is well known that } \varepsilon \leq 1 \text{ is acceptable.}
\]

\[
\Pr[M(D) \in S] \leq e^\varepsilon \Pr[M(D') \in S] + \delta.
\]

We refer to Warner’s RR \(\mathcal{R}\) with parameter \(\varepsilon\) as \(\varepsilon\text{-RR}\).

**Randomized Response.** We use Warner’s RR (Randomized Response) [64] to provide LDP. Given \(\varepsilon\in\mathbb{R}_{\geq0}\), Warner’s RR \(\mathcal{R}_\varepsilon^W: [0,1] \rightarrow [0,1] \) maps \(x \in [0,1] \) to \(y \in [0,1] \) with the probability:

\[
\Pr[\mathcal{R}_\varepsilon^W(x)=y] = \begin{cases} 
\frac{e^{\varepsilon x}}{e^\varepsilon+1} & \text{if } x=y \\
\frac{e^{\varepsilon}}{e^\varepsilon+1} & \text{otherwise}.
\end{cases}
\]

\(\mathcal{R}_\varepsilon^W\) provides \(\varepsilon\text{-LDP}\) in Definition 3.2, where \(X=[0,1]\). We refer to Warner’s RR \(\mathcal{R}_\varepsilon^W\) with parameter \(\varepsilon\) as \(\varepsilon\text{-RR}\).

**DP on Graphs.** For graphs, we can consider two types of DP: edge DP and node DP [30, 56]. Edge DP hides the existence of one edge, whereas node DP hides the existence of one node along with its adjacent edges. In this paper, we focus on edge DP because existing one-round local triangle counting algorithms [32, 33, 67, 68] use edge DP. In other words, we are interested in how much the estimation error is reduced at the same value of \(\varepsilon\) in edge DP by shuffling. Although node DP is much stronger than edge DP, it is much harder to attain and often results in a much larger \(\varepsilon\) [17, 59]. Thus, we leave an algorithm for shuffle node DP with small \(\varepsilon\) (e.g.,
\( \epsilon \leq 1 \) for future work. Another interesting avenue of future work is establishing a lower bound on the estimation error for node DP.

Edge DP assumes that anyone (except for user \( v_i \)) can be an adversary who infers edges of user \( v_i \) and that the adversary can obtain all edges except for edges of \( v_i \) as background knowledge. Note that the central and local models have different definitions of neighboring data in edge DP. Specifically, edge DP in the central model \([56]\) considers two graphs that differ in one edge. In contrast, edge LDP \([55]\) considers two neighbor lists that differ in one bit:

**Definition 3.3 ((\( \epsilon, \delta \))-edge DP [56]).** Let \( n \in \mathbb{N}, \epsilon \in \mathbb{R}_{\geq 0}, \) and \( \delta \in [0, 1] \). A randomized algorithm \( M \) with domain \( \mathcal{G} \) provides \((\epsilon, \delta)\)-edge DP if for any two neighboring graphs \( G, G' \in \mathcal{G} \) that differ in one edge and any \( S \subseteq \text{Range}(M) \),

\[
\Pr[M(G) \in S] \leq e^\epsilon \Pr[M(G') \in S] + \delta.
\]

**Definition 3.4 ((\( \epsilon \))-edge LDP [55]).** Let \( \epsilon \in \mathbb{R}_{\geq 0} \). A local randomizer \( R \) with domain \([0, 1]\) provides \( \epsilon \)-edge LDP if for any two neighbor lists \( a_i, a'_i \in \{0, 1\}^n \) that differ in one bit and any \( S \subseteq \text{Range}(R) \),

\[
\Pr[R(a_i) \in S] \leq e^\epsilon \Pr[R(a'_i) \in S].
\]

As with edge LDP, we define element DP, which considers two adjacency matrices that differ in one bit, in the central model:

**Definition 3.5 ((\( \epsilon, \delta \))-element DP).** Let \( n \in \mathbb{N}, \epsilon \in \mathbb{R}_{\geq 0}, \) and \( \delta \in [0, 1] \). A randomized algorithm \( M \) with domain \( \mathcal{G} \) provides \((\epsilon, \delta)\)-element DP if for any two neighboring graphs \( G, G' \in \mathcal{G} \) that differ in one bit in the corresponding adjacency matrices \( A, A' \in \{0, 1\}^{n \times n} \) and any \( S \subseteq \text{Range}(M) \),

\[
\Pr[M(G) \in S] \leq e^\epsilon \Pr[M(G') \in S] + \delta.
\]

Although element DP and edge DP have different definitions of neighboring data, they are closely related to each other:

**Proposition 3.6.** If a randomized algorithm \( M \) provides \((\epsilon, \delta)\)-element DP, it also provides \((2\epsilon, 2\delta)\)-edge DP.

**Proof.** Adding or removing one edge affects two bits in an adjacency matrix. Thus, by group privacy \([22]\), any \((\epsilon, \delta)\)-element DP algorithm \( M \) provides \((2\epsilon, 2\delta)\)-edge DP. \( \Box \)

Similarly, if a randomized algorithm \( M \) in the central model applies a local randomizer \( R \) providing \( \epsilon \)-edge LDP to each neighbor list \( a_i (1 \leq i \leq n) \), it provides \( 2\epsilon \)-edge DP \([32]\).

In this case, we use the shuffling technique to provide \((\epsilon, \delta)\)-element DP and then Proposition 3.6 to provide \((2\epsilon, 2\delta)\)-edge DP. We also compare our shuffling algorithms providing \((\epsilon, \delta)\)-element DP and \((2\epsilon, 2\delta)\)-edge DP with local algorithms providing \( \epsilon \)-edge LDP and \( 2\epsilon \)-edge DP to see how much the estimation error is reduced by introducing the shuffle model and a very small \( \delta \) (\( \ll \frac{1}{n} \)).

### 3.3 Shuffle Model

We consider the following shuffle model. Each user \( v_i \in V \) obfuscates her personal data using a local randomizer \( R \) providing \( \epsilon_L \)-LDP for \( \epsilon_L \in \mathbb{R}_{\geq 0} \). Note that \( R \) is common to all users. User \( v_i \) encrypts the obfuscated data and sends it to a shuffler. Then, the shuffler randomly shuffles the encrypted data and sends the results to a data collector. Finally, the data collector decrypts them. The common assumption in the shuffle model is that the shuffler and the data collector do not collude with each other. Under this assumption, the shuffler cannot access the obfuscated data, and the data collector cannot link the obfuscated data to the users. Hereinafter, we omit the encryption/decryption process because it is clear from the context.

We use the privacy amplification result by Feldman et al. \([25]\):

**Theorem 3.7 (Privacy amplification by shuffling [25]).** Let \( n \in \mathbb{N} \) and \( \epsilon_L \in \mathbb{R}_{\geq 0} \). Let \( X \) be the set of input data for each user. For any \( i \)-th user, and \( x_{1:n} = (x_1, \ldots, x_n) \in X^n \). Let \( R : X \rightarrow Y \) be a local randomizer providing \( \epsilon_L \)-LDP. Let \( M_S : X^n \rightarrow Y^n \) be an algorithm that given a dataset \( x_{1:n} \) computes \( y_i = R(x_i) \) for \( i \in [n] \), samples a uniform random permutation \( \pi \) over \([n]\), and outputs \( y_{\pi(1)}, \ldots, y_{\pi(n)} \). Then for any \( \delta \in [0, 1] \) such that \( \epsilon_L \leq \log(n) \frac{1}{\log(2/\delta)} \), \( M_S \) provides \((\epsilon, \delta)\)-DP, where

\[
\epsilon = f(n, \epsilon_L, \delta)
\]

and

\[
f(n, \epsilon_L, \delta) = \log \left( 1 + e^\epsilon - 1 \right) e^{\epsilon L} + \left( 8 \sqrt{e^{2\epsilon^2} \log(4/\delta)} + 8e^{2\epsilon L} \right) \frac{e^{\epsilon L}}{n} \right).
\]

Thanks to the shuffling, the shuffled data \( y_{\pi(1)}, \ldots, y_{\pi(n)} \) available to the data collector provides \((\epsilon, \delta)\)-DP, where \( \epsilon \ll \epsilon_L \).

