Privacy-preserving Triangle Counting in Directed Graphs

Ziyao Wei[1], Qing Liu[1,2], Zhikun Zhang[1,2], Shouling Ji[1], Yunjun Gao[1,2]
[1]*College of Computer Science and Technology, Zhejiang University,*
[2]*Zhejiang Key Laboratory of Big Data Intelligent Computing*
{*wei_zy, qingliucs, zhikun, sji, gaoyj*}*@zju.edu.cn*

***Abstract*—In directed graphs, the relationship between users is asymmetric, resulting in two types of triangles: *cycle triangles* and *flow triangles*. This paper studies the problem of privacy-preserving triangle counting in directed graphs. Based on different applications, we consider two scenarios, i.e., trusted and untrusted servers. In the literature, privacy-preserving triangle counting in undirected graphs has been widely studied. However, directly applying these algorithms to address our problem suffers from many issues. Concretely, for the trusted server scenario, the differentially private triangle counting algorithms, designed for undirected graphs, exhibit suboptimal performance when applied to directed graphs. Hence, we propose a new centralized differentially private algorithm that adds Laplacian noise to the exact numbers by analyzing global sensitivity. Furthermore, for the untrusted server scenario, the existing techniques cannot be used to count cycle and flow triangles with differential privacy because the local view of each user in directed graphs is limited to out-neighbors rather than all neighbors. Therefore, we design a novel locally differentially private algorithm to provide local unbiased estimation, which implies that after aggregating all the local estimations on the central server side, an unbiased estimation for the numbers of cycle and flow triangles is deduced. Empirical experiments on six real-world graph datasets demonstrate that our proposed algorithms achieve high efficiency and utility.**

***Index Terms*—triangle counting, differential privacy, directed graph**

I. Introduction

Directed graphs can effectively model asymmetric relationships between entities. For instance, in a social network, if user u follows user v but v does not follow u, a directed edge (u, v) is present, while the edge (v, u) is absent. Real-world directed graphs encompass a variety of domains, including social networks [1]–[8], e-commerce networks [9], [10], protein-protein interactions [11], [12], and very large scale integration (VLSI) circuits [13].

A triangle is the smallest cohesive unit in graphs, and counting triangles in a given graph is a crucial task in graph mining and analysis [14]–[18]. [19]–[25] explore privacy-preserving triangle counting in undirected graphs, which is to release the numbers of triangles in the given graph under differential privacy with high utility. In directed graphs, triangles, which are different from those in undirected graphs, include two distinct types, *cycle triangles* and *flow triangles*, as illustrated in Fig. 1. Given a directed graph G, triangle counting aims to count the number of cycle and flow triangles in G, which supports numerous applications, such as community search [1], [2] and clustering coefficient computation [6], [7]. However,

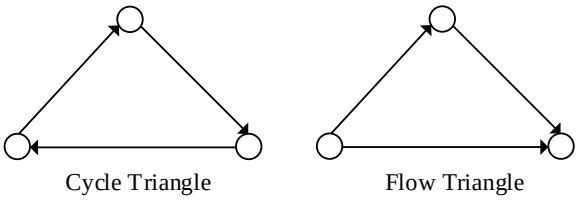


Fig. 1: Triangle Types

disclosing the exact number of cycle and flow triangles in directed graphs might leak users' *private relationships*. For example, consider a social network shown in Fig. 2, consisting of 11 users. Assume that each user's follow list is private and everyone is only aware of their own followees, which are referred to as *out-neighbors* in directed graphs. Let G_1 and G_2 are two social networks differing by only one edge (v_2, v_{11}). Assume that the attacker already knew that the user v_2's following list contains $\{v_1, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$. But, he is not sure whether v_2 follows v_{11} and wants to infer this relationship. According to Fig. 2, G_1 and G_2 represent whether v_2 follows v_{11}. Since G_1 and G_2 have different flow triangles, the attacker can use the triangle information to infer the relationship (v_2, v_{11}). Therefore, it is necessary to protect the triangles' information in directed graphs. Motivated by this, for the first time, we study the problem of privacy-preserving triangle counting in directed graphs.

Edge Differential Privacy: edge-CDP & edge-LDP. To preserve the edge/relationship information in graphs, *edge differential privacy* (edge-DP) has been widely adopted [22], [26], [27]. The general idea of edge-DP is to guarantee that the impact of a *single edge* on the final output of a randomized graph analysis algorithm is limited. In this paper, we consider two scenarios when using edge-DP for triangle counting in directed graphs.

- The central server is *trusted* and has access to the whole graph. Its objective is to publish the results of triangle counting to untrusted third parties for research purposes. For example, Facebook possesses the follower/followee information of all its users and permits third-party researchers to obtain the number of triangles in its social network, since the researchers may need to measure the cohesiveness of the social network via computing the clustering coefficient. In such cases, the central server can use *edge centralized differential privacy* (edge-CDP)

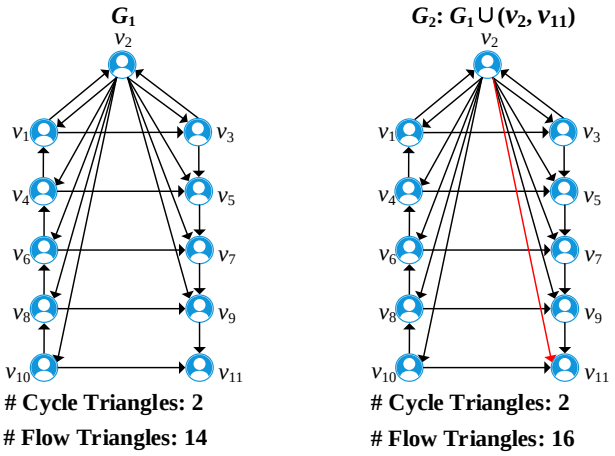


Fig. 2: Directed Triangle Counting

to perturb the results of triangle counting to preserve the edge/relationship information of its users when answering third-party counting requirement. The edge-CDP allows the central server to collect neighbor lists and provides the third-party with the noisy result. Then, the third-party cannot determine whether a specific edge exists.

- The central server is *untrusted* and does not have access to the whole graph. The objective is to collect the triangle counting information from individual users. For example, AT&T has a vast number of users, yet it does not know users' local contacts. The cohesiveness of the telephone network can also be measured via the clustering coefficient. Since the clustering coefficient computation requires the central server to determine the number of triangles within its users' networks, it can permit individual users to apply *edge local differential privacy* (edge-LDP) to obfuscate their private contact information before uploading it to the server. Upon receiving the obfuscated data, the central server can then employ aggregation algorithms to derive the triangle counting results. The edge-LDP ensures the neighbor list collected by the central server is protected. Then, the central server cannot determine whether a specific edge exists.

Centralized Solutions for Trusted Servers. In the literature, the triangle counting problem under edge-CDP for undirected graphs has been well studied [19]–[22]. A straightforward method for counting triangles in directed graphs is to adapt the triangle counting algorithms for undirected graphs, thereby separately counting cycle and flow triangles. In doing so, the total privacy budget should be split into two parts: one for the counting of cycle triangles, and the other for the counting of flow triangles. However, it is challenging to choose an appropriate allocation for these two privacy budgets to optimize utility and can result in a waste of the privacy budget. To address this issue, we propose a new centralized approach that counts the cycle and flow triangles simultaneously. Specifically, we first employ a graph projection technique [28] to obtain a *projected-directed graph* for the given directed graph. Then, we count the numbers of cycle and flow triangles within this projected-directed graph. Finally, we adjust the counting results using *Laplacian noise*, whose parameter is determined by the global sensitivity and the total privacy budget. In our proposed approach, there is no need to split the privacy budget, thereby avoiding the waste of the privacy budget. The primary challenge of this approach is *how to analyze the global sensitivity of the counting function*. To facilitate this analysis, we construct a *case-directed graph*, based on which we can deduce the largest difference in the total number of cycle and flow triangles resulting from the addition or removal of a specific edge.

Local Solutions for Untrusted Servers. Existing studies [23]–[25] have proposed numerous algorithms for triangle counting under edge-LDP in undirected graphs. However, in undirected graphs, a vertex is aware of all its neighbors, whereas in directed graphs, a vertex may be only aware of its *out-neighbors*. For example, on platforms such as YouTube [29], WhatsApp [30], and Weibo [31], through some settings, users may only view their followees, but not followers. This difference in local view renders existing algorithms unsuitable for our problem. This is because the triangle counting algorithms of undirected graphs cannot calculate the local estimation, which is computed based on the local view of each user and reported to the central server. Therefore, we design a new local algorithm to count cycle and flow triangles for untrusted servers. Our proposed algorithm consists of three steps. First, we generate a *noisy graph* by employing a randomized response technique [32]. Second, each user downloads the noisy graph from the central server and computes the local estimation of the numbers of cycle and flow triangles by injecting *Laplacian noise*. Then, each user reports their local estimation of the numbers of cycle and flow triangles back to the central server. Finally, the central server aggregates the local estimations of all users. For the local algorithm, the most important issue is *how each user calculates a useful local estimation of the numbers of cycle and flow triangles*. To tackle this issue, we design a novel *local unbiased estimation*, which is derived from the counts of local subgraphs. Utilizing the local unbiased estimation, the central server can obtain an unbiased estimate of the numbers of cycle and flow triangles in the given directed graph.

In summary, we make the following contributions:

- To the best of our knowledge, this is the first work to address the problem of triangle counting in directed graphs with differential privacy.
- We propose a centralized algorithm with differential privacy guarantees. To validate its utility from a theoretical standpoint, we prove that the algorithm's output is an unbiased estimate and provide an analysis of the output's variance.
- We develop a locally differentially privacy algorithm. We prove that this algorithm also produces an unbiased estimate and present an analysis of the upper bound of the output's variance.
- We conduct comprehensive experiments on six real-world

datasets, which showcase the high efficiency and utility of our proposed approaches.

Roadmap. We review the related works in Section II. Section III introduces the preliminaries. Section IV proposes the centralized differentially private releasing algorithms for triangle counting in directed graphs. Section V proposes the locally differentially private releasing algorithms for triangle counting in directed graphs. Experiments are presented in Secton VI. Finally, we conclude this paper in Section VII.

II. Related Work

Triangle Counting with Differential Privacy. By now, many efforts have been devoted to the study of triangle counting with differential privacy [33]. The existing algorithms can be divided into centralized and non-centralized differential privacy.

For the centralized differential privacy, the straightforward approach for triangle counting is to add the Laplacian noise to the counting function [19]. To reduce the sensitivity, Nissim et al. [20] and Blocki et al. [21] propose the smooth sensitivity and restricted sensitivity, respectively. Avoiding the Laplacian mechanism, Zhang et al. [22] propose a ladder framework and apply their framework for subgraph counting problems including triangle counting. With a relaxed version of edge differential privacy, Rastogi et al. [34] propose a general algorithm for releasing the count of specified subgraphs including triangles. Karwa et al. [27] further improve the algorithm in [34], and the proposed algorithm satisfies a stronger notation of privacy. For the exponential random graph, Lu and Miklau [35] propose algorithms for estimating the alternating k-triangle [36]. Under node differential privacy, Kasiviswanathan et al. [28] and Ding et al. [37], [38] propose triangle counting algorithms based on linear programming and a projection method, respectively. Besides, Chen and Zhou [39] present a solution to subgraph counting for any subgraphs including triangles, which could satisfy either edge differential privacy or node differential privacy.

For the non-centralized differential privacy, Imola et al. [24] present two algorithms with local differential privacy for triangle counting, one is non-interactive, and the other is interactive. Then, following this work, Imola et al. [25] address the drawback of the interactive triangle counting algorithm of high communication cost. Theoretically, Eden et al. [23] prove the lower bounds of the additive errors of triangle counting algorithms with local differential privacy, including both non-interactive and interactive. Sun et al. [40] propose the decentralized differential privacy and design a framework that can calculate the numbers of subgraphs including triangles. Liu et al. [41] propose the edge relationship differential privacy and a two-phase framework for triangle counting. Imola et al. [42] propose a wedge shuffling technique and apply it to triangle counting. Liu et al. [43] propose a crypto-assisted differentially private triangle counting system, named CARGO.

Note that existing techniques are proposed for undirected graphs. In this paper, we consider triangle counting with differential privacy in directed graphs.