Feldman et al. \([25]\) also propose an efficient method to numerically compute a tighter upper bound than the closed-form upper bound in Theorem 3.7. We use both the closed-form and numerical upper bounds in our experiments. Specifically, we use the numerical upper bounds in Section 7 and compare the numerical bound with the closed-form bound in the full version \([34]\).

Assume that \( \epsilon \) and \( \delta \) in (3) are constants. Then, by solving for \( \epsilon_L \) and changing to big O notation, we obtain \( \epsilon_L = \log(n) + O(1) \). This is consistent with the upper bound \( \epsilon = O(\epsilon_L^2 / \sqrt{n}) \) in \([25]\), from which we obtain \( \epsilon_L = \log(n) + O(1) \). Similarly, the privacy amplification bound in \([18]\) can also be expressed as \( \epsilon_L = \log(n) + O(1) \). We use the bound in \([25]\) because it is the state-of-the-art, as described in Section 2.

### 3.4 Utility Metrics

We use the MSE (Mean Squared Error) in our theoretical analysis and the relative error in our experiments. The MSE is the expectation of the squared error between a true value and its estimate. Let \( f : \mathcal{G} \rightarrow \mathbb{Z}_{\geq 0} \) be a subgraph count function that can be instantiated by \( f^* \) or \( f^\circ \). Let \( \hat{f} : \mathcal{G} \rightarrow \mathbb{R} \) be the corresponding estimator. Let \( MSE : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0} \) be the MSE function, which maps the estimate \( \hat{f}(G) \) to the MSE. Then the MSE can be expressed as \( MSE(\hat{f}(G)) = \mathbb{E}[((f(G) - \hat{f}(G))^2) \right] \), where the expectation is taken over the randomness in the estimator \( \hat{f} \). By the bias-variance decomposition \([52]\), the MSE can be expressed as a summation of the squared bias \( \mathbb{E}[(\hat{f}(G) - f(G))^2] \) and the variance \( \mathbb{V}[(\hat{f}(G) - \mathbb{E}[\hat{f}(G)])^2] \). Thus, for an unbiased estimator \( \hat{f} \) satisfying \( \mathbb{E}[(\hat{f}(G) - f(G))] = 0 \), the MSE is equal to the variance, i.e., \( MSE(\hat{f}(G)) = \mathbb{V}[(\hat{f}(G)) \right] \).

Although the MSE is suitable for theoretical analysis, it tends to be large when the number \( n \) of users is large. This is because the true triangle and 4-cycle counts are very large when \( n \) is large – \( f^*(G) = O(nd^2_{\max}) \) and \( f^\circ(G) = O(nd^3_{\max}) \). Therefore, we use the relative error in our experiments. The relative error is an absolute error
divided by the true value and is given by \( \frac{|f^a(G) - f^b(G)|}{\min|f^a(G), f^b(G)|} \), where \( \eta \in \mathbb{R}_{>0} \) is a small positive value. Following the convention [14, 16, 66], we set \( \eta = \frac{n}{1000} \).

When the relative error is well below 1, the estimate is accurate. Note that the absolute error smaller than 1 would be impossible under DP with meaningful \( \epsilon \) (e.g., \( \epsilon \leq 1 \)), as we consider counting queries. However, the relative error (= absolute error / true count) much smaller than 1 is possible under DP with meaningful \( \epsilon \).

4 Shuffle Model for Graphs

In this work, we apply the shuffle model to graph data to accurately estimate subgraph counts, such as triangles and 4-cycles. Section 4.1 explains our technical motivation. In particular, we explain why it is challenging to apply the shuffle model to graph data. Section 4.2 proposes a wedge shuffle technique to overcome the technical challenge.

4.1 Our Technical Motivation

The shuffle model has been introduced to dramatically reduce the privacy budget \( \epsilon \) (hence the estimation error at the same \( \epsilon \)) in tabular data [51, 63] or gradients [26, 47]. However, it is very challenging to apply the shuffle model to graph data, as explained below.

Figure 2 shows the shuffle model for graph data, where each user \( v_i \) has her neighbor list \( a_i \in \{0,1\}^n \). The main challenge here is that the shuffle model uses a standard definition of LDP for the local randomizer and that a neighbor list is high-dimensional data, i.e., \( n \)-dim binary string. Specifically, LDP in Definition 3.2 requires any pair of inputs \( x \) and \( x' \) to be indistinguishable; i.e., the inequality (1) must hold for all pairs of possible inputs. Thus, if we use the entire neighbor list as input data (i.e., \( a_i = x_i \) in Theorem 3.7), either privacy or utility is destroyed for large \( n \).

To illustrate this, consider the following example. Assume that \( n = 10^5 \) and \( \delta = 10^{-8} \). Each user \( v_i \) applies \( \epsilon_0 \)-RR with \( \epsilon_0 = 1 \) to each bit of her neighbor list \( a_i \). This mechanism is called the randomized neighbor list [55] and provides \( \epsilon_0 \)-edge LDP. However, the privacy budget \( \epsilon_L \) in the standard LDP (Definition 3.2) is extremely large – by group privacy [22], \( \epsilon_L = n\epsilon_0 = 10^3 \). Because \( \epsilon_L \) is much larger than \( \log_{\frac{1000}{1000}} = 8.09 \), we cannot use the privacy amplification result in Theorem 3.7. This is evident from the fact that the shuffled data \( y_{\pi(1)}, \ldots, y_{\pi(n)} \) are easily re-identified when \( n \) is large. If we use \( \epsilon_0 \)-RR with \( \epsilon_0 = \frac{n}{10} \), we can use the amplification result (as \( \epsilon_L = \frac{n}{1000} = 1 \)). However, it makes obfuscated data almost a random string and destroys the utility because \( \epsilon_0 \) is too small.

In this work, we address this issue by introducing a basic technique, which we call wedge shuffling.
In this work, we focus on triangles and 4-cycles and present algorithms with upper bounds on the estimation error based on our wedge shuffle technique.

5 TRIANGLE COUNTING BASED ON WEDGE SHUFFLING

Based on our wedge shuffle technique, we first propose a one-round triangle counting algorithm. Section 5.1 describes the overview of our algorithms. Section 5.2 proposes an algorithm for counting triangles involving a specific user-pair as a building block of our triangle counting algorithm. Section 5.3 proposes our triangle counting algorithm. Section 5.4 proposes a technique to significantly reduce the variance in our triangle counting algorithm. Section 5.5 summarizes the performance guarantees of our triangle algorithms.

5.1 Overview

Our wedge shuffle technique tells the data collector the number of common friends of \(v_i\) and \(v_j\). However, this information is not sufficient to count triangles in the entire graph. Therefore, we introduce three additional techniques: (i) sending local edges, (ii) sampling disjoint user-pairs, and (iii) variance reduction by ignoring sparse user-pairs. Below, we briefly explain each technique.

Sending Local Edges. First, we consider the problem of counting triangles involving a specific user-pair \((v_i, v_j)\) and propose an algorithm to send local edges between \(v_i\) and \(v_j\), along with shuffled wedges, to the data collector. We call this the WSLE (Wedge Shuffling with Local Edges) algorithm.

Wedge information (small noise)

\(\nu_k \in (0,1)\)

\(\nu_j \in (0,1)\)

Figure 4: Overview of our WSLE (Wedge Shuffling with Local Edges) algorithm with inputs \(v_i\) and \(v_j\).

Figure 5: Overview of our triangle counting algorithm. We use our WSLE algorithm with each user-pair.

5.2 WSLE (Wedge Shuffling with Local Edges)

Algorithm. We first propose the WSLE algorithm as a building block of our triangle counting algorithm. WSLE counts triangles involving a specific user-pair \((v_i, v_j)\).

Algorithm 2 shows WSLE. Let \(F_{ij}^L : G \rightarrow Z_{\geq 0}\) be a function that takes \(G \in G\) as input and outputs the number \(F_{ij}^L(G)\) of triangles involving \((v_i, v_j)\) in \(G\). Let \(\hat{F}_{ij}^L(G) \in \mathbb{R}\) be an estimate of \(F_{ij}^L(G)\).

We first call the function LocalPrivacyBudget, which calculates a local privacy budget \(\epsilon_L\) from \(n, \epsilon, \delta\) and \(\ell(1)\). Specifically, this function calculates \(\epsilon_L\) such that \(\epsilon_L\) is a closed-form upper bound (i.e., \(\epsilon = f(n, \epsilon, \delta)\) in (2)) or numerical upper bound in the shuffle model with \(n\) users. Given \(\epsilon_L\), we can easily calculate the closed-form or numerical upper bound \(\epsilon_L\) by \(\ell(3)\) and the open source code in [25]1, respectively. Thus, we can also easily calculate \(\epsilon_L\) from \(\epsilon\) by calculating a lookup table for pairs \((\epsilon, \ell(1))\) in advance.

Then, we run our wedge shuffle algorithm WS in Algorithm 1 (line 2); i.e., each user \(v_k \in L_{-(i,j)}\) sends her obfuscated wedge indicator \(y_k = R_{ij}^L(w_{i-k,j-k})\) to the shuffled, and the shuffled sends

1https://github.com/apple/ml-shuffling-amplification.
Theoretical Properties.
Below, we show some theoretical properties of WSLE. First, we prove that the estimate \( \hat{f}_{ij}(G) \) is unbiased:

**Theorem 5.1.** For any indices \( i, j \in [n] \), the estimate produced by WSLE satisfies \( \mathbb{E}[\hat{f}_{ij}(G)] = f_{ij}(G) \).

Next, we show the MSE (= variance). Recall that in the shuffle model, \( \ell_L = \log n + O(1) \) when \( \varepsilon \) and \( \delta \) are constants. We show the MSE for a general case and for the shuffle model:

**Theorem 5.2.** For any indices \( i, j \in [n] \), the estimate produced by WSLE provides the following utility guarantee:

\[
\text{MSE}(\hat{f}_{ij}(G)) = \mathbb{V}[\hat{f}_{ij}(G)] \leq \frac{\sigma_{1y} + 4(1 - 2q)^2 d_{max}^2}{(1 - 2q)^2} \leq \text{err}_{WSLE}(n, d_{max}, q, q_L).
\]

(5)

When \( \varepsilon \) and \( \delta \) are constants and \( \ell_L = \log n + O(1) \), we have

\[
\text{err}_{WSLE}(n, d_{max}, q, q_L) = O(d_{max}^2).
\]

(6)

The equation (6) follows from (5) because \( q_L = \frac{1}{e\ell_L} = \frac{1}{n^\varepsilon \ell_L^{1 + \varepsilon}} \). Because WSLE is a building block for our triangle counting algorithm, we introduce the notation \( \text{err}_{WSLE}(n, d_{max}, q, q_L) \) for our upper bound in (5). Observing (5), if we do not use the shuffling technique (i.e., \( \ell_L = \varepsilon \)), then \( \text{err}_{WSLE}(n, d_{max}, q, q_L) = O(n + d_{max}) \) when \( \varepsilon \) and \( \delta \) are constants. In contrast, in the shuffle model where we have \( \ell_L = \log n + O(1) \), then \( \text{err}_{WSLE}(n, d_{max}, q, q_L) = O(d_{max}^2) \). This means that wedge shuffling reduces the MSE from \( O(n + d_{max}) \) to \( O(d_{max}^2) \), which is significant when \( d_{max} \ll n \).

5.3 Triangle Counting
Based on WSLE, we propose an algorithm that counts triangles in the entire graph \( G \). We denote this algorithm by WShuffle\(_\Delta\), as it applies wedge shuffling to triangle counting.

Algorithm 3 shows WShuffle\(_\Delta\). First, the data collector samples disjoint user-pairs, ensuring that no user falls in two pairs. Specifically, it calls the function RandomPermutation, which samples a uniform random permutation \( \sigma \) over \( [n] \) (line 4). Then, it samples disjoint user-pairs as \( (\sigma_1(1), \sigma_2(2)), (\sigma_1(3), \sigma_2(4)), \ldots, (\sigma_1(2t-1), \sigma_2(2t)) \), where \( t \in [\frac{n}{2}] \). The parameter \( t \) represents the number of users and controls the trade-off between the MSE and the time complexity. When \( t = [\frac{n}{2}] \), the MSE is minimized and the time complexity is maximized. The data collector sends the sampled user-pairs to users (line 2).

Then, we run our wedge algorithm WSLE in Algorithm 2 with each sampled user-pair as input (lines 3-5). Finally, the data collector estimates the triangle count \( f_\Delta(G) \) as follows:

\[
\hat{f}_\Delta(G) = \frac{\sum_{t=1}^{t} a_{ij}(G)}{n}.
\]

(7)

(6) Note that a single triangle is never counted by more than one user-pair, as the user-pairs never overlap. Later, we prove that \( \hat{f}_\Delta(G) \) in (7) is unbiased.

**Theoretical Properties.** We prove that WShuffle\(_\Delta\) provides DP:

**Theorem 5.3.** WShuffle\(_\Delta\) provides \((\varepsilon, \delta)\)-element DP and \((2\varepsilon, 2\delta)\)-edge DP.

Theorem 5.3 comes from the fact that WSLE with a user-pair \( (u_i, u_j) \) provides \((\varepsilon, \delta)\)-DP for each element in the \( ij \)-th and \( ji \)-th columns of the adjacency matrix \( A \) and that WShuffle\(_\Delta\) samples disjoint user-pairs, i.e., it uses each element of \( A \) at most once.

Note that running WSLE with all \( \binom{n}{2} \) user-pairs provides \((n - 2)\varepsilon, (n - 2)\delta)\)-DP, as it uses each element of \( A \) at most \( n - 2 \) times. The
privacy budget is very large, even using the advanced composition [22, 35]. We avoid this issue by sampling user-pairs that share no common users.

We also prove that WShuffle₂ provides an unbiased estimate:

**Theorem 5.4.** The estimate produced by WShuffle₂ satisfies
\[ \mathbb{E}[\hat{f}^α(G)] = f^α(G). \]

Next, we analyze the MSE (= variance) of WShuffle₂. This analysis is non-trivial because WShuffle₂ samples each user-pair without replacement. In this case, sample voters are not independent. However, we can prove that the variance of the sum of user-pairs in Theorem 5.2. This brings us to the following result:

**Theorem 5.5.** The estimate produced by WShuffle₂ provides the following utility guarantee:
\[ \text{MSE}(\hat{f}^α(G)) = \mathbb{E}[(\hat{f}^α(G) - f^α(G))^2]. \]

For example, assume that \( t = \left\lceil \frac{n}{4} \right\rceil \). When we do not shuffle wedges (i.e., \( \ell = n \)), then err\_WSLE\( n, d_{\max}, q, q_l \) = \( O(n^2 d_{\max}^2) \), and MSE in (8) is \( O(n^4 + n d_{\max}^2) \). When we shuffle wedges, the MSE is \( O(n^3 d_{\max}^2) \). Thus, when we ignore the factor of \( d_{\max} \), our wedge shuffle technique reduces the MSE from \( O(n^4) \) to \( O(n^3) \) in triangle counting. The factor of \( n^3 \) is caused by the RR for local edges. This is intuitive because a large amount of noise is added to the local edges.

Finally, we analyze the time complexity of WShuffle₂. The time complexity of running WSLE with all \( \binom{n}{2} \) user-pairs is \( O(n^3) \), as there are \( O(n^3) \) user-pairs in total and WSLE requires the time complexity of \( O(n) \). In contrast, the time complexity of WShuffle₂ with \( t = \left\lceil \frac{n}{4} \right\rceil \) is \( O(n^2) \) because it samples \( O(n) \) user-pairs. Thus, WShuffle₂ reduces the time complexity from \( O(n^3) \) to \( O(n^2) \) by user-pair sampling. We can further reduce the time complexity at the cost of increasing the MSE by setting \( t \) small, i.e., \( t \ll \left\lceil \frac{n}{4} \right\rceil \).

### 5.4 Variance Reduction

**Algorithm.** WShuffle₂ achieves the MSE of \( O(n^3) \) when we ignore the factor of \( d_{\max} \). To provide a smaller estimation error, we propose a variance reduction technique that ignores sparse user-pairs. We denote our triangle counting algorithm with the variance reduction technique by WShuffle*₂.