Triangle-based Directed Graphs Analysis. The cycle and flow triangles are widely used for directed graph analysis. Takaguchi and Yoshida [44] employ cycle triangles and flow triangles to design cycle truss and flow truss, respectively, which can be used to discover subgraphs with different structures. Then, Liu et al. [1] consider cycle and flow triangles simultaneously and propose D-truss, which requires that each edge should be contained in k_c cycle triangles and k_f flow triangles. D-truss can be effectively used for community search. Fagiolo [6] propose a directed clustering coefficient, which includes flow (transitive) clustering coefficient and cycle (cyclic) clustering coefficient, to cluster directed graphs. Trolliet et al. [7] extend the directed clustering coefficients to design an interest clustering coefficient to measure the clustering of directed social graphs with interest links. Moreover, Parente and Colosimo [45] employ cycle and flow triangles to estimate the influence of the network structure on dynamical processes, which is used to model the multiplex brain network. However, these works ignore the privacy issue, which is the focus of this paper.

III. Preliminaries

In this section, we introduce the key concepts of edge-CDP and edge-LDP for directed graphs. We consider a directed graph $G = (V, E)$, where V and E denote the sets of edges and vertices, respectively. Let $|V| = n$ and $|E| = m$. Let $\lambda \in \{0, 1\}^n$ be an n-dimensional vector. We employ λ_i to denote the out-neighbor list of a vertex $v_i \in V$. If the j-th dimension of λ_i is 1, it means that vertex v_j is an out-neighbor of v_i. For example, in Fig. 2, since v_2 and v_3 are out-neighbors of v_1, $\lambda_1 = (0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0)^\top$. The out-degree of v, denoted by $\deg^+_G(v)$, is the number of v's out-neighbors.

A. ϵ-edge-CDP over Directed Graphs

Centralized differential privacy assumes that the central server is trustworthy and aims to protect user privacy by perturbing the output. It guarantees that for any two neighboring datasets differing by a single data record, the outputs of the algorithm have an indistinguishable distribution. Before formally defining ϵ-edge-CDP over directed graphs, we give the definition of *neighboring directed graphs*.

Definition 1 (Neighboring Directed Graphs)**.** *Given two directed graphs $G = (V, E)$ and $G' = (V, E')$ with the same vertex set V, G and G' are neighbors if (1) $|E - E'| = 0$ and $|E' - E| = 1$, or (2) $|E - E'| = 1$ and $|E' - E| = 0$.*

In other words, the neighboring directed graphs are two directed graphs that share the same vertex set but differ by one edge. Based on this, we formally define *ϵ-edge centralized differential privacy* (ϵ-edge-CDP) over directed graphs.

Definition 2 (ϵ-edge-CDP over Directed Graphs)**.** *Given a randomized algorithm $\mathcal{A} : \mathcal{G} \to \mathcal{R}$, $\mathcal{G}$ and $\mathcal{R}$ denote the input and output domains of $\mathcal{A}$, respectively. $\mathcal{A}$ satisfies ϵ-edge-CDP,*

if for any pair of neighboring directed graphs, $G, G^{'} \in \mathcal{G}$, and any subset of possible outputs $\mathcal{O} \subseteq \mathcal{R}$, it holds that

$$\Pr[\mathcal{A}(G) \in \mathcal{O}] \leq e^{\epsilon}\Pr[\mathcal{A}(G^{'}) \in \mathcal{O}].$$

In other words, for any pair of neighboring directed graphs G and $G^{'}$, the ϵ-edge-CDP requires that $\mathcal{A}$ produces similar output distributions. The similarity of $\mathcal{A}(G)$ and $\mathcal{A}(G^{'})$ can be adjusted by the privacy budget ϵ. The smaller ϵ is, the more similar $\mathcal{A}(G)$ and $\mathcal{A}(G^{'})$ are, thereby providing stronger privacy protection. Specially, when $\epsilon = 0$, the distribution of $\mathcal{A}(G)$ and $\mathcal{A}(G^{'})$ are same, thus the attacker could not infer the information from the outputs.

B. ϵ-edge-LDP over Directed Graphs

Local differential privacy assumes that the central server is untrustworthy. It allows the data owners to perturb their private data locally before uploading the perturbed data to the server. Unlike edge-CDP, which is defined on two neighboring directed graphs, edge-LDP over directed graphs is based on the *neighboring out-neighbor lists*.

Definition 3 (Neighboring Out-neighbor Lists)**.** *For a directed graph $G = (V, E)$ and a vertex $v_i \in V$, let $\lambda_i \in \{0,1\}^n$ be the list of v_i's out-neighbors. Given another list $\lambda_i' \in \{0,1\}^n$, λ_i and λ_i' are considered neighboring out-neighbor lists if λ_i and $\lambda_i^{'}$ differ in one element.*

Two out-neighbor lists are neighboring if they differ by one bit. Based on Definition 3, we formally define *ϵ-edge local differential privacy* (ϵ-edge-LDP) over directed graphs as follows.

Definition 4 (ϵ-edge-LDP)**.** *A randomized algorithm $\mathcal{A} : \{0,1\}^n \to \mathcal{R}$ satisfies ϵ-edge-LDP if for any pair of neighboring out-neighbor lists $\lambda_i, \lambda_i^{'} \in \{0,1\}^n$, and any subset of possible outputs $\mathcal{O} \subseteq \mathcal{R}$, it holds that*

$$\Pr[\mathcal{A}(\lambda_i) \in \mathcal{O}] \leq e^{\epsilon}\Pr[\mathcal{A}(\lambda_i^{'}) \in \mathcal{O}].$$

C. Implementation Mechanism and Properties of Differential Privacy

Laplacian Mechanism. It is the most commonly used mechanism to implement differential privacy. Given a data analysis function f, the Laplacian mechanism adds Laplacian noise to f to achieve differential privacy. The scale of the Laplacian noise is determined by the global sensitivity GS_f of the function f and the privacy budget ϵ.

Definition 5 (Global Sensitivity [19])**.** *Given a function $f : \mathcal{D} \to \mathbb{R}^d$, the global sensitivity of f is defined as,*

$$GS_f = \max_{D,D' \in \mathcal{D}: D \sim D'} \|f(D) - f(D^{'})\|_1,$$

where $D \sim D^{'}$ denotes that D and $D^{'}$ are two neighboring datasets.

Theorem 1 (Laplacian Mechanism [46])**.** *Given a function $f : \mathcal{D} \to \mathbb{R}^d$, the Laplacian mechanism is defined as: $M_L(D, f, \epsilon) = f(D) + (X_1, X_2, \cdots, X_d)^{\top}$, where X_i i.i.d. follows a Laplacian distribution $\mathrm{Lap}(\frac{GS_f}{\epsilon})$. For a parameter b, the Laplacian distribution has the density function $\mathrm{Lap}(b)(x) = \frac{1}{2b}\exp(-\frac{|x|}{b})$. The Laplacian mechanism preserves ϵ-differential privacy.*

Randomized Response. The randomized response [32] is used to implement local differential privacy. Qin et al. [47] prove that applying randomized responses to neighboring lists of each user provides ϵ-edge-LDP.

Theorem 2. *If a randomized algorithm $\mathcal{A}$ outputs a noisy neighbor list $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)^{\top} \in \{0,1\}^n$ such that $\forall j \in [n]$, $\alpha_j \neq \lambda_{i,j}$ with probability $p_1 = \frac{1}{e^{\epsilon}+1}$, then $\mathcal{A}$ provides ϵ-edge-LDP.*

It is easy to verify that Theorem 2 also holds for the out-neighbor list of a vertex in directed graphs.

Post-Processing and Composition. Differential privacy has several nice properties. The post-processing theorem ensures that the output can be processed while still maintaining differential privacy. The sequential (resp. parallel) composition theorem ensures that the sequential (resp. parallel) composition of differential privacy algorithms also satisfies differential privacy.

Property 1 (Post-Processing Theorem [46])**.** *Let $\mathcal{A}$ be an algorithm and f be an arbitrary randomized mapping. If $\mathcal{A}$ satisfies ϵ-differential privacy, $f \circ \mathcal{A}$ also satisfies ϵ-differential privacy.*

Property 2 (Sequential Composition Theorem [25])**.** *Let $\mathcal{A}_1$ be an algorithm satisfying ϵ_1-differential privacy; M be the output of $\mathcal{A}_1$; $\mathcal{A}_2(M)$ be an algorithm satisfying ϵ_2-differential privacy. Then, the sequential composition $(\mathcal{A}_1, \mathcal{A}_2(M))$ satisfies $(\epsilon_1 + \epsilon_2)$-differential privacy.*

Property 3 (Parallel Composition Theorem [46])**.** *Let $\mathcal{A}_1$ be an algorithm satisfying ϵ_1-differential privacy; $\mathcal{A}_2$ be an algorithm satisfying ϵ_2-differential privacy. Then, the parallel composition $(\mathcal{A}_1, \mathcal{A}_2)$ satisfies $(\epsilon_1 + \epsilon_2)$-differential privacy.*

D. Problem Definition

We formally define the centralized and locally differentially private triangle counting problem. Notice that, in centralized scenario, we consider that the input domain $\tilde{\mathcal{G}}$ only contains the bounded graphs, where for each $G \in \tilde{\mathcal{G}}$, the maximum degree of vertices in G does not exceed given projection degree $\tilde{d}_{max}$.

Problem 1. *Given a directed graph $G = (V, E)$, where all vertices are public and all edges are private, and a privacy budget ϵ, we aim to release the numbers of cycle triangles $f_{c\triangle}(G)$ and flow triangles $f_{f\triangle}(G)$ in G while satisfying ϵ-edge-CDP.*

Problem 2. *Given a directed graph $G = (V, E)$ represented as out-neighbor lists λ_1, λ_2, $\cdots$, λ_n, where user $v_i (i = 1, 2, \cdots, n)$ holds λ_i, and a privacy budget ϵ, we aim to release the numbers of cycle triangles $f_{c\triangle}(G)$ and flow triangles $f_{f\triangle}(G)$ in G while satisfying ϵ-edge-LDP.*

Algorithm 1: Centralized Algorithm

Input : Directed graph $G = (V, E)$; projection degree $\tilde{d}_{max} \in \mathbb{Z}_{\geq 0}$; privacy budget $\epsilon \in \mathbb{R}_{\geq 0}$; the number of all the vertices n

Output: The released numbers of cycle triangles $\hat{f}_{c\triangle}(G, \epsilon)$ and flow triangles $\hat{f}_{f\triangle}(G, \epsilon)$ in G

1 $\tilde{G} \leftarrow$Graph Projection(G, $\tilde{d}_{max}$);

2 $f_{\triangle}(\tilde{G}) \leftarrow$ Triangle Counting($\tilde{G}$);

3 Compute global sensitivity $GS_{f\triangle} \leftarrow n + 3\tilde{d}_{max} - 4$;

4 Add the noise $\hat{f}_{\triangle}(\tilde{G}, \epsilon) \leftarrow f_{\triangle}(\tilde{G}) + (\mathrm{Lap}(\frac{GS_{f_{\triangle}}}{\epsilon}), \mathrm{Lap}(\frac{GS_{f_{\triangle}}}{\epsilon}))^{\top}$;

5 **return** $(\hat{f}_{c\triangle}(G, \epsilon), \hat{f}_{f\triangle}(G, \epsilon))^{\top}$;

IV. Directed Triangle Counting with Edge-CDP

In this section, we propose centralized algorithms to address Problem 1. Note that due to the space limitation, some proofs can be found in our technical report [48].

A. Strawman Solution

The basic approach to addressing Problem 1 is to employ the algorithms for undirected graphs to count cycle and flow triangles separately. To this end, we first employ the graph projection on input-directed graph. Then, we count the numbers of cycle and flow triangles in the projected-directed graph. Next, we should divide the total privacy budget ϵ into two parts: ϵ_1 and ϵ_2. The noises $\mathrm{Lap}(\frac{GS_{f_{c\triangle}}}{\epsilon_1})$ and $\mathrm{Lap}(\frac{GS_{f_{f\triangle}}}{\epsilon_2})$ are added to the counts of cycle and flow triangles, respectively, like that used for undirected graphs. Releasing those counts satisfy ϵ_1-edge-CDP and ϵ_2-edge-CDP, respectively. Hence, after the graph projection, the basic approach satisfies ϵ-edge-CDP according to Property 3.

However, the basic approach suffers from the following limitation. For a fixed privacy budget ϵ, it is not easy to determine an optimal privacy budget allocation ratio for ϵ_1 and ϵ_2. The utility of the basic approach depends on this privacy budget allocation ratio, and if it is not properly set, the approach may fail to achieve optimal utility, resulting in the underuse of the privacy budget ϵ.

B. Centralized Solution Overview

We design a centralized differentially private releasing algorithm based on the principle of improving utility. To address the limitation of the strawman solution, which may lead to wasting the privacy budget, the algorithm releases the counts of cycle and flow triangles simultaneously. We analyze the global sensitivity of the counting function to implement ϵ-edge-CDP. As shown in Fig. 3, the centralized differentially private releasing algorithm contains three phases: (1) graph projection; (2) triangle counting; and (3) counting perturbation.