As explained in Section 5.3, the factor of \( n^3 \) is caused by the RR for local edges. However, most user-pairs \( v_i \) and \( v_j \) have a very small minimum degree \( d_i, d_j \ll d_{\max} \), and there is no edge \((v_i, v_j)\) between them in almost all cases. In addition, even if there is an edge \((v_i, v_j)\), the number of triangles involving the sparse user-pair is very small (at most \( \min(d_i, d_j) \)) and can be approximated by 0. By ignoring such sparse user-pairs, we can dramatically reduce the variance of the RR for local edges at the cost of a small bias. This is an intuition behind our variance reduction technique.

Algorithm 4 shows WShuffle*₂. This algorithm detects sparse user-pairs based on the degree information. However, user \( v_i \)'s degree \( d_i \) can leak the information about edges of \( v_i \). Thus, WShuffle*₂ calculates a differentially private estimate of \( d_i \) within one round. Specifically, WShuffle*₂ uses two privacy budgets: \( \ell_1, \ell_2 \in \mathbb{R}_{\geq 0} \). The first budget \( \ell_1 \) is for privately estimating \( d_i \), whereas the second budget \( \ell_2 \) is for WSLE. Lines 1 to 5 in Algorithm 4 are the same as those in Algorithm 3, except that Algorithm 4 uses \( \ell_2 \) to provide \((\ell_2, \delta)\)-element DP. After these processes, each user \( v_i \) adds the Laplacian noise Lap\( \left( \frac{1}{\delta} \right) \) with mean 0 and scale \( \frac{1}{\delta} \) to her degree \( d_i \) and sends the noisy degree \( \tilde{d}_i \) (\( = d_i + \text{Lap} \left( \frac{1}{\delta} \right) \)) to the data collector (lines 6-8). Because the sensitivity [22] of \( d_i \) (the maximum distance of \( d_i \) between two neighbor lists that differ in one bit) is 1, adding Lap\( \left( \frac{1}{\delta} \right) \) to \( d_i \) provides \( \text{element DP} \).

Then, the data collector estimates the average degree \( d_{\text{avg}} \) as
\[ d_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \tilde{d}_i \] sets a threshold \( d_{\text{th}} \) of the minimum degree to \( d_{\text{th}} = c d_{\text{avg}} \), where \( c \in \mathbb{R}_{\geq 0} \) is a small positive number, e.g., \( c \in [1, 10] \) (line 9). Finally, the data collector estimates \( f^\alpha(G) \) as
\[ \hat{f}^\alpha(G) = \frac{n(n-1)}{2} \sum_{i,j} \hat{f}_{\sigma(i),\sigma(j)}(G). \]

**Table:** Theorem 4. Estimate \( \hat{f}^\alpha(G) \) of \( f^\alpha(G) \).

/* Sample disjoint user-pairs */
1 [d] \( \sigma \leftarrow \text{RandomPermutation}(\{1, 2, \ldots, n\}); \)
2 [d] Send \((\sigma(1), \sigma(2)), \ldots, (\sigma(t-1), \sigma(t))\) to users;
3 foreach \( i \in \{1, 3, \ldots, t-1\} \) do
4 \[ \hat{f}^\alpha_{\sigma(i),\sigma(i+1)}(G) \leftarrow \text{WSLE}(A, \ell_2, \sigma(i), \sigma(i+1)); \]
5 end /* Send noisy degrees */
6 for \( i = 1 \to n \) do
7 \[ [v_i] \tilde{d}_i \leftarrow d_i + \text{Lap} \left( \frac{1}{\delta} \right); \] Send \( \tilde{d}_i \) to the data collector;
8 end /* Calculate a variance-reduced estimate */
9 [d] \( d_{\text{avg}} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \tilde{d}_i; \) Send \( d_{\text{th}} \) to the data collector;
10 [d] \( D \leftarrow \{i | i = 1, 3, \ldots, 2t-1, \min(\tilde{d}_i, \tilde{d}_{i+1}) > d_{\text{th}}\}; \)
11 [d] \( \hat{f}^\alpha(G) \leftarrow \frac{n(n-1)}{2} \sum_{i \in D} \hat{f}^\alpha_{\sigma(i),\sigma(i+1)}(G); \)
12 [d] return \( \hat{f}^\alpha(G) \)

Algorithm 4: Our triangle counting algorithm with variance reduction WShuffle*₂. WSLE is shown in Algorithm 2.
The parameter \( c \) controls the trade-off between the bias and variance of the estimate \( \hat{f}^\Delta(G) \). The larger \( c \), the more user-pairs are ignored. Thus, as \( c \) increases, the bias increases, and the variance is reduced. In practice, a small \( c \) not less than 1 results in a small MSE because most real graphs are scale-free networks that have a power-law degree distribution [10]. In the scale-free networks, most users’ degrees are smaller than the average degree \( d_{\text{avg}} \). For example, in the BA (Barabási-Albert) graph model [10, 29], most users’ degrees are \( d_{\text{avg}}/2 \). Thus, if we set \( c \in [1, 10] \), for example, then most user-pairs are ignored (i.e., \(|D| \ll 1\)), which leads to a significant reduction of the variance at the cost of a small bias.

Recall that the parameter \( t \) in \( W\text{Shuffle}_\lambda \) controls the trade-off between the MSE and the time complexity. Although \( W\text{Shuffle}_\lambda \) always samples \( t \) disjoint user-pairs, we can modify \( W\text{Shuffle}_\lambda \) so that it stops sampling user-pairs right after the estimate \( \hat{f}^\Delta(G) \) in (10) is converged. We can also sample dense user-pairs \((v_i, v_j)v_i \) with large noisy degrees \( d_i \) and \( d_j \) at the beginning (e.g., by sorting users in descending order of noisy degrees) to improve the MSE for small \( t \). Evaluating such improved algorithms is left for future work.

**Theoretical Properties.** As with \( W\text{Shuffle}_\lambda \), \( W\text{Shuffle}_\Delta \) provides the following privacy guarantee:

**Theorem 5.6.** \( W\text{Shuffle}_\lambda \) provides \((\epsilon_1 + \epsilon_2, \delta)-\text{element DP and } (2(\epsilon_1 + \epsilon_2), 2\delta)-\text{edge DP.} \)

Next, we analyze the bias of \( W\text{Shuffle}_\lambda \). Here, we assume users have a small degree using parameters \( \lambda \in \mathbb{R}_{\geq 0} \) and \( \alpha \in [0, 1) \):

**Theorem 5.7.** Suppose that in \( G \), there exist \( \lambda \in \mathbb{R}_{\geq 0} \) and \( \alpha \in [0, 1) \) such that at most \( n^\alpha \) users have a degree larger than \( \lambda d_{\text{avg}} \). Suppose \( W\text{Shuffle}_\lambda \) is run with \( c \geq \lambda \). Then, the estimator produced by \( W\text{Shuffle}_\lambda \) provides the following bias guarantee:

\[
\text{Bias}[\hat{f}^\Delta(G)] = |\mathbb{E}[\hat{f}^\Delta(G)] - f^\Delta(G)| \leq \frac{nc^2d_{\text{avg}}^2}{3} + 4n^\alpha \frac{3}{\epsilon^2} \tag{11}
\]

The values of \( \lambda \) and \( \alpha \) depend on the original graph \( G \). In the scale-free networks, \( \alpha \) is small for a moderate value of \( \lambda \). For example, in the BA graph with \( n = 107614 \) and \( d_{\text{avg}} = 200 \) used in the full version [34], \( \alpha = 0.5, 0.6, 0.8, \) and 0.9 when \( \lambda = 10.1, 5.4, 1.6, \) and 0.9, respectively. When \( c = 1 \) and \( \epsilon_1 = 1 \), their bias can be expressed as 

\[
O(n^2d_{\text{avg}}^3). \tag{12}
\]

Finally, we show the variance of \( W\text{Shuffle}_\lambda \). This result assumes that \( c \) is bigger (\( \geq (1-\alpha)\log n/\epsilon d_{\text{avg}} \)) than \( \lambda \). We assume this because otherwise, many sparse users (with \( d_i \leq \lambda d_{\text{avg}} \)) have a noisy degree \( d_i \geq \lambda d_{\text{avg}} \), causing the set \( D \) to be noisy. In practice, the gap between \( c \) and \( \lambda \) is small because \( \log n \) is much smaller than \( d_{\text{avg}} \).