Phase 1: Graph Projection. Graph projection aims to reduce the maximum out-degree of vertices in the input-directed graph. After graph projection, the out-degree of each vertex in the projected-directed graph would not exceed $\tilde{d}_{max}$, where $\tilde{d}_{max}$ is given projection degree. The graph projection technique could significantly reduce the global sensitivity of the counting function [24], [28], which is shown in Section IV-C. In the graph projection phase, for each vertex v in the input-directed graph G, we first randomly shuffle the integers of $\{1, 2, 3, \cdots, \deg^{+}_{G}(v)\}$, then we traverse the out-neighbor list of v. For the i-th vertex in the out-neighbor list, if the i-th number is larger than $\tilde{d}_{max}$, then the i-th vertex will be removed from the out-neighbor list.

Phase 2: Triangle Counting. In the triangle counting phase, we compute the exact numbers of cycle and flow triangles in the projected-directed graph $\tilde{G}$ returned from Phase 1.

Phase 3: Counting Perturbation. In the counting perturbation phase, we implement ϵ-edge-CDP through Laplacian mechanism. We add i.i.d. Laplacian noise to the counting results returned from Phase 2. The parameter of the Laplacian noise is $\frac{GS_{f_{\triangle}}}{\epsilon}$. Here, $GS_{f_{\triangle}}$ is the global sensitivity of counting function $f_{\triangle}$ and ϵ is the privacy budget. By adding the noise, the algorithm satisfies ϵ-edge-CDP. We refer the readers to Section IV-C for more details of Phase 3.

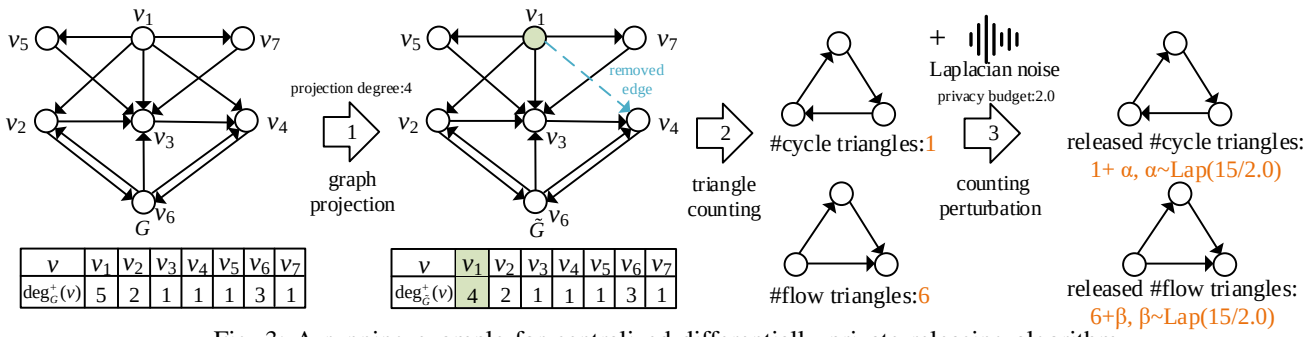
As shown in Algorithm 1, the Centralized Algorithm first runs graph projection and triangle counting (Lines 1, 2). Then the algorithm computes the global sensitivity $GS_{f_{\triangle}}$ of the counting function $f_{\triangle}$ (Line 3). The $GS_{f_{\triangle}}$ is $n + 3\tilde{d}_{max} - 4$, where n is the number of all vertices in the input-directed graph (i.e. the same as the projected-directed graph) and $\tilde{d}_{max}$ is the projection degree. Then the algorithm adds the i.i.d. Laplacian noise to the exact numbers of cycle triangles $f_{c\triangle}(\tilde{G})$ and flow triangles $f_{f\triangle}(\tilde{G})$ in $\tilde{G}$ (Line 4). Finally, the algorithm returns the released numbers of cycle triangles $\hat{f}_{c\triangle}(G, \epsilon)$ and flow triangles $\hat{f}_{f\triangle}(G, \epsilon)$ in G (Line 5).

Example 1. *In the example of Fig. 3, the central server can access the whole graph G. The directed graph G contains 7 nodes $v_1, v_2, v_3, v_4, v_5, v_6, v_7$ and the out-degrees of these vertices in order are $5, 2, 1, 1, 1, 3, 1$. Firstly, the graph projection process runs on G. The projection degree is set to 4. Hence, after the graph projection, the out-neighbors of v_1 is set to $\{v_2, v_3, v_5, v_7\}$. The edge (v_1, v_4) is eliminated. Then the algorithm finds the numbers of cycle and flow triangles in the projected-directed graph $\tilde{G}$, those are 1 and 6. Then the counting perturbation process adds the Laplacian noise to the exact numbers. The parameter of the Laplacian noise is related to the global sensitivity of the counting function and the privacy budget. Especially, after the projection, the global sensitivity of the counting function is $n+3\tilde{d}_{max}-4$, that is 15 and the privacy budget is 2.0, thus the parameter of Laplacian noise is 7.5. After the perturbation, the center server releases the numbers of cycle and flow triangles in G, which are $1+\alpha$ and $6+\beta$. Here, α and β i.i.d. follow $\mathrm{Lap}(7.5)$.*

C. Sensitivity Analysis

Given the counting function $f_{\triangle} : \tilde{\mathcal{G}} \rightarrow \mathbb{Z}^2_{\geq 0}$, we analyze the global sensitivity of $f_{\triangle}$ according to Definition 5. Theorem 3 analyzes the global sensitivity of $f_{\triangle}$ with graph projection.

Theorem 3. *Given the counting function $f_{\triangle} : \tilde{\mathcal{G}} \rightarrow \mathbb{Z}^2_{\geq 0}$, where $\tilde{\mathcal{G}} = \{G : \max\{\deg^{+}_{G}(v) : v \in V_G\} \leq \tilde{d}_{max}, |V_G| = n\}$ (i.e., V_G denotes the vertex set of G), the global sensitivity of $f_{\triangle}$ is $n + 3\tilde{d}_{max} - 4$.*



v	v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\deg_G^+(v)$	5	2	1	1	1	3	1

v	v_1	v_2	v_3	v_4	v_5	v_6	v_7
$\deg_{\tilde{G}}^+(v)$	4	2	1	1	1	3	1

Fig. 3: A running example for centralized differentially private releasing algorithm

For the directed graphs without graph projection, the global sensitivity of $f_\triangle$ is $4(n-2)$. It is because, given a directed edge (u, v) and a vertex w, (u, v) and w can form at most 4 triangles. With the help of the graph projection, the global sensitivity of $f_\triangle$ can be reduced to $n+3\tilde{d}_{max}-4$, where $\tilde{d}_{max}$ is usually much smaller than n. Thus, the global sensitivity of $f_\triangle$ can be reduced by $3(n-\tilde{d}_{max})-4$.

D. Algorithm Analysis

Privacy Analysis. We discuss the privacy of the centralized differentially private releasing algorithm.

Theorem 4. *For any input privacy budget ϵ, after the graph projection, the centralized differentially private releasing algorithm satisfies ϵ-edge-CDP.*

Utility Analysis. Firstly, we prove that the output is the unbiased estimation for the exact numbers of cycle and flow triangles. Next, we analyze the variance of the output. Notice that the output is the unbiased estimation, thus the variance of the output could reflect the magnitude of the error between the output and the exact result according to bias-variance decomposition [49].

Theorem 5. *Let $\hat{f}_\triangle(G,\epsilon) = (\hat{f}_{c\triangle}(G,\epsilon), \hat{f}_{f\triangle}(G,\epsilon))^\top$ be the output of centralized differentially private releasing algorithm. Then, for all $\epsilon \in \mathbb{R}_{\geq 0}$, $\tilde{d}_{max} \in \mathbb{Z}_{\geq 0}$, and G such that any vertex in G has at most $\tilde{d}_{max}$ out-neighbors,*

$$\mathbb{E}[(\hat{f}_{c\triangle}(G,\epsilon), \hat{f}_{f\triangle}(G,\epsilon))] = (f_{c\triangle}(G), f_{f\triangle}(G)),$$

$$\mathbb{V}[\hat{f}_{c\triangle}(G,\epsilon)] = \frac{2}{\epsilon^2}(n^2 + 6\tilde{d}_{max}n - 8n + 9\tilde{d}_{max}^2 - 24\tilde{d}_{max} + 16),$$

$$\mathbb{V}[\hat{f}_{f\triangle}(G,\epsilon)] = \frac{2}{\epsilon^2}(n^2 + 6\tilde{d}_{max}n - 8n + 9\tilde{d}_{max}^2 - 24\tilde{d}_{max} + 16).$$

Complexity Analysis. Let n be the number of all vertices, d_{max} is the largest number of out-neighbors of each vertex and $\tilde{d}_{max}$ is the projection degree. The centralized differentially private releasing algorithm contains three phases. The first phase is graph projection and takes $O(nd_{max})$ time. The second phase is triangle counting and takes $O(n\tilde{d}_{max}^2)$ time. The third phase is counting perturbation and takes $O(1)$ time. The overall complexity is $O(n\tilde{d}_{max}^2 + nd_{max})$.

V. Directed Triangle Counting with Edge-LDP

In this section, we propose local algorithms to address Problem 2.

A. Strawman Solution

The straightforward approach for Problem 2 is a two-phase method. The algorithm first runs the randomized response on each user side. Each user reports the perturbed out-neighbor list to the central server. The central server aggregates all the neighbor lists to a noisy graph, and then counts the numbers of cycle and flow triangles in the noisy graph. According to Theorem 2 and Property 1, this approach satisfies ϵ-edge-LDP for each user.

The straightforward approach faces two main limitations: (1) The approach will consume huge running time, that is because the noisy graph is usually denser than the input graph. Thus, for each user, the number of out-neighbors of each user in the noisy graph is more than the input graph and the triangle counting requires traversing out-neighbors of each user repeatedly; (2) The approach cannot achieve a satisfactory utility, since the noisy graph is usually denser than the input-directed graph.

B. Local Solution Overview

We design a locally differentially private releasing algorithm. The total privacy budget ϵ is divided into two parts ϵ_1, ϵ_2 for two phase in the algorithm, respectively. To address the challenge that the local view of each user is too limited, we generate a noisy graph through employing randomized response on each user side. This process satisfies ϵ_1-edge-LDP for each user. Each user can download the noisy graph from the central server. The local estimation is generated by each user and reported to the central server. To make the second phase satisfy the ϵ_2-edge-LDP for each user, we analyze the global sensitivity of the local estimation. Compared with the strawman solution, this method reduces the number of times to traverse the out-neighbors of each user in the noisy graph. In addition, this method can provide an unbiased estimation of the numbers of cycle and flow triangles in the input-directed graph, which implies that this method reaches higher utility than the strawman solution. As shown in Fig. 4, the locally differentially private releasing algorithm contains two phases: (1) noisy graph generation; (2) local estimation.

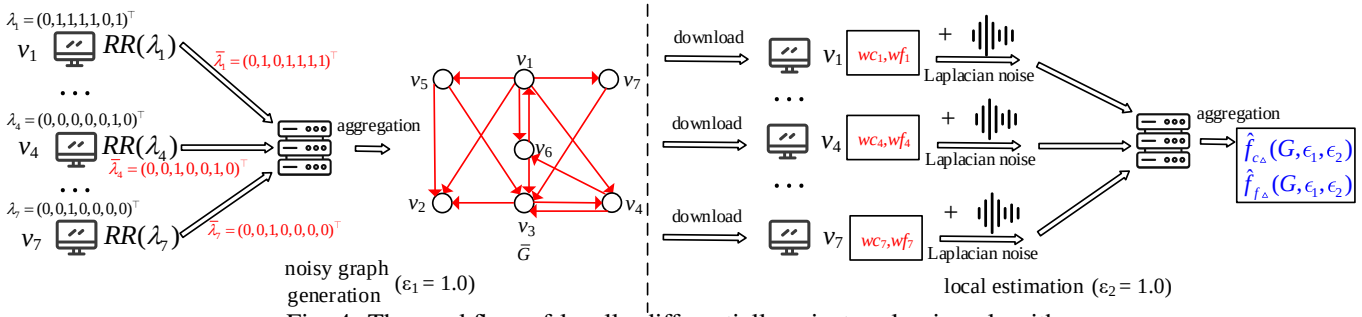


Fig. 4: The workflow of locally differentially private releasing algorithm

Phase 1: Noisy Graph Generation. Noisy graph could bring more information about the whole input-directed graph to each user. In the noisy graph generation phase, for each user v_i in the input-directed graph G, we process the randomized response for v_i's out-neighbor list λ_i. Then each user reports the noisy out-neighbor list $\overline{\lambda}_i$ to the central server. The central server aggregates all the noisy out-neighbor lists to generate the noisy graph $\overline{G}$. By using randomized response, the noisy graph generation phase satisfies ϵ_1-edge-LDP for each user. The details of Phase 1 are in Section V-C.

Phase 2: Local Estimation. Each user should report useful local estimation to the central server, then the central server can aggregate all the local estimations to the estimated numbers of cycle and flow triangles. In the local estimation phase, for each user v_i in the input-directed graph G, we run the out-neighbor list projection to set some element of λ_i from 1 to 0, such that the number of v_i's out-neighbors does not exceed the projection degree $\tilde{d}_{max}$. We design a local unbiased estimator and the estimator first needs to obtain the cardinalities of 6 sets, respectively. Then the estimator generates the local estimation $(wc_i, wf_i)^\top$ through these numbers. To generate these sets, v_i should download the noisy graph from the central server, which is generated from Phase 1. To satisfy ϵ_2-edge-LDP for this phase, the Laplacian mechanism is used. The parameter of the Laplacian noise is $\frac{GS_{(wc_i,wf_i)^\top}}{\epsilon_2}$. Here, $GS_{(wc_i,wf_i)^\top}$ is the global sensitivity of the local estimation $(wc_i, wf_i)^\top$ and ϵ_2 is the privacy budget allocated in this phase. By adding the noise, this phase satisfies ϵ_2-edge-LDP for each user. The details of Phase 2 are in Section V-D.

As Fig. 4 shows, the central server can only view all the vertices in the whole graph G, and edges in G cannot be viewed. For any user v_i $(i = 1, 2, \cdots, 7)$, we generate noisy out-neighbor list $\overline{\lambda}_i$ via the randomized response. Then, each user reports $\overline{\lambda}_i$ to the central server. The central server aggregates $\overline{\lambda}_1, \overline{\lambda}_2, \cdots, \overline{\lambda}_7$ to a noisy graph $\overline{G}$. Next, each user v_i downloads the noisy graph from the central server. Through the noisy graph $\overline{G}$ and projected local view $\tilde{\lambda}_i$, the local estimation $(wc_i, wf_i)^\top$ is generated. After adding the Laplacian noise, the noisy local estimation is sent to the central server. Finally, the central server aggregates all the local estimations to obtain the estimated numbers of cycle triangles $\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2)$ and flow triangles $\hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2)$ in G.

Algorithm 2: Noisy Graph Generation

Input : Directed graph $G = (V, E)$ represented as out-neighbor lists $\lambda_1, \lambda_2, \cdots, \lambda_n$; privacy budget for Phase 1 $\epsilon_1 \in \mathbb{R}_{\geq 0}$
Output: Noisy graph $\overline{G} = (V, \overline{E})$

```
// User
1  p_1 ← 1/(e^{ε_1}+1);
2  for i = 1 to n do
3      λ̄_{i,i} ← 0;
4      for j = 1 to n, j ≠ i do
5          b ← Bernoulli(p_1);
6          λ̄_{i,j} ← λ_{i,j} ⊕ b;
7      Release λ̄_i to the central server;
// Server
8  Ḡ ← aggregate λ̄_i, i ∈ [n];
9  return Ḡ;
```

C. Noisy Graph Generation

Due to the structure of directed graphs, for each user v_i, v_i can only view the out-neighbors. However, triangles, unlike stars, v_i could not ensure that he is in either cycle triangles or flow triangles. In particular, each user in cycle triangles can only view one other user. For flow triangles, one user can not view two other users, another user can only view one other user, and the remaining user can view two other users. Although there is one user who can view the other two users, he can not be sure that there is an edge between the other two users. Thus, for each user v_i, to provide the useful local estimation to the central server, more information about the whole input-directed graph needs to be obtained. Based on this observation, we run the randomized response process on each user's side, allowing each user to release the perturbed out-neighbor list to the central server. On the central server side, the server aggregates all the out-neighbor lists to generate a noisy graph, then each user can download the noisy graph from the central server.

As shown in Algorithm 2, the Noisy Graph Generation first runs on each user side. The algorithm initializes the parameter p_1 used in the randomized response to $\frac{1}{e^{\epsilon_1}+1}$ (Line 1). Then on each user v_i side, the algorithm traverses the v_i's out-neighbor list. For each element $\lambda_{i,j}$ in λ_i, the algorithm generates a random number b following the Bernoulli distribution of parameter p_1. Then the algorithm generates the element $\overline{\lambda}_{i,j}$ by $\lambda_{i,j} \oplus b$. Notice that the self-loop of v_i is always set to inexistence, since the numbers of cycle and flow triangles will

not be affected by self-loops. After the generation of the noisy out-neighbor list $\overline{\lambda}_i$, v_i reports $\overline{\lambda}_i$ to the central server. (Lines 2-7) On the central server side, the algorithm aggregates all the out-neighbor lists to generate a noisy graph $\overline{G}$ (Line 8).

Example 2. *As Fig. 4 shows, each user v_i ($i = 1, 2, \cdots, 7$) holds the out-neighbor list λ_i. In this case, the total privacy budget is 2.0. We allocate the same amount of privacy budget to each phase, that is, the privacy budget for each phase is 1.0. For example, v_1's out-neighbor list $\lambda_1 = (0, 1, 1, 1, 1, 0, 1)^\top$. The parameter p_1 of the randomized response is set to $\frac{1}{e^{\epsilon_1}+1} = \frac{1}{e+1}$. When using randomized response on v_1's side, the noisy out-neighbor list $\overline{\lambda}_1 = (0, 1, 0, 1, 1, 1, 1)^\top$ is generated. Then v_1 reports the $\overline{\lambda}_1$ to the central server. Notice that $\overline{\lambda}_2, \overline{\lambda}_3, \cdots, \overline{\lambda}_7$ are also generated and reported to the central server. The central server aggregates all the λ_i ($i = 1, 2, \cdots, 7$) to a noisy graph $\overline{G}$. We can see that in $\overline{G}$, $\{v_2, v_4, v_5, v_6, v_7\}$ is all the out-neighbor lists of v_1, which is corresponding to $\overline{\lambda}_1$.*

D. Local Estimation

Similar to the centralized differentially private releasing algorithm, for each user v_i, before the estimating process, we use out-neighbor list projection to remove some out-neighbors of v_i. This aims to reduce the global sensitivity of the local estimation, leading to the reduction of the variance of the algorithmic output. Besides, the projection could also reduce the times of out-neighbor lists traversal, then reducing the running time. To design the local estimator, we form the local triangles, local 2-stars, and local single edges from the projected local view $\tilde{\lambda}_i$ and the noisy graph $\overline{G} = (V, \overline{E})$. The method is replacing edges invisible to v_i with edges in the noisy graph.

We consider local subgraphs,

$$T_i = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = 1, (v_j, v_k) \in \overline{E}, (v_k, v_i) \in \overline{E}\}, \quad (1)$$

$$T_i^{(1)} = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = 1, (v_j, v_k) \in \overline{E}, i \neq k\}, \quad (2)$$

$$T_i^{(2)} = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = 1, (v_k, v_i) \in \overline{E}, j \neq k\}, \quad (3)$$

$$L_i = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = 1, i \neq k, j \neq k\}. \quad (4)$$

T_i represents the set of local cycle triangles containing v_i; $T_i^{(1)}$ represents the set of local cycle 2-stars starting from v_i; $T_i^{(2)}$ represents the set of local cycle 2-stars whose middle vertex is v_i; L_i represents the set of local single edges starting from v_i.

Similarly, we consider local subgraphs,

$$R_i = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = \tilde{\lambda}_{i,k} = 1, (v_j, v_k) \in \overline{E}\}, \quad (5)$$

$$S_i = \{(v_i, v_j, v_k) : \tilde{\lambda}_{i,j} = \tilde{\lambda}_{i,k} = 1, j \neq k\}. \quad (6)$$

R_i represents the set of local flow triangles containing v_i; S_i represents the set of local flow 2-stars whose central vertex is v_i.

Let t_i, $t_i^{(1)}$, $t_i^{(2)}$, l_i, r_i, and s_i be the cardinalities of T_i, $T_i^{(1)}$, $T_i^{(2)}$, L_i, R_i, and S_i, respectively. We design the local unbiased estimation $(wc_i, wf_i)^\top = (t_i - p_1 t_i^{(1)} - p_1 t_i^{(2)} + p_1^2 l_i, r_i - p_1 s_i)^\top$. Next, we show the unbiasedness of the estimation after the aggregation of the central server.

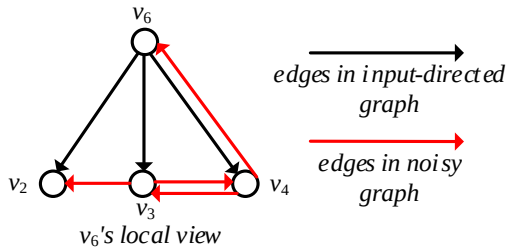


Fig. 5: Local view of v_6

Theorem 6. *Let $(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^n wc_i, \frac{1}{1-2p_1}\sum_{i=1}^n wf_i)^\top$ be the estimation of the numbers of cycle and flow triangles in input-directed graph G (s.t. the maximum out-degree of vertices in G is not exceed the projection degree $\tilde{d}_{max}$), such that for any user v_i, $wc_i = t_i - p_1 t_i^{(1)} - p_1 t_i^{(2)} + p_1^2 l_i$ and $wf_i = r_i - p_1 s_i$, then*

$$\mathbb{E}[(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^n wc_i, \frac{1}{1-2p_1}\sum_{i=1}^n wf_i)] = (f_{c\triangle}(G), f_{f\triangle}(G)).$$

i.e., $(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^n wc_i, \frac{1}{1-2p_1}\sum_{i=1}^n wf_i)^\top$ is the unbiased estimation for the numbers of cycle and flow triangles in input-directed graph G.

Theorem 7. *For each user v_i, given the local estimation $(wc_i, wf_i)^\top$, the global sensitivity of $(wc_i, wf_i)^\top$ is $2(n-2) + 2\tilde{d}_{max}$.*

As shown in Algorithm 3, the Local Estimation first runs on each user side. On each user v_i side, the algorithm has 2 steps, the first is out-neighbor list projection, and the second is using local unbiased estimator to generate the local estimation. In Step 1, the algorithm first initializes $\tilde{\lambda}_i$ to λ_i (Line 3). Then the algorithm generates a random permutation $\mathcal{P}$ of integers ranged from 1 to $\deg_G^+(v)$ (Line 4). Next, for any element $\tilde{\lambda}_{i,j}$ in $\tilde{\lambda}_i$, if $\tilde{\lambda}_{i,j} = 1$, that is $(v_i, v_j) \in E$, then we determine whether the corresponding integer $\mathcal{P}[j]$ in $\mathcal{P}$ is larger than $\tilde{d}_{max}$, and if so, set the $\tilde{\lambda}_{i,j}$ to 0 (Lines 5-7). In Step 2, the algorithm first computes the cardinalities of 6 sets. By using these numbers, the algorithm generates a local unbiased estimation $(wc_i, wf_i)^\top$. (Lines 9-11) Then, the algorithm computes the global sensitivity $GS_{(wc_i, wf_i)^\top}$ of $(wc_i, wf_i)^\top$, and adds the i.i.d. Laplacian noise to $(wc_i, wf_i)^\top$ whose parameter is determined by $GS_{(wc_i, wf_i)^\top}$ and the privacy budget for Phase 2 ϵ_2 (Lines 12, 13). After adding the noise, the algorithm releases $(\hat{wc}_i, \hat{wf}_i)^\top$ to the central server (Line 14). On the server side, the algorithm aggregates all the estimations reported by each user and obtains the estimation of the numbers of cycle triangles $\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2)$ and flow triangles $\hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2)$ in input-directed graph G (Line 15).