**Theorem 5.8.** Suppose that in \( G \), there exist \( \lambda \in \mathbb{R}_{\geq 0} \) and \( \alpha \in [0, 1) \) such that at most \( n^\alpha \) users have a degree larger than \( \lambda d_{\text{avg}} \). Suppose \( W\text{Shuffle}_\lambda \) is run with \( c \geq \lambda + (1-\alpha)\log n/\epsilon d_{\text{avg}} \). Then, the estimator produced by \( W\text{Shuffle}_\lambda \) provides the following variance guarantee:

\[
\forall[f^\Delta(G)] \leq \frac{n^2d_{\text{max}}^4}{9} + \frac{2n^{2+2\alpha}}{9t} \text{err}_{\text{WLocal}}(n, d_{\text{max}}, q, q_L) + \frac{n^{2+4\alpha}d_{\text{max}}^4}{36t}. \tag{12}
\]

When \( \epsilon_1, \epsilon_2, \) and \( \delta \) are constants, \( \epsilon_L = \log n = O(1), \) and \( t = \lceil \frac{2}{\epsilon^2} \rceil \),

\[
\forall[f^\Delta(G)] \leq O(n^2d_{\text{max}}^4 + n^{1+2\alpha}d_{\text{max}}^4). \tag{13}
\]

The first time in (12) is actually \( \frac{2n^{2+2\alpha}}{9t} \text{err}_{\text{WLocal}}(n, d_{\text{max}}, q, q_L) \) and is much smaller than \( n^2d_{\text{max}}^4 \). We express it as \( O(n^2d_{\text{max}}^4) \) in (13) for simplicity. See the full version [34] for details.

For the first two, second, and third terms in (12) are caused by the randomness in the choice of \( D \), the RR, and user-pair sampling, respectively. By (13), our variance reduction technique reduces the variance from \( O(n^5) \) to \( O(n^\alpha) \) where \( \gamma \in \{2, 3\} \) when we ignore the factor of \( d_{\text{max}} \). Because the MSE is the sum of the squared bias and the variance, it is also \( O(n^\alpha) \).

The value of \( \gamma \) in our bound \( O(n^\alpha) \) depends on the parameter \( \epsilon \) in \( W\text{Shuffle}_\lambda \). For example, in the BA graph \( n = 107614, d_{\text{avg}} = 200 \), \( \gamma = 2.2, 2.6, \) and 2.8 \((\alpha = 0.5, 0.6, 0.8, \) and 0.9\) when \( c = 10.4, 5.6, 1.7, \) and 1.0, respectively, and \( \epsilon_1 = 1 \). Thus, the variance decreases with increase in \( \epsilon \). However, by (11), a larger \( \epsilon \) results in a larger bias. In our experiments, we show that \( W\text{Shuffle}_\lambda \) provides a small estimation error when \( \epsilon = 1 \) to 4. When \( \epsilon = 1 \), \( W\text{Shuffle}_\lambda \) empirically works well despite a large \( \gamma \) because most users’ degrees are smaller than \( d_{\text{avg}} \) in practice, as explained above. This indicates that our upper bound in (13) might not be tight when \( \epsilon \) is around 1. Improving the bound is left for future work.

### 5.5 Summary

Table 1 summarizes the performance guarantees of one-round triangle algorithms providing edge DP. Here, we consider a variant of \( W\text{Shuffle}_\lambda \) that does not shuffle wedges (i.e., \( \epsilon_2 = \epsilon \)) as a one-round local algorithm. We call this variant \( W\text{Local}_\Delta \) (Wedge Local). We also show the variance of \( ARR_\Delta \) [33] and \( RR_\Delta \) [32]. The time complexity of \( ARR_\Delta \) is \( O(n^3) \), and that of \( RR_\Delta \) is \( O(n^2) \) when we set the sampling probability \( p_0 \in [0, 1] \) of the ARR to \( p_0 = O(n^{-1/3}) \). We prove the variance of \( ARR_\Delta \) in this case and \( RR_\Delta \) in the full version [34]. We do not show the other one-round local algorithms [67, 68] in Table 1 for two reasons: (i) they have the time complexity of \( O(n^3) \) and suffer from a larger estimation error than \( RR_\Delta \) [33]; (ii) their upper-bounds on the variance and bias are unclear.

Table 1 shows that our \( W\text{Shuffle}_\lambda \) dramatically outperforms the three local algorithms – when we ignore \( d_{\text{max}} \), the MSE of \( W\text{Shuffle}_\lambda \) is \( O(n^\alpha) \) where \( \epsilon \in \{2, 3\} \), whereas that of the local algorithms is \( O(n^5) \) or \( O(n^3) \). We also show this through experiments.

---

1. The first term in (12) is actually \( \frac{2n^{2+2\alpha}}{9t} \text{err}_{\text{WLocal}}(n, d_{\text{max}}, q, q_L) \) and is much smaller than \( n^2d_{\text{max}}^4 \). We express it as \( O(n^2d_{\text{max}}^4) \) in (13) for simplicity. See the full version [34] for details.
2. Technically speaking, the algorithms of \( RR_\Delta \) and the one-round local algorithms in [67, 68] involve counting the number of triangles in a dense graph. This can be done in time \( O(n^\alpha) \), where \( \epsilon \in \{2, 3\} \) and \( O(n^\alpha) \) is the time required for matrix multiplication. However, these algorithms are of theoretical interest, and they do not outperform naive matrix multiplication except for very large matrices [8]. Thus, we assume implementations that use naive matrix multiplication in \( O(n^\alpha) \) time.
Note that both ARR and RB provide pure DP (δ = 0), whereas our shuffle algorithms provide approximate DP (δ > 0). However, it would not make a noticeable difference, as δ is sufficiently small (e.g., δ = 10^{-8} \approx \frac{1}{2} in our experiments).

Comparison with the Central Model. Finally, we note that our WSShuffle* is worse than algorithms in the central model in terms of the estimation error.

Specifically, Imola et al. [32] consider a central algorithm that adds the Laplacian noise Lap(\frac{d_{\text{max}}}{\varepsilon}) to the true count \( f^\wedge_i(G) \) and outputs \( \frac{f^\wedge_i(G) + \text{Lap}(\frac{d_{\text{max}}}{\varepsilon})}{\varepsilon} \). This central algorithm provides (ε, 0)-edge DP. In addition, the estimate is unbiased, and the variance is \( 2\delta_{\text{max}}^2 = O(d_{\text{max}}^2) \). Thus, the central algorithm provides a much smaller MSE (\approx variance) than WSShuffle*.

However, our WSShuffle* is preferable to central algorithms in terms of the trust model—the central model assumes that a single party accesses personal data of all users and therefore has a risk that the entire graph is leaked from the party. WSShuffle* can also be applied to decentralized social networks, as described in Section 1.

### 6 4-CYCLE COUNTING BASED ON WEDGE SHUFFLING

Next, we propose a one-round 4-cycle counting algorithm in the shuffle model. Section 6.1 explains its overview. Section 6.2 proposes our 4-cycle counting algorithm and shows its theoretical properties. Section 6.3 summarizes the performance guarantees of our 4-cycle algorithms.

#### 6.1 Overview

We apply our wedge shuffling technique to 4-cycle counting with two additional techniques: (i) bias correction and (ii) sampling disjoint user-pairs. Below, we briefly explain each of them.