Example 3. *On the user side, we take v_6 as an example. As Fig. 5 shown, v_6 can view 3 exact edges (v_6, v_2), (v_6, v_3), and (v_6, v_4). Besides, v_6 also knows that (v_3, v_2), (v_3, v_4), (v_4, v_3), and (v_4, v_6) are in noisy graph. Then, we count the local*

Algorithm 3: Local Estimation

Input : Directed graph $G = (V, E)$ represented as out-neighbor lists $\lambda_1, \lambda_2, \cdots, \lambda_n$; noisy graph $\overline{G} = (V, \overline{E})$; privacy budget for Phase 2 $\epsilon_2 \in \mathbb{R}_{\geq 0}$; projection degree $\tilde{d}_{max} \in \mathbb{Z}_{\geq 0}$; the parameter p_1

Output: The released numbers of cycle triangles $\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2)$ and flow triangles $\hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2)$ in G

// User

for $i = 1$ *to* n **do**

  Step 1: Out-neighbor list projection;

  Initialize $\tilde{\lambda}_i \leftarrow \lambda_i$;

  $\mathcal{P} \leftarrow$ a random permutation of $\{1, 2, 3, \cdots, \deg^+_G(v)\}$;

  for $j = 1$ *to* n **do**

    if $\tilde{\lambda}_{i,j} == 1$ *and* $\mathcal{P}[j] > \tilde{d}_{max}$ **then**

      $\tilde{\lambda}_{i,j} \leftarrow 0$;

  Step 2: Local unbiased estimator;

  $t_i, t_i^{(1)}, t_i^{(2)}, l_i \leftarrow$ the cardinalities of $T_i, T_i^{(1)}, T_i^{(2)}, L_i$;

  $r_i, s_i \leftarrow$ the cardinalities of R_i, S_i;

  $wc_i, wf_i \leftarrow t_i - p_1 t_i^{(1)} - p_1 t_i^{(2)} + p_1^2 l_i, r_i - p_1 s_i$;

  Compute $GS_{(wc_i, wf_i)^\top} \leftarrow 2(n-2) + 2\tilde{d}_{max}$;

  $(\hat{wc}_i, \hat{wf}_i)^\top \leftarrow (wc_i, wf_i)^\top + (\mathrm{Lap}(\frac{GS_{(wc_i,wf_i)^\top}}{\epsilon_2}), \mathrm{Lap}(\frac{GS_{(wc_i,wf_i)^\top}}{\epsilon_2}))^\top$;

  Release $(\hat{wc}_i, \hat{wf}_i)^\top$ to the central server;

// Server

$(\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2))^\top \leftarrow (\frac{1}{3(2p_1-1)^2} \sum_{i=1}^n \hat{wc}_i, \frac{1}{1-2p_1} \sum_{i=1}^n \hat{wf}_i)^\top$;

return $(\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2))^\top$;

subgraphs. Obviously, (v_6, v_3, v_4) is a local cycle triangle containing v_6 (i.e., $(v_6, v_3, v_4) \in T_6$). Besides, (v_6, v_3, v_4) can also form a local cycle 2-star starting from v_6 (i.e., $(v_6, v_3, v_4) \in T_6^{(1)}$), a local cycle 2-star whose middle vertex is v_6 (i.e., $(v_6, v_3, v_4) \in T_6^{(2)}$), and a local single edge (i.e., $(v_6, v_3, v_4) \in L_6$). The counting results of T_6, $T_6^{(1)}$, $T_6^{(2)}$, and L_6 are 1, 3, 3, and 15, respectively. Similarly, we can find the counting results of R_6 and S_6 are 3 and 6, respectively. Then we compute the local estimation $(wc_6, wf_6)^\top$ of v_6. Before reporting the local estimation to the central server, we should add the Laplacian noise to $(wc_6, wf_6)^\top$. On the central server side, we aggregate all the local estimations to obtain the final results, that is, $(\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2))^\top$.

E. Algorithm Analysis

Privacy Analysis. We discuss the privacy of the locally differentially private releasing algorithm.

Theorem 8. *For any input privacy budgets ϵ_1, ϵ_2, the locally differentially private releasing algorithm satisfies $(\epsilon_1 + \epsilon_2)$-edge-LDP for each user.*

Utility Analysis. Firstly, we prove that the output is the unbiased estimation for the exact numbers of cycle and flow triangles. Next, we analyze the variance of the output. Notice that the output is the unbiased estimation, thus the variance of the output could reflect the magnitude of the error between the output and the exact result.

Theorem 9. *We denote the output of locally differentially private releasing algorithm as $\hat{f}_\triangle(G, \epsilon_1, \epsilon_2) = (\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2))^\top$. Then, for all $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$, $\tilde{d}_{max} \in \mathbb{Z}_{\geq 0}$, and G such that any vertex in G has at most $\tilde{d}_{max}$ out-neighbors,*

$$\mathbb{E}[(\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2))] = (f_{c\triangle}(G), f_{f\triangle}(G)).$$

Theorem 10. *We denote the output of locally differentially private releasing algorithm as $\hat{f}_\triangle(G, \epsilon_1, \epsilon_2) = (\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2), \hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2))^\top$. Then, for all $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$, $\tilde{d}_{max} \in \mathbb{Z}_{\geq 0}$, and G such that any vertex in G has at most $\tilde{d}_{max}$ out-neighbors,*

$$\mathbb{V}[\hat{f}_{c\triangle}(G, \epsilon_1, \epsilon_2)] \leq O(\frac{e^{4\epsilon_1}}{(1-e^{\epsilon_1})^4} n^3 (\tilde{d}_{max}^2 + \frac{1}{\epsilon_2^2})),$$
$$\mathbb{V}[\hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2)] \leq O(\frac{e^{\epsilon_1}}{(1-e^{\epsilon_1})^2} n^2 (\tilde{d}_{max}^2 + \frac{e^{\epsilon_1} n}{\epsilon_2^2})).$$

Complexity Analysis. Let n be the number of all vertices, d_{max} is the largest number of out-neighbors of each vertex and $\tilde{d}_{max}$ is the projection degree. We analyze the complexity of each user and the central server, respectively. We also analyze the communication complexity. Each user first runs the randomized response to perturb the neighbor list and then releases it, which consumes $O(n)$ time and $O(n)$ communication time. Then the central server receives the noisy neighbor lists from all the users, and aggregates the noisy graph. This takes $O(n^2)$ time. Then each user receives the noisy graph from the central server, which consumes $O(n^2)$ communication time. For any user, the graph projection takes $O(d_{max})$ time, and obtaining the local estimation takes $O(n\tilde{d}_{max})$ time, then adding the Laplacian noise and releasing the local estimation both take $O(1)$ time. The central server receives the local estimation and aggregates all the estimations, which takes $O(n)$ time. In summary, each client takes $O(n\tilde{d}_{max})$ time and the central server takes $O(n^2)$ time. In these procedures, the communication between the per-user and the central server takes $O(n^2)$ time.

Discussion. In some scenarios, users can be aware of his/her both followees and followers. However, we would like to claim that our proposed algorithm LDP still works in these scenarios. It is because LDP only uses the out-neighbor lists as input. For privacy, LDP would satisfy the edge-LDP for each user due to the following reason. When a user's out-neighbor list is fixed, for two neighboring in-neighbor lists, the distributions of the algorithmic output are the same. For utility, LDP only requires the out-neighbor lists as input and provides unbiased estimations for the numbers of cycle and flow triangles.

VI. Experiments

We conduct experiments on a server with Intel(R) Xeon(R) CPU E5-2650 and 128 GB main memory. All experiments are implemented in C++ on the CentOS operating system.

TABLE I: Statistics of the datasets

Dataset	$\|V\|$	$\|E\|$	$f_{c\triangle}$	$f_{f\triangle}$
Bitcoin OTC (BO)	5.9**K**	35.6**K**	38,581	125,886
Wiki-Vote (WV)	7.1**K**	103.7**K**	43,975	746,557
Math Overflow (MO)	24.8**K**	228**K**	760,663	2,923,310
As-Caida (AC)	26.5**K**	106.8**K**	72,730	218,190
Cit-HepPh (CH)	34.5**K**	421.5**K**	524	1,288,013
P2P-Gnutella (PG)	36.7**K**	88.3**K**	59	1,531

A. Experimental Setup

Datasets. We use six real-world directed graphs in our experiments. All datasets are from SNAP[1]. Table I shows the statistics of these real-world directed graphs.

Parameters and Metrics. The parameters tested in the experiments include privacy budget ϵ, projection degree $\tilde{d}_{max}$, graph size $|V|$, and privacy budget allocation ratio $\frac{\epsilon_1}{\epsilon}$. The default values of these four parameters are 2.0, d_{max} (i.e., the maximum out-degree in a directed graph), $|V|$, and 0.5, respectively.

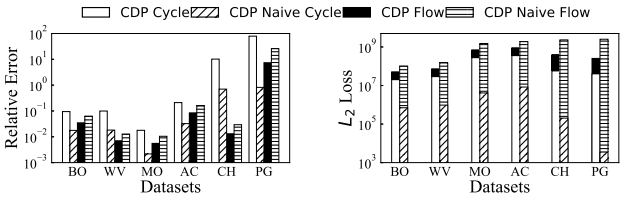
B. Evaluation of Centralized Solutions

In this set of experiments, we evaluate the performance of centralized solutions, including the strawman solution **CDP Naive** mentioned in Section IV-A and our proposed algorithm **CDP**.

CDP VS. CDP Naive. Firstly, we compare CDP and CDP Naive in terms of utility on all datasets. For CDP Naive, the privacy budget allocation ratio for ϵ_1 and ϵ_2 is set to $1:1$. The results are shown in Fig. 6. For the sum of the L_2 loss of the numbers of cycle and flow triangles, CDP is smaller than CDP Naive. The sum of the L_2 loss is calculated by $(f_{c\triangle}-\hat{f}_{c\triangle})^2+(f_{f\triangle}-\hat{f}_{f\triangle})^2$. Therefore, it can reflect the whole error of the algorithmic output. Thus, CDP reaches higher utility than CDP Naive. For the number of cycle triangles (i.e., CDP Cycle and CDP Naive Cycle), the relative error and L_2 loss of CDP are larger than that of CDP Naive. For the number of flow triangles (i.e., CDP Flow and CDP Naive Flow), the relative error and L_2 loss of CDP are smaller than that of CDP Naive. That is because CDP Naive uses an excessive amount of the privacy budget to count cycle triangles. Notice that the relative error of the numbers of cycle and flow triangles are calculated by $\frac{|f_{c\triangle}-\hat{f}_{c\triangle}|}{f_{c\triangle}}$, $\frac{|f_{f\triangle}-\hat{f}_{f\triangle}|}{f_{f\triangle}}$, respectively. Since there may be a big gap between $f_{c\triangle}$ and $f_{f\triangle}$, the sum of relative error cannot reasonably reflect the whole error of the algorithmic output. For PG, the CDP performs large relative error of the number of cycle triangles, since the number of cycle triangles in PG $f_{c\triangle}$ is too small, which leading to that a little noise on $\hat{f}_{c\triangle}$ will bring large relative error.

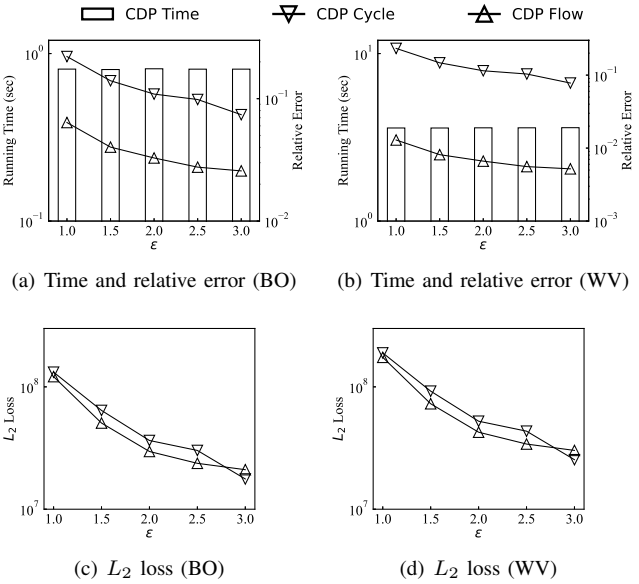
Impact of ϵ. Then, we test the performance of CDP by varying the privacy budget ϵ. The results are shown in Fig. 7. We can observe that with the growth of ϵ, (1) the running time of CDP keeps stable, and (2) both the relative error and L_2 loss of CDP decrease. The reason behind this is that the privacy budget ϵ only affects the addition of the Laplacian noise, and thus the running time of CDP is insensitive to ϵ. In addition, when ϵ increases, the parameter b of the density function of the Laplacian distribution decreases, leading to the reduced noise. Then the relative error and L_2 loss of CDP become smaller.