**Bias Correction.** As with triangles, we begin with the problem of counting 4-cycles involving specific users \( v_i \) and \( v_j \). We can leverage the noisy wedges output by our wedge shuffle algorithm WS to estimate such a 4-cycle count. Specifically, let \( f^\wedge_{i,j} : G \rightarrow \mathbb{Z} \) be a function that, given \( G \in G \), outputs the number \( f^\wedge_{i,j}(G) \) of 4-cycles for which users \( v_i \) and \( v_j \) are opposite nodes, i.e., the number of unordered pairs \((k, k')\) such that \( v_i - v_k - v_j - v_{k'} = v_i \) is a path in \( G \). Each pair \((k, k')\) satisfies the above requirement if and only if \( v_i - v_k - v_j \) and \( v_i - v_{k'} - v_j \) are wedges in \( G \). Thus, we have \( f^\wedge_{i,j}(G) = \binom{\sigma_i}{2} \binom{\sigma_j}{2} \), where \( f^\wedge_{i,j} \) is the number of wedges between \( v_i \) and \( v_j \). Based on this, we calculate an unbiased estimate \( f^\wedge_{i,j} \) of the wedge count using WS. Then, we calculate an estimate of the 4-cycle count as \( \hat{f}^\wedge_{i,j} \). Here, it should be noted that the estimate \( \hat{f}^\wedge_{i,j} \) is biased, as proved later. Therefore, we perform bias correction— we subtract a positive value from the estimate to obtain an unbiased estimate \( \hat{f}^\wedge_{i,j}(G) \) of the 4-cycle count.

Note that unlike WSLE, no edge between \((u_i, v_j)\) needs to be sent. In addition, thanks to the privacy amplification by shuffling, all wedges can be sent with small noise.

#### 6.2 4-Cycle Counting Algorithm

Algorithm 5 shows our 4-cycle counting algorithm WSShuffle*.

WS is shown in Algorithm 1.

**Sampling Disjoint User-Pairs.** Having an estimate \( \hat{f}^\wedge_{i,j}(G) \), we turn our attention to estimating 4-cycle count \( f^\wedge(G) \) in the entire graph \( G \). As with triangles, a naive solution using estimates \( \hat{f}^\wedge_{i,j}(G) \) for all \( \binom{n}{2} \) user-pairs \((u_i, v_j)\) results in very large \( \varepsilon \) and \( \delta \). To avoid this, we sample disjoint user-pairs and obtain an unbiased estimate of \( f^\wedge(G) \) from them.

**Algorithm 5: Our 4-cycle counting algorithm WSShuffle*.

\begin{algorithm}[H]
\begin{algorithmic}[1]
\State \textbf{Input:} Adjacency matrix \( A \in \{0, 1\}^{n \times n} \), \( \varepsilon, \delta \in \mathbb{R}_{\geq 0} \), \( \beta \in [0, 1] \), \( t \in \left[ \frac{1}{\max_i d_i} \right] \).
\State \textbf{Output:} Estimate \( \hat{f}^\wedge(G) \) of \( f^\wedge(G) \).
\State 1: \( \varepsilon_1 \leftarrow \text{LocalPrivacyBudget}(n, \varepsilon, \delta) \).
\State 2: \( [d] q_L \leftarrow \frac{1}{\varepsilon_1 \varepsilon t} \).
\State 3: \( [d] \sigma \leftarrow \text{RandomPermutation}(n) \).
\State 4: \( [d] \text{Send} (\sigma_i, \sigma_j) \), \( i \neq j \), \( \sigma(2t-1), \sigma(2t-2) \) to users;
\State 5: \textbf{foreach} \( i \in \{1, 3, \ldots, 2t-1\} \) \textbf{do}
\State 6: \( y_{\pi_i(k)} \leftarrow \text{WS}(A, \varepsilon_L, (\sigma_i, \sigma_{i+1})); \)
\State 7: \( [d] \hat{f}^\wedge_{i,\sigma(i)}(G) \leftarrow \sum_{k \in L_\sigma(\sigma_i, \sigma(i+1))} \frac{u_k - q_L}{1 - 2q_L} \).
\State 8: \( [d] \hat{f}^\wedge_{i,\sigma(i)}(G) \leftarrow \hat{f}^\wedge_{i,\sigma(i)}(G) + \frac{\hat{f}^\wedge_{i,\sigma(i)}(G) - 1}{2} - \frac{2 - 2q_L(1 - q_L)}{1 - 2q_L} \).
\State 9: \textbf{end}
\State 10: \( \text{\textbf{// Calculate an unbiased estimate}} \)
\State 11: \( \hat{f}^\wedge(G) \leftarrow \frac{n(n-1)}{2t} \sum_{i=1, 3, \ldots, 2t-1} \hat{f}^\wedge_{i,\sigma(i)}(G) \).
\State 11: \( \text{\textbf{return}} \hat{f}^\wedge(G) \).
\end{algorithmic}
\end{algorithm}

\[ f^\wedge_{i,j}(G) = \sum_{k \in L_{\sigma(i,j)}} \frac{u_k - q_L}{1 - 2q_L} \]  

Later, we will prove that \( \hat{f}^\wedge_{i,j}(G) \) is an unbiased estimator. As with (4), this estimate involves the sum over the set \( \{y_{\pi(k)}\} \) and does not require knowing the permutation \( \pi \) produced by the shuffler. Then, we obtain an unbiased estimator of \( f^\wedge_{i,j}(G) \) as follows:

\[ \hat{f}^\wedge_{i,j}(G) = \frac{f^\wedge_{i,j}(G) - 1}{2} - \frac{2 - 2q_L(1 - q_L)}{1 - 2q_L} \]
Table 2: Performance guarantees of one-round 4-cycle counting algorithms providing edge DP.

| Algorithm   | Model  | Variance          | Bias  | Time   |
|-------------|--------|-------------------|-------|--------|
| WShuffle\(\epsilon\) | shuffle | \(O(n^6d_{max}^6 + n^2d_{max}^4)\) | 0     | \(O(n^2)\) |
| WLocal\(\epsilon\) | local    | \(O(n^6 + n^2d_{max}^4)\) | 0     | \(O(n^2)\) |

(line 8). Note that there is a quadratic relationship between \(f_{ij}^4(G)\) and \(f_{ij}^2(G)\), i.e., \(f_{ij}^4(G) = \left(f_{ij}^2(G)\right)^2\). Thus, even though \(f_{ij}^2(G)\) is unbiased, we must subtract a term from \(f_{ij}^2(G)\) (i.e., bias correction) to obtain an unbiased estimator \(\hat{f}_{ij}^4(G)\). This forms the right-hand side of (15) and ensures that \(f_{ij}^4(G)\) is unbiased.

Finally, we sum and scale \(\hat{f}_{ij}^4(i,\sigma(i)+1)(G)\) for each \(i\) to obtain an estimate \(\hat{f}^4(G)\) of the 4-cycle count \(f^4(G)\) in the entire graph \(G\):

\[
\hat{f}^4(G) = \frac{n(n-1)}{4} \sum_{i=1}^{2t} \hat{f}_{ij}^4(i,\sigma(i)+1)(G)
\]

(line 10). Note that it is possible that a single 4-cycle is counted twice; e.g., a 4-cycle \((v_1,v_2,v_3,v_4)\) is possibly counted by \((v_i,v_k)\) and \((v_j,v_l)\) if these user-pairs are selected. However, this is not an issue, because all 4-cycles are equally likely to be counted zero times, once, or twice. We also prove later that \(\hat{f}^4(G)\) in (16) is an unbiased estimate of \(f^4(G)\).

Theoretical Properties.

Theorem 6.1. WShuffle\(\epsilon\) provides \((\epsilon, \delta)-element\) DP and \((2\epsilon, 2\delta)\)-edge DP.

In addition, thanks to the design of (15), we can show that WShuffle\(\epsilon\) produces an unbiased estimate of \(f^4(G)\):

Theorem 6.2. The estimate produced by WShuffle\(\epsilon\) satisfies \(\mathbb{E}[\hat{f}^4(G)] = f^4(G)\).

Finally, we show the MSE (= variance) of \(f^4(G)\):

Theorem 6.3. The estimate produced by WShuffle\(\epsilon\) satisfies

\[
\text{MSE}(\hat{f}^4(G)) = \mathbb{E}[(\hat{f}^4(G) - f^4(G))^2] \\
\leq \frac{9n^5q_1(d_{max} + nq_2)^2}{16(t - 2q_1)^4} + \frac{n^3d_{max}^6}{64t}.
\]

(17)

When \(\epsilon\) and \(\delta\) are constants, \(\epsilon_L = \log n + O(1)\), and \(t = \lceil \frac{n}{\epsilon} \rceil\), we have

\[
\text{MSE}(\hat{f}^4(G)) = \mathbb{E}[(\hat{f}^4(G) - f^4(G))^2] \\
= O\left(n^3d_{max}^6 + n^2d_{max}^4\right).
\]

(18)

We designed experiments to answer these questions.

7.1 Experimental Set-up

We used the following two real graph datasets:

- **Gplus:** The first dataset is the Google+ dataset [49] denoted by Gplus. This dataset includes a social graph \(G = (V,E)\) with \(n = 107614\) users and 12238285 edges, where an edge \((u,v)\) represents that a user \(u\) follows or is followed by \(v\). The average and maximum degrees are \(d_{avg} = 227.4\) and \(d_{max} = 20127\), respectively.

- **IMDB:** The second dataset is the IMDB (Internet Movie Database) [1] denoted by IMDB. This dataset includes a bipartite graph between 896308 actors and 428440 movies. From this, we extracted a graph \(G = (V,E)\) with \(n = 896308\) actors and 57064358 edges, where an edge represents that two actors have played in the same movie. The average and maximum degrees are \(d_{avg} = 127.3\) and \(d_{max} = 15451\), respectively; i.e., IMDB is more sparse than Gplus.

In the full version [34], we also evaluate our algorithms using the Barabási-Albert graphs [10, 29], which have a power-law degree distribution. Moreover, in [34], we evaluate our 4-cycle algorithms using bipartite graphs generated from Gplus and IMDB.

For triangle counting, we evaluated the following four one-round algorithms: WShuffle\(\epsilon\), WShuffle\(\Delta\), WLocal\(\epsilon\), and ARR\(\epsilon\) [33]. We did not evaluate RR\(\Delta\) [32], because it was too inefficient – it was reported in [32] that when \(n = 10^6\), RR\(\Delta\) would require over 30 years even on a supercomputer. The same applies to the one-round local algorithms in [67, 68] with the same time complexity (\(= O(n^3)\)).

indicates that our WShuffle\(\epsilon\) may not work well in an extremely sparse graph where \(d_{max} < n^\frac{1}{2}\). However, \(d_{max} \gg n^\frac{1}{2}\) holds in most social graphs; e.g., the maximum number of friends is much larger than 100 when \(n = 10^8\). In this case, WShuffle\(\epsilon\) can accurately estimate the 4-cycle count, as shown in our experiments.

Comparison with the Central Model. As with triangles, our WShuffle\(\epsilon\) is worse than algorithms in the central model in terms of the estimation error.

Specifically, analogously to the central algorithm for triangles [32], we can consider a central algorithm that outputs \(f^4(G) + \text{Lap}(\frac{d_{max}}{\epsilon})\). This algorithm provides \((\epsilon, 0)\)-edge DP and the variance of \(2d_{max}^4\) is \(O(d_{max}^4)\). Because \(d_{max}\) is much smaller than \(n\), this central algorithm provides a much smaller MSE (= variance) than WShuffle\(\epsilon\). This indicates that there is a trade-off between the trust model and the estimation error.

7 EXPERIMENTAL EVALUATION

Based on the performance guarantees summarized in Tables 1 and 2, we pose the following research questions:

**RQ1.** How much do our entire algorithms (WShuffle\(\epsilon\), WShuffle\(\Delta\), and WShuffle\(\epsilon\), and WLocal\(\epsilon\)) outperform the local algorithms?

**RQ2.** For triangles, how much does our variance reduction technique decrease the relative error?

**RQ3.** How small relative errors do our entire algorithms achieve with a small privacy budget?
For 4-cycle counting, we compared WShuffle\(\_\Delta\) with WLocal\(\_\Delta\). Because WLocal\(\_\Delta\) is the first local 4-cycle counting algorithm (to our knowledge), we did not evaluate other algorithms.

In our shuffle algorithms WShuffle\(\_\Delta\), WShuffle\(\_\Lambda\), and WShuffle\(\_\Delta\), we set \(\delta = 10^{-8}\) \((\ll 1)\) and \(\tau = \frac{1}{2}\). We used the numerical upper bound in [25] for calculating \(\epsilon\) in the shuffle model. In WShuffle\(\_\Lambda\), we set \(\epsilon \in [0.1, 4]\) and divided the total privacy budget \(\epsilon\) as \(\epsilon_1 = \frac{\epsilon}{10}\) and \(\epsilon_2 = \frac{9\epsilon}{10}\). Here, we assigned a small budget to \(\epsilon_1\) because a degree \(d_i\) has a very small sensitivity \((= 1)\) and Laplace \(\frac{1}{\epsilon_2}\) is very small. In ARR\(\_\Delta\), we set the sampling probability \(p_0\) to \(p_0 = n^{-1/3}\) or \(0.1n^{-1/3}\) so that the time complexity is \(O(n^2)\).

We ran each algorithm 20 times and evaluated the average relative error over the 20 runs. In the full version [34], we show that the standard error of the average relative error is small.

### 7.2 Experimental Results

#### Relative Error vs. \(\epsilon\)

We first evaluated the relation between the relative error and \(\epsilon\) in element DP or edge LDP, i.e., \(2\epsilon\) in edge DP. We also measured the time to estimate the triangle/4-cycle count from the adjacency matrix \(A\) using a supercomputer [3] with two Intel Xeon Gold 6148 processors (2.40 GHz, 20 Cores) and 412 GB main memory.

Figure 6 shows the relative error \((\epsilon = 1)\). Here, we show the performance of WShuffle\(\_\Delta\) when we do not add the Laplacian noise (denoted by WShuffle\(\_\Lambda\) (w/o Lap)). In IMDB, we do not show ARR\(\_\Delta\) with \(p_0 = n^{-1/3}\), because it takes too much time (longer than one day). Table 3 highlights the relative error when \(\epsilon = 0.5\) or 0.1. It also shows the running time of counting triangles or 4-cycles when \(\epsilon = 1\) (we verified that the running time had little dependence on \(\epsilon\)).

Figure 6 and Table 3 show that our shuffle algorithms dramatically improve the local algorithms. In triangle counting, WShuffle\(\_\Delta\) outperforms WLocal\(\_\Delta\) by one or two orders of magnitude and ARR\(\_\Delta\), by even more\(^5\). WShuffle\(\_\Delta\) also requires less running time than ARR\(\_\Lambda\) with \(p_0 = n^{-1/3}\). Although the running time of ARR\(\_\Delta\) can be improved by using a smaller \(p_0\), it results in a higher relative error. In 4-cycle counting, WShuffle\(\_\Delta\) significantly outperforms WLocal\(\_\Delta\). The difference between our shuffle algorithms and the local algorithms is larger in IMDB because it is more sparse; i.e., the difference between \(d_{\text{max}}\) and \(n\) is larger in IMDB. This is consistent with our theoretical results in Tables 1 and 2.

Figure 6 and Table 3 also show that WShuffle\(\_\Delta\) outperforms WShuffle\(\_\Lambda\), especially when \(\epsilon\) is small. This is because the variance is large when \(\epsilon\) is small. In addition, WShuffle\(\_\Delta\) significantly outperforms WShuffle\(\_\Lambda\) in IMDB because WShuffle\(\_\Delta\) significantly reduces the variance when \(d_{\text{max}} \ll n\), as shown in Table 1. In other words, this is also consistent with our theoretical results. For example, when \(\epsilon = 0.5\), our variance reduction technique reduces the relative error from 1.41 to 0.488 (about one-third) in IMDB.

Furthermore, Figure 6 shows that the relative error of WShuffle\(\_\Lambda\) is hardly changed by adding the Laplacian noise. This is because the sensitivity of each user’s degree \(d_i\) is very small \((= 1)\). In this case, the Laplacian noise is also very small.

Our WShuffle\(\_\Delta\) achieves a relative error of 0.3 \((\ll 1)\) when the privacy budget is \(\epsilon = 0.5\) or 1 in element DP \((2\epsilon = 1\) or 2 in edge DP). WShuffle\(\_\Delta\) achieve a relative error of 0.15 to 0.3 with a smaller privacy budget \((e.g., \epsilon = 0.2)\) because it does not send local edges – the error of WShuffle\(\_\Delta\) is mainly caused by user-pair sampling that is independent of \(\epsilon\).