[1]https://snap.stanford.edu/data/index.html



(a) Relative error (b) L_2 loss

Fig. 6: CDP VS. CDP Naive



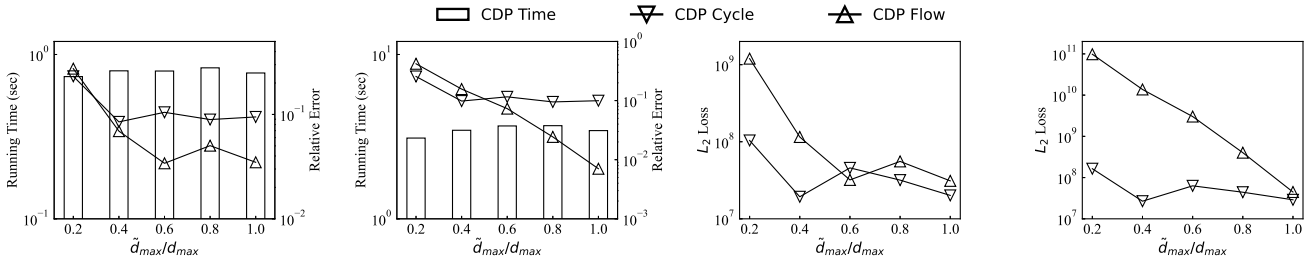
(a) Time and relative error (BO) (b) Time and relative error (WV)

(c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 7: The impact of ϵ on CDP

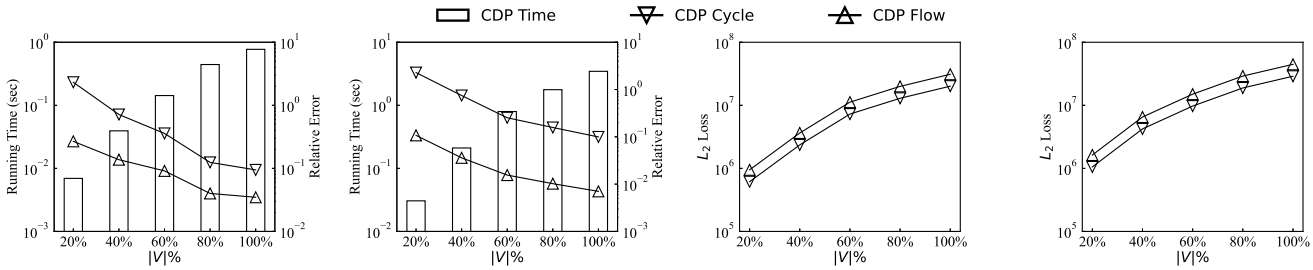
Impact of $\tilde{d}_{max}$. Next, we study the impact of projection degree $\tilde{d}_{max}$ on CDP. Fig. 8 shows the results. When $\tilde{d}_{max}$ increases, the running time fluctuates slightly. This is because at least 95% of the vertices have degrees no larger than $0.2d_{max}$. Hence, only a small number of vertices are affected in the graph projection process. Moreover, for small $\tilde{d}_{max}$, the difference between the projected directed graph and input directed graph is large, leading to inaccurate counting results and large relative error and L_2 loss. The larger $\tilde{d}_{max}$, the smaller difference between the projected directed graph and input directed graph. Hence, the relative error and L_2 loss of CDP decrease.

Impact of $|V|$. We explore the impact of graph size $|V|$ on CDP. To this end, we generate a set of subgraphs by randomly sample a certain percentage of vertices from original graphs, ranging from 20% to 100%. The results are shown in Fig. 9. Obviously, the larger graph, the more running time of CDP. For utility, when the graph becomes larger, the relative error of CDP decreases while the L_2 loss of CDP increases. It is because the larger graph usually contain more cycle and flow triangles. The relative error is calculated by dividing the absolute error (the difference between the counted triangle number and the actual triangle number) by the actual



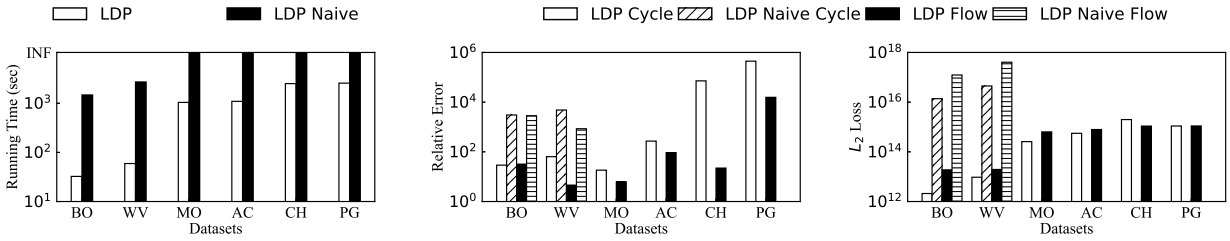
(a) Time and relative error (BO) (b) Time and relative error (WV) (c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 8: The impact of $\tilde{d}_{max}$ on CDP



(a) Time and relative error (BO) (b) Time and relative error (WV) (c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 9: The impact of $|V|$ on CDP



(a) Running time (b) Relative error (c) L_2 loss

Fig. 10: LDP VS. LDP Naive

triangle number. When the actual triangle number increases, the relative error decreases. The L_2 loss is the square of the difference between the counted triangle number and the actual triangle number. When the the actual triangle number grows, the difference becomes large as well, resulting in larger L_2 loss.

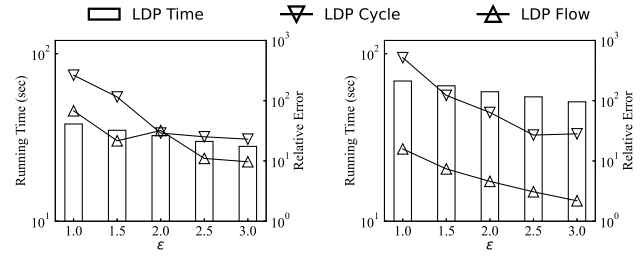
C. Evaluation of Local Solutions

In this subsection, we test the performance of local solutions, including the baseline **LDP Naive** mentioned in Section V-A and our proposed algorithm **LDP**.

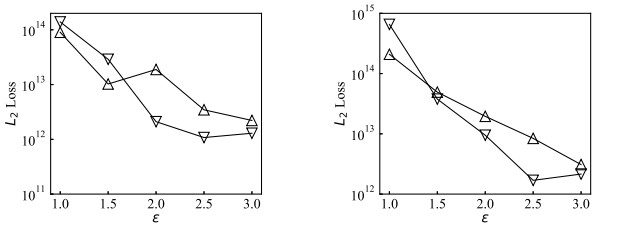
LDP VS. LDP Naive. First of all, we compare the performance of LDP and LDP Naive. For LDP, the privacy budget allocation ratio for ϵ_1 and ϵ_2 is set to $1:1$. The results are shown in Fig. 10. Note that we terminate the algorithm if the running time exceeds 2 hours and we denote it by INF. We can observe that in Fig. 10(a), LDP is at least two orders of magnitude faster than LDP Naive. For the datasets MO, AC, CH, and PG, the LDP Naive can not complete within 3 hours. For the utility shown in Fig. 10(b) and Fig.10(c), both the relative error and L_2 loss of LDP are at least two orders of magnitude better than that of LDP Naive. Overall, LDP significantly outperforms LDP Naive.

Impact of ϵ. Then, we test the impact of privacy budget ϵ on LDP. Fig. 11 shows the results. We can see that the running time, relative error, and L_2 loss of LDP decrease with the increase of ϵ. It is because, for LDP, the parameter p_1 of the randomized response process depends on ϵ. When ϵ increases, p_1 decreases. Since $|E| \ll |V|^2$, the noisy graph usually becomes less dense, requiring less time to count the triangles. Moreover, the greater the value of ϵ, the more similar the noisy graph and original graph are. Hence, the counted numbers of cycle and flow triangles in the noisy graph are closer to that of the original graph, resulting in smaller relative error and L_2 loss.

Impact of $\tilde{d}_{max}$. Next, we evaluate the impact of projection degree $\tilde{d}_{max}$ on LDP. The results are shown in Fig. 12. As the same with CDP, when $\tilde{d}_{max}$ increases, the running time of LDP almost remains stable. That is because a small number of vertices are affected in the graph projection process. Moreover, the relative error and L_2 loss of LDP also do not fluctuate much. The reason behind is that for LDP, each user can view not only the local projected out-neighbor list but also the noisy

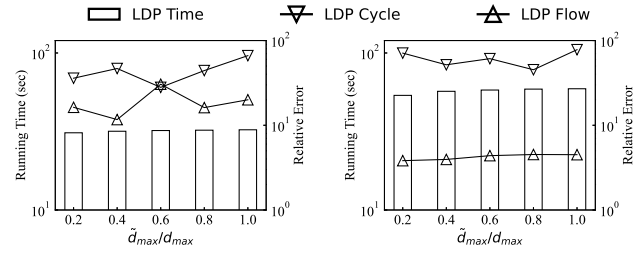


(a) Time and relative error (BO) (b) Time and relative error (WV)

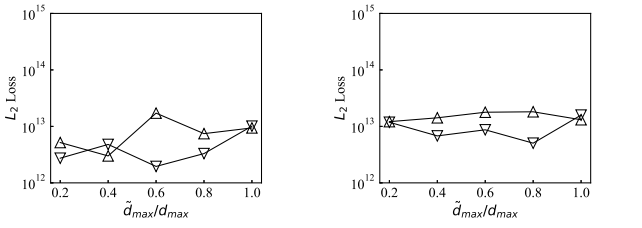


(c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 11: The impact of ϵ on LDP



(a) Time and relative error (BO) (b) Time and relative error (WV)



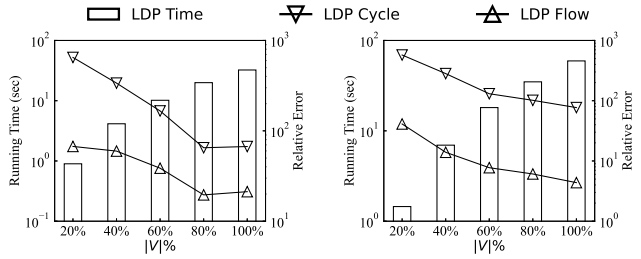
(c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 12: The impact of $\tilde{d}_{max}$ on LDP

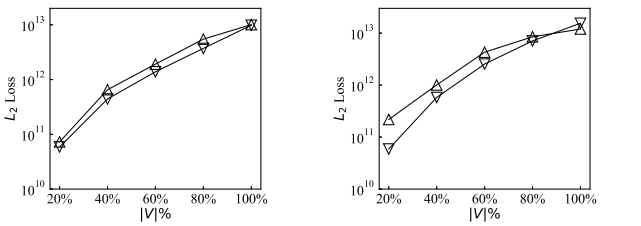
graph. Hence, the relative error and L_2 loss of LDP are not sensitive to $\tilde{d}_{max}$.

Impact of $|V|$. We study the impact of graph size $|V|$ on LDP. The results are shown in Fig. 13. As expected, the running time of LDP increases over larger graphs. For utility, with the growth of the graph size, the relative error of LDP decreases but the L_2 loss of LDP increases. This phenomenon is due to the more cycle and flow triangles in larger graphs.

Impact of $\frac{\epsilon_1}{\epsilon}$. Finally, we explore the influence of $\frac{\epsilon_1}{\epsilon}$ on LDP. The results are shown in Fig. 14. When $\frac{\epsilon_1}{\epsilon}$ increases, the running time of LDP decreases. It is because the more privacy budget used in the first phase, the less dense the noisy graph. Hence, the local estimation phase takes less time. For utility, when $\frac{\epsilon_1}{\epsilon}$ increases, the relative error and L_2 loss of LDP firstly decrease, and then increase. This phenomenon indicates that to achieve good utility, we can allocate the privacy budgets as evenly as possible in each stage.

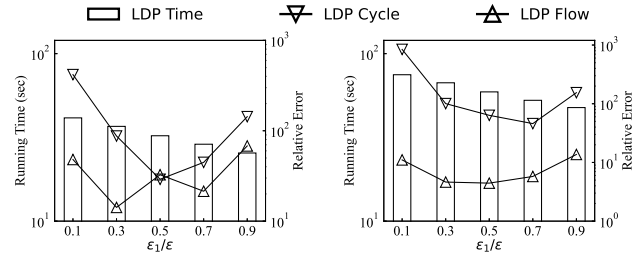


(a) Time and relative error (BO) (b) Time and relative error (WV)

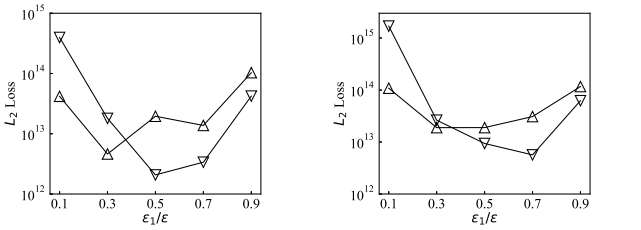


(c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 13: The impact of $|V|$ on LDP



(a) Time and relative error (BO) (b) Time and relative error (WV)



(c) L_2 loss (BO) (d) L_2 loss (WV)

Fig. 14: The impact of $\frac{\epsilon_1}{\epsilon}$ on LDP

VII. Conclusion

In this paper, we have studied the privacy-preserving triangle counting problem in directed graphs. To address this problem, we have proposed two algorithms, with centralized and local differential privacy, respectively. The centralized differentially private releasing algorithm integrates the novel global sensitivity analysis of the counting function. The locally differentially private releasing algorithm employs a novel local unbiased estimation based on the numbers of local subgraphs. Theoretical analysis and empirical evaluations confirm the efficiency and utility of our proposed algorithms.

Acknowledgements

This work was supported in part by the NSFC under Grants No. (62025206, U23A20296, and 62302444), and Zhejiang Province's "Lingyan" R&D Project under Grant No. 2024C01259. Yunjun Gao is the corresponding author of the work.

REFERENCES

[1] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *SIGMOD 2020*. ACM, 2020, pp. 2183–2197.

[2] A. Tian, A. Zhou, Y. Wang, and L. Chen, "Maximal d-truss search in dynamic directed graphs," *Proc. VLDB Endow.*, vol. 16, no. 9, pp. 2199–2211, 2023.

[3] Y. Fang, Z. Wang, R. Cheng, H. Wang, and J. Hu, "Effective and efficient community search over large directed graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2093–2107, 2019.

[4] B. Yang, D. Wen, L. Qin, Y. Zhang, X. Wang, and X. Lin, "Fully dynamic depth-first search in directed graphs," *Proc. VLDB Endow.*, vol. 13, no. 2, pp. 142–154, 2019.

[5] X. Liao, Q. Liu, J. Jiang, X. Huang, J. Xu, and B. Choi, "Distributed d-core decomposition over large directed graphs," *Proc. VLDB Endow.*, vol. 15, no. 8, pp. 1546–1558, 2022.

[6] G. Fagiolo, "Clustering in complex directed networks." *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 76 2 Pt 2, p. 026107, 2006.

[7] T. Trolliet, N. Cohen, F. Giroire, L. Hogie, and S. Pérennes, "Interest clustering coefficient: a new metric for directed networks like twitter," *J. Complex Networks*, vol. 10, no. 1, 2021.

[8] X. Zhou, L. Zhu, W. Li, and Z. Zhang, "A sublinear time algorithm for opinion optimization in directed social networks via edge recommendation," in *KDD 2023*. ACM, 2023, pp. 3593–3602.

[9] Y. Peng, X. Lin, M. Yu, W. Zhang, and L. Qin, "TDB: breaking all hop-constrained cycles in billion-scale directed graphs," in *ICDE 2023*. IEEE, 2023, pp. 137–150.

[10] S. Virinchi and A. Saladi, "BLADE: biased neighborhood sampling based graph neural network for directed graphs," in *WSDM 2023*. ACM, 2023, pp. 42–50.

[11] D. W. McDonald and S. He, "Hyperbolic embedding of attributed and directed networks," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 7003–7015, 2023.

[12] M. R. Hamedani, J. Ryu, and S. Kim, "ELTRA: an embedding method based on learning-to-rank to preserve asymmetric information in directed graphs," in *CIKM 2023*. ACM, 2023, pp. 2116–2125.

[13] Q. Zhang, Y. Du, Z. Su, C. Li, J. Xu, Z. Chen, and Z. Lü, "Threshold-based responsive simulated annealing for directed feedback vertex set problem," in *AAAI 2024*. AAAI Press, 2024, pp. 20 856–20 864.

[14] Q. C. Liu and C. Seshadhri, "Brief announcement: improved massively parallel triangle counting in O(1) rounds," in *PODC 2024*. ACM, 2024, pp. 519–522.

[15] X. Gou and L. Zou, "Sliding window-based approximate triangle counting over streaming graphs with duplicate edges," in *SIGMOD 2021*. ACM, 2021, pp. 645–657.

[16] L. Hu, L. Zou, and Y. Liu, "Accelerating triangle counting on GPU," in *SIGMOD 2021*. ACM, 2021, pp. 736–748.

[17] D. A. Bader, F. Li, A. Ganeshan, A. Gündogdu, J. Lew, O. A. Rodriguez, and Z. Du, "Triangle counting through cover-edges," in *HPEC 2023*. IEEE, 2023, pp. 1–7.

[18] X. Gou and L. Zou, "Sliding window-based approximate triangle counting with bounded memory usage," *VLDB J.*, vol. 32, no. 5, pp. 1087–1110, 2023.

[19] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC 2006*, ser. Lecture Notes in Computer Science, vol. 3876. Springer, 2006, pp. 265–284.

[20] K. Nissim, S. Raskhodnikova, and A. D. Smith, "Smooth sensitivity and sampling in private data analysis," in *STOC 2007*. ACM, 2007, pp. 75–84.

[21] J. Blocki, A. Blum, A. Datta, and O. Sheffet, "Differentially private data analysis of social networks via restricted sensitivity," in *ITCS 2013*. ACM, 2013, pp. 87–96.

[22] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Private release of graph statistics using ladder functions," in *SIGMOD 2015*. ACM, 2015, pp. 731–745.

[23] T. Eden, Q. C. Liu, S. Raskhodnikova, and A. D. Smith, "Triangle counting with local edge differential privacy," in *ICALP 2023*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 52:1–52:21.

[24] J. Imola, T. Murakami, and K. Chaudhuri, "Locally differentially private analysis of graph statistics," in *USENIX Security 2021*. USENIX Association, 2021, pp. 983–1000.

[25] J. Imola, T. Murakami, and K. Chaudhuri, "Communication-efficient triangle counting under local differential privacy," in *USENIX Security 2022*. USENIX Association, 2022, pp. 537–554.

[26] Q. Yuan, Z. Zhang, L. Du, M. Chen, P. Cheng, and M. Sun, "Privgraph: Differentially private graph data publication by exploiting community information," in *USENIX Security 2023*. USENIX Association, 2023, pp. 3241–3258.

[27] V. Karwa, S. Raskhodnikova, A. D. Smith, and G. Yaroslavtsev, "Private analysis of graph structure," *Proc. VLDB Endow.*, vol. 4, no. 11, pp. 1146–1157, 2011.

[28] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. D. Smith, "Analyzing graphs with node differential privacy," in *TCC 2013*, ser. Lecture Notes in Computer Science, vol. 7785. Springer, 2013, pp. 457–476.

[29] [Online]. Available: https://www.youtube.com/

[30] [Online]. Available: https://www.whatsapp.com/

[31] [Online]. Available: https://weibo.com/

[32] S. L. Warner, "Randomized response: a survey technique for eliminating evasive answer bias." *Journal of the American Statistical Association*, vol. 60 309, pp. 63–6, 1965.

[33] C. Dwork, "Differential privacy," in *ICALP 2006*, ser. Lecture Notes in Computer Science, vol. 4052. Springer, 2006, pp. 1–12.

[34] V. Rastogi, M. Hay, G. Miklau, and D. Suciu, "Relationship privacy: output perturbation for queries with joins," in *PODS 2009*. ACM, 2009, pp. 107–116.

[35] W. Lu and G. Miklau, "Exponential random graph estimation under differential privacy," in *KDD 2014*. ACM, 2014, pp. 921–930.

[36] T. A. B. Snijders, P. E. Pattison, G. L. Robins, and M. S. Handcock, "New specifications for exponential random graph models," *Sociological Methodology*, vol. 36, no. 1, pp. 99–153, 2006.

[37] X. Ding, X. Zhang, Z. Bao, and H. Jin, "Privacy-preserving triangle counting in large graphs," in *CIKM 2018*. ACM, 2018, pp. 1283–1292.

[38] X. Ding, S. Sheng, H. Zhou, X. Zhang, Z. Bao, P. Zhou, and H. Jin, "Differentially private triangle counting in large graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5278–5292, 2022.

[39] S. Chen and S. Zhou, "Recursive mechanism: towards node differential privacy and unrestricted joins," in *SIGMOD 2013*. ACM, 2013, pp. 653–664.

[40] H. Sun, X. Xiao, I. Khalil, Y. Yang, Z. Qin, W. H. Wang, and T. Yu, "Analyzing subgraph statistics from extended local views with decentralized differential privacy," in *CCS 2019*. ACM, 2019, pp. 703–717.

[41] Y. Liu, S. Zhao, Y. Liu, D. Zhao, H. Chen, and C. Li, "Collecting triangle counts with edge relationship local differential privacy," in *ICDE 2022*. IEEE, 2022, pp. 2008–2020.

[42] J. Imola, T. Murakami, and K. Chaudhuri, "Differentially private triangle and 4-cycle counting in the shuffle model," in *CCS 2022*. ACM, 2022, pp. 1505–1519.

[43] S. Liu, Y. Cao, T. Murakami, J. Liu, and M. Yoshikawa, "CARGO: crypto-assisted differentially private triangle counting without trusted servers," in *ICDE 2024*. IEEE, 2024, pp. 1671–1684.

[44] T. Takaguchi and Y. Yoshida, "Cycle and flow trusses in directed networks," *CoRR*, vol. abs/1603.03519, 2016. [Online]. Available: http://arxiv.org/abs/1603.03519

[45] F. Parente and A. Colosimo, "Modelling a multiplex brain network by local transfer entropy," *Scientific Reports*, vol. 11, no. 1, p. 15525, 2021.

[46] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3-4, pp. 211–407, 2014.

[47] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren, "Generating synthetic decentralized social graphs with local differential privacy," in *CCS 2017*. ACM, 2017, pp. 425–438.

[48] Z. Wei, Q. Liu, Z. Zhang, S. Ji, and Y. Gao, "Privacy-preserving triangle counting in directed graphs (technical report)." 2024, https://github.com/ZJU-DAILY/PrivTC/blob/main/PrivTC-ICDE2025-TechnicalReport.pdf.

[49] K. P. Murphy, *Machine learning - a probabilistic perspective*, ser. Adaptive computation and machine learning series. MIT Press, 2012.

APPENDIX

A. Proof of Theorem 3

According to the Definition 5, we deduce the global sensitivity $GS_{f_\triangle}$ of $f_\triangle$. The $GS_{f_\triangle}$ can be rewritten as:

$$\begin{aligned} GS_{f_\triangle} &= \max_{G,G^{'}\in\tilde{\mathcal{G}}:G\sim G^{'}} \|f_\triangle(G) - f_\triangle(G^{'})\|_1 \\ &= \max_{G,G^{'}\in\tilde{\mathcal{G}}:G\sim G^{'}} |f_{c\triangle}(G^{'}) - f_{c\triangle}(G)| \\ &\quad + |f_{f\triangle}(G^{'}) - f_{f\triangle}(G)|. \end{aligned}$$

We analyze the special case of two neighboring graphs G and $G^{'}$. Let the graph shown in Fig. 15 be G. Notice that double-direction arrows in Fig. 15 represent the existence of two edges, each with two vertices as a tail to the other vertex. If we delete the edge (v_1, v_2) and let the graph be $G^{'}$, then we analyze the change of the numbers of cycle triangles and flow triangles, respectively. It is easy to find that the number of cycle triangles will be reduced by $\tilde{d}_{max}$. Similarly, we find that the number of flow triangles will be reduced by $n+2\tilde{d}_{max}-4$.

For any directed graph G, if we delete an edge (v_1, v_2) from G, then the change of the number of cycle triangles would never exceed $\tilde{d}_{max}$, because v_2 at most has $\tilde{d}_{max}$ out-neighbors. We also claim that the change of the number of flow triangles would never exceed $n + 2\tilde{d}_{max} - 4$. For any flow triangle that contains (v_1, v_2), we assume the third vertex is v_i, which is different from v_1 and v_2. Let $N_G^+(v_1)$ and $N_G^+(v_2)$ denote the sets of the out-neighbors of v_1 and v_2 in G, respectively. Consider 4 cases for v_i:

Case-1: $v_i \in N_G^+(v_1) \cap N_G^+(v_2)$. In this case, there are at most 3 possible flow triangles contains v_1, v_2, and v_i.

Case-2: $v_i \in N_G^+(v_1)$ and $v_i \notin N_G^+(v_2)$. In this case, there are at most 1 possible flow triangles contains v_1, v_2, and v_i.

Case-3: $v_i \in N_G^+(v_2)$ and $v_i \notin N_G^+(v_1)$. In this case, there are at most 1 possible flow triangles contains v_1, v_2, and v_i.

Case-4: $v_i \notin N_G^+(v_1)$ and $v_i \notin N_G^+(v_2)$. In this case, there are at most 1 possible flow triangles contains v_1, v_2, and v_i.

Due to the constraint of maximum out-degree in directed graph G, the cardinality of $N_G^+(v_1) \cap N_G^+(v_2)$ is at most $\tilde{d}_{max} - 1$. Thus, the number of flow triangles that contain (v_1, v_2) would not exceed $n + 2\tilde{d}_{max} - 4$.