In summary, our WShuffle\(\_\Delta\) and WShuffle\(\_\Lambda\) significantly outperform the local algorithms and achieve a relative error much smaller than 1 with a reasonable privacy budget, i.e., \(\epsilon \leq 1\).

---

\(^5\)Note that ARR\(\_\Lambda\) uses only the lower-triangular part of the adjacency matrix \(A\) and therefore provides \(\epsilon\)-edge DP (rather than \(2\epsilon\)-edge DP); i.e., it does not suffer from the doubling issue explained in Section 3.2. However, Figure 6 shows that WShuffle\(\_\Delta\) significantly outperforms ARR\(\_\Lambda\), even if we double \(\epsilon\) for only WShuffle\(\_\Lambda\).
Relative Error vs. $n$. Next, we evaluated the relation between the relative error and $n$. Specifically, we randomly selected $n$ users from all users and extracted a graph with $n$ users. Then we set $\epsilon = 1$ and changed $n$ to various values starting from 2000.

Figure 7 shows the results ($c = 1$). When $n = 2000$, WS$\text{Shuffle}_\Delta$ and WS$\text{Shuffle}_2$ provide relative errors close to WLocal$\Delta$ and WLocal$2$, respectively. This is because the privacy amplification effect is limited when $n$ is small. For example, when $n = 2000$ and $\epsilon = 1$, the numerical bound is $\epsilon L = 1.88$. The value of $\epsilon L$ increases with an increase in $n$; e.g., when $n = 107614$ and 896308, the numerical bound is $\epsilon L = 5.86$ and 7.98, respectively. This explains that our shuffle algorithms significantly outperform the local algorithms when $n$ is large in Figure 7.

Parameter $c$ in WS$\text{Shuffle}_\Delta$. Finally, we evaluated our WS$\text{Shuffle}_\Delta$ while changing the parameter $c$ that controls the bias and variance. Recall that as $c$ increases, the bias is increased, and the variance is reduced. We set $\epsilon = 0.1$ or 1 and changed $c$ from 0.1 to 4.

Figure 8 shows the results. Here, we also show the relative error of WS$\text{Shuffle}_\Delta$. We observe that the optimal $c$ is different for $\epsilon = 0.1$ and $\epsilon = 1$. The optimal $c$ is around 3 to 4 for $\epsilon = 0.1$, whereas the optimal $c$ is around 0.5 to 1 for $\epsilon = 1$. This is because the variance of WS$\text{Shuffle}_\Delta$ is large (resp. small) when $\epsilon$ is small (resp. large). For a small $\epsilon$, a large $c$ is effective in significantly reducing the variance. For a large $\epsilon$, a small $c$ is effective in keeping a small bias.

We also observe that WS$\text{Shuffle}_2$ is always better than (or almost the same as) WS$\text{Shuffle}_\Delta$ when $c = 1$ or 2. This is because most users’ degrees are smaller than the average degree $d_{avg}$, as described in Section 5.4. When $c = 1$ or 2, most user-pairs are ignored. Therefore, we can significantly reduce the variance at the cost of a small bias.

Summary. In summary, our answers to the three questions at the beginning of Section 7 are as follows. RQ1: Our WS$\text{Shuffle}_\Delta$ and WS$\text{Shuffle}_2$ outperform the one-round local algorithms by one or two orders of magnitude (or even more). RQ2: Our variance reduction technique significantly reduces the relative error (e.g., by about one-third) for a small $\epsilon$ in a sparse dataset. RQ3: WS$\text{Shuffle}_\Delta$ achieves a relative error of 0.3 ($\leq 1$) when $\epsilon = 0.5$ or 1 in element...

Figure 7: Relative error vs. $n$ ($\epsilon = 1, c = 1$).

Figure 8: Relative error vs. parameter $c$ in WS$\text{Shuffle}_\Delta$ ($n = 107614$ in Gplus, $n = 896308$ in IMDB).

DP (2$\epsilon = 1$ or 2 in edge DP). WS$\text{Shuffle}_2$ achieves a relative error of 0.15 to 0.3 with a smaller privacy budget: $\epsilon = 0.2$.

8 CONCLUSION

In this paper, we made the first attempt (to our knowledge) to shuffle graph data for privacy amplification. We proposed wedge shuffling as a basic technique and then applied it to one-round triangle and 4-cycle counting with several additional techniques. We showed upper bounds on the MSE for each algorithm. We also showed through comprehensive experiments that our one-round shuffle algorithms significantly outperform the one-round local algorithms and achieve a small relative error with a reasonable privacy budget, e.g., smaller than 1 in edge DP.

For future work, we would like to apply wedge shuffling to other subgraphs such as 3-hop paths [62] and $k$-triangles [38].

ACKNOWLEDGMENTS

Kamalika Chaudhuri and Jacob Imola would like to thank ONR under N00014-20-1-2334 and UC Lab Fees under LFR 18-548554 for research support. Takao Murakami was supported in part by JSPS KAKENHI JP22H05521 and JP19H01109.

REFERENCES

[1] 2005. 12th Annual Graph Drawing Contest. http://mozart.diei.unipg.it/gdcontest/contest2005/index.html.
[2] 2012. What to Do When Your Facebook Profile is Maxed Out on Friends: https://authoritypublishing.com/social-media/what-to-do-when-your-facebook-profile-is-maxed-out-on-friends/.
[3] 2020. AI Bridging Cloud Infrastructure (ABC1). https://abci.ai/.
[4] 2021. The diaspora* Project. https://diasporafoundation.org/.
[5] 2021. Mastodon: Giving Social Networking back to You. https://joinmastodon.org/.
[6] 2022. Tools: Triangle4CycleShuffle. https://github.com/Triangle4CycleShuffle/Triangle4CycleShuffle.
[7] Maryam Ahakbarpour, Amartya Shanku Biwaa, Themistoklis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. 2018. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. Algorithmica 80, 2 (2018), 668–697.
[8] Josh Alman and Virginia Vassilevska Williams. 2021. A Refined Laser Method and Faster Matrix Multiplication. In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA’21), 522–539.
[9] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. 2019. The Privacy Blanket of the Shuffle Model. In Proceedings of the 39th Annual Cryptology Conference on Advances in Cryptology (CRYPTO’19). 638–667.
[61] Seyed-Vahid Sanei-Mehri, Yu Zhang, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2019. FLEET: Butterfly Estimation from a Bipartite Graph Stream. In Proceedings of the 28th ACM International Conference on Information & Knowledge Management (CIKM’19). 1201–1210.

[62] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qui, Hu (Wendy) Wang, and Ting Yu. 2019. Analyzing Subgraph Statistics from Extended Local Views with Decentralized Differential Privacy. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS’19). 703–717.

[63] Tianhao Wang, Bolin Ding, Min Xu, Zhiqong Huang, Cheng Hong, Jingren Zhou, Ninghui Li, and Somesh Jha. 2020. Improving Utility and Security of the Shuffler-based Differential Privacy. Proceedings of the VLDB Endowment 13, 13 (2020), 3545–3558.

[64] Stanley L. Warner. 1965. Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias. J. Amer. Statist. Assoc. 60, 309 (1965), 63–69.

[65] Bin Wu, Ke Yi, and Zhenguang Li. 2016. Counting Triangles in Large Graphs by Random Sampling. IEEE Transactions on Knowledge and Data Engineering 28, 8 (2016), 2013–2020.

[66] Xiaokui Xiao, Gabriel Bender, Michael Hay, and Johannes Gehrke. 2011. iReduct: Differential Privacy with Reduced Relative Errors. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD’11). 229–240.

[67] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2020. Towards Locally Differential Private Generic Graph Metric Estimation. In Proceedings of the IEEE 36th International Conference on Data Engineering (ICDE’20). 1922–1925.

[68] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2021. LF-GDPR: A Framework for Estimating Graph Metrics with Local Differential Privacy. IEEE Transactions on Knowledge and Data Engineering (Early Access) (2021), 1–16. https://doi.org/10.1109/TKDE.2020.3047124