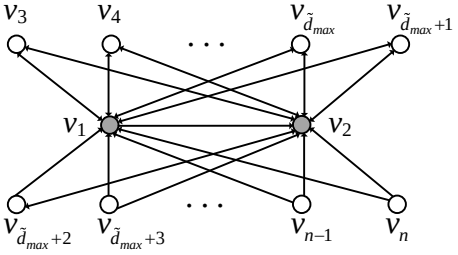


Fig. 15: Global Sensitivity Analysis

B. Proof of Theorem 4

The centralized differential private releasing algorithm first computes the exact numbers of cycle triangles and flow triangles in the projected graph. We denote the counting function as $f_\triangle : \tilde{\mathcal{G}} \to \mathbb{Z}^2_{\geq 0}$. Then, the algorithm adds the Laplacian noise to the function $f_\triangle$ to implement edge-CDP. The parameter of the Laplacian noise is $\frac{n+3\tilde{d}_{max}-4}{\epsilon}$, where $n+3\tilde{d}_{max}-4$ is the global sensitivity of $f_\triangle$ and ϵ is the privacy budget. According to Theorem 1, after the graph projection, the centralized differential private releasing algorithm satisfies the ϵ-edge-CDP.

C. Proof of Theorem 5

The expectation of the Laplacian noise is zero. Thus,

$$\begin{aligned} &\mathbb{E}[(\hat{f}_{c\triangle}(G,\epsilon), \hat{f}_{f\triangle}(G,\epsilon))] \\ =&\mathbb{E}[(f_{c\triangle}(G), f_{f\triangle}(G)) \\ &+ (\mathrm{Lap}(\frac{n+3\tilde{d}_{max}-4}{\epsilon}), \mathrm{Lap}(\frac{n+3\tilde{d}_{max}-4}{\epsilon}))] \\ =&(f_{c\triangle}(G), f_{f\triangle}(G)) + (0,0) \\ =&(f_{c\triangle}(G), f_{f\triangle}(G)). \end{aligned}$$

According to the variance of the random variable which is Laplace distributed,

$$\begin{aligned} &\mathbb{V}[\hat{f}_{c\triangle}(G,\epsilon)] = \mathbb{V}[f_{c\triangle}(G) + \mathrm{Lap}(\frac{n+3\tilde{d}_{max}-4}{\epsilon})] \\ =&\mathbb{V}[\mathrm{Lap}(\frac{n+3\tilde{d}_{max}-4}{\epsilon})] \\ =&\frac{2}{\epsilon^2}(n^2 + 6\tilde{d}_{max}n - 8n + 9\tilde{d}_{max}^2 - 24\tilde{d}_{max} + 16). \end{aligned}$$

Similarly,

$$\begin{aligned} &\mathbb{V}[\hat{f}_{f\triangle}(G,\epsilon)] \\ =&\frac{2}{\epsilon^2}(n^2 + 6\tilde{d}_{max}n - 8n + 9\tilde{d}_{max}^2 - 24\tilde{d}_{max} + 16). \end{aligned}$$

D. Proof of Theorem 6

Let $t_* = \sum_{i=1}^n t_i$, $t_*^{(1)} = \sum_{i=1}^n t_i^{(1)}$, $t_*^{(2)} = \sum_{i=1}^n t_i^{(2)}$, and $l_* = \sum_{i=1}^n l_i$. Let $l_*^{(0)}$ be the number of tuples (v_i, v_j, v_k), such that v_i, v_j, and v_k are the vertices in the input graph and $\lambda_{i,j} = 1$, and $\lambda_{j,k} = \lambda_{k,i} = 0$. Let $l_*^{(1)}$ be the number of tuples (v_i, v_j, v_k), such that $\lambda_{i,j} = \lambda_{j,k} = 1$, and $\lambda_{k,i} = 0$. Let $l_*^{(2)}$ be the number of tuples (v_i, v_j, v_k), such that $\lambda_{i,j} = \lambda_{k,i} = 1$, and $\lambda_{j,k} = 0$. Let $l_*^{\triangle}$ be the number of tuples (v_i, v_j, v_k), such that $\lambda_{i,j} = \lambda_{j,k} = \lambda_{k,i} = 1$. It is easy to show that $l_* = l_*^{(0)} + l_*^{(1)} + l_*^{(2)} + l_*^{\triangle}$ and $l_*^{\triangle} = 3f_{c\triangle}(G)$, where $f_{c\triangle}(G)$ is the exact number of cycle triangles in the input-directed graph G.

Considering a cycle triangle, the triangle is counted $3(1-p_1)^2$ times in the expectation of t_*. Considering a cycle 2-star, the cycle 2-star is counted $p_1(1-p_1)$ times in the expectation of t_*. Considering a single edge, the single edge is counted p_1^2 times in the expectation of t_*, then

$$\mathbb{E}[t_*] = (1-p_1)^2 l_*^{\triangle} + p_1^2 l_*^{(0)} + p_1(1-p_1)(l_*^{(1)} + l_*^{(2)}).$$

Similarly,

$$\begin{aligned} \mathbb{E}[t_*^{(1)}] &= (1-p_1)l_*^{\triangle} + p_1 l_*^{(0)} + (1-p_1)l_*^{(1)} + p_1 l_*^{(2)}, \\ \mathbb{E}[t_*^{(2)}] &= (1-p_1)l_*^{\triangle} + p_1 l_*^{(0)} + (1-p_1)l_*^{(2)} + p_1 l_*^{(1)}. \end{aligned}$$

Then, we obtain,

$$\begin{aligned}\mathbb{E}[\sum_{i=1}^{n} wc_i] &= \mathbb{E}[\sum_{i=1}^{n}(t_i - p_1 t_i^{(1)} - p_1 t_i^{(2)} + p_1^2 l_i)] \\ &= \mathbb{E}[t_* - p_1 t_*^{(1)} - p_1 t_*^{(2)} + p_1^2 l_*] \\ &= \mathbb{E}[t_*] - p_1\mathbb{E}[t_*^{(1)}] - p_1\mathbb{E}[t_*^{(2)}] + p_1^2\mathbb{E}[l_*] \\ &= 3(2p_1-1)^2 f_{c\triangle}(G).\end{aligned}$$

Similarly, let $r_* = \sum_{i=1}^{n} r_i$, $s_* = \sum_{i=1}^{n} s_i$. Let $s_*^{(0)}$ be the number of tuples (v_i, v_j, v_k), such that $\lambda_{i,j} = \lambda_{i,k} = 1$, and $\lambda_{j,k} = 0$. Let $s_*^{\triangle}$ be the number of tuples (v_i, v_j, v_k), such that $\lambda_{i,j} = \lambda_{i,k} = \lambda_{j,k} = 1$. Then it is easy to show that $s_* = s_*^{(0)} + s_*^{\triangle}$ and $s_*^{\triangle} = f_{f\triangle}(G)$, where $f_{f\triangle}(G)$ is the exact number of flow triangles in the input-directed graph G.

Considering a flow triangle, the triangle is counted $(1-p_1)$ times in the expectation of r_*. Considering a flow 2-star, the triangle is counting p_1 times in the expectation of r_*, then

$$\mathbb{E}[r_*] = (1-p_1)s_*^{\triangle} + p_1 s_*^{(0)}.$$

Then, we obtain,

$$\begin{aligned}\mathbb{E}[\sum_{i=1}^{n} wf_i] &= \mathbb{E}[\sum_{i=1}^{n}(r_i - p_1 s_i)] \\ &= \mathbb{E}[r_* - p_1 s_*] \\ &= (1-2p_1) f_{f\triangle}(G).\end{aligned}$$

Thus,

$$\begin{aligned}&\mathbb{E}[(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^{n} wc_i, \frac{1}{1-2p_1}\sum_{i=1}^{n} wf_i)] \\ =&(f_{c\triangle}(G), f_{f\triangle}(G)).\end{aligned}$$

E. Proof of Theorem 7

Considering two out-neighbor lists λ_i and $\lambda_i^{'}$ of v_i differ in one bit, let d_i and $d_i^{'}$ be the number of 1 elements in λ_i and $\lambda_i^{'}$, respectively. Let $\tilde{\lambda}_i$ and $\tilde{\lambda}_i^{'}$ be the projected out-neighbor lists correspond to λ_i and $\lambda_i^{'}$, respectively. Similarly, let t_i, $t_i^{(1)}$, $t_i^{(2)}$, l_i, r_i, s_i, wc_i, wf_i and $t_i^{'}$, $t_i^{(1)'}$, $t_i^{(2)'}$, $l_i^{'}$, $r_i^{'}$, $s_i^{'}$, $wc_i^{'}$, $wf_i^{'}$ correspond to λ_i and $\lambda_i^{'}$, respectively. Consider two cases: (1) $d_i < \tilde{d}_{max}$; (2) $d_i \geq \tilde{d}_{max}$.

Case-1: $d_i < \tilde{d}_{max}$.

Without loss of generality, let $d_i^{'} = d_i + 1$. It is easy to show that $l_i^{'} - l_i = (n-2)$. We claim that $t_i^{'} - t_i \leq l_i^{'} - l_i$, since the t_i is more restrictive on tuples than l_i. Similarly, $t_i^{(1)'} - t_i^{(1)} \leq l_i^{'} - l_i$, $t_i^{(2)'} - t_i^{(2)} \leq l_i^{'} - l_i$.

Then, for the local estimation of the number of cycle triangles,

$$\begin{aligned}&|wc_i^{'} - wc_i| \\ =&|t_i^{'} - t_i - p_1(t_i^{(1)'} - t_i^{(1)}) - p_1(t_i^{(2)'} - t_i^{(2)}) + p_1^2(l_i^{'} - l_i)| \\ \leq&\max\{1+p_1^2, 2p_1\}|l_i^{'} - l_i| \\ =&\max\{1+p_1^2, 2p_1\}(n-2) \\ \leq&2(n-2).(\text{by} \quad 0 < p_1 = \frac{1}{e^{\epsilon_1}+1} \leq \frac{1}{2})\end{aligned}$$

It is easy to show that $s_i^{'} - s_i = 2\binom{d_i+1}{2} - 2\binom{d_i}{2} = 2d_i$. Similarly, we claim that $r_i^{'} - r_i \leq s_i^{'} - s_i$.

Then, for the local estimation of the number of flow triangles,

$$\begin{aligned}|wf_i^{'} - wf_i| &= |r_i^{'} - r_i - p_1(s_i^{'} - s_i)| \\ &\leq \max\{1-p_1, p_1\}|s_i^{'} - s_i| \\ &< 2\tilde{d}_{max}.\end{aligned}$$

Case-2: $d_i \geq \tilde{d}_{max}$.

There are two subcases, namely, **Case-2a** for $d_i^{'} = d_i + 1$ and **Case-2b** for $d_i^{'} = d_i - 1$.

For **Case-2a**, $d_i^{'} = d_i + 1$, then after the graph projection, $\tilde{d}_i^{'} = \tilde{d}_i = \tilde{d}_{max}$. It is easy to show $\tilde{\lambda}_i$ and $\tilde{\lambda}_i^{'}$ differ in 0 or 2 bits. If $\tilde{\lambda}_i$ and $\tilde{\lambda}_i^{'}$ differ in 0 bit, then $|wc_i^{'} - wc_i| = 0$ and $|wf_i^{'} - wf_i| = 0$.

If $\tilde{\lambda}_i$ and $\tilde{\lambda}_i^{'}$ differ in 2 bits, then $|t_i^{'} - t_i| \leq 2(n-2)$, $|t_i^{(1)'} - t_i^{(1)}| \leq 2(n-2)$, $|t_i^{(2)'} - t_i^{(2)}| \leq 2$, $|l_i^{'} - l_i| = 0$. Then, for the local estimation of the number of cycle triangles,

$$\begin{aligned}&|wc_i^{'} - wc_i| \\ =&|t_i^{'} - t_i - p_1(t_i^{(1)'} - t_i^{(1)}) - p_1(t_i^{(2)'} - t_i^{(2)}) + p_1^2(l_i^{'} - l_i)| \\ =&|t_i^{'} - t_i - p_1(t_i^{(1)'} - t_i^{(1)}) - p_1(t_i^{(2)'} - t_i^{(2)})| \\ \leq&\max\{1, 2p_1\} \cdot 2(n-2) \\ \leq&2(n-2).\end{aligned}$$

It is easy to show that $|r_i^{'} - r_i| \leq 2(\tilde{d}_i - 2) + 2 = 2\tilde{d}_i - 2 < 2\tilde{d}_{max}$, $|s_i^{'} - s_i| = 0$. Then, for the local estimation of the number of flow triangles,

$$\begin{aligned}|wf_i^{'} - wf_i| &= |r_i^{'} - r_i - p_1(s_i^{'} - s_i)| \\ &= |r_i^{'} - r_i| \\ &< 2\tilde{d}_{max}.\end{aligned}$$

For **Case-2b**, $d_i^{'} = d_i - 1$. If $d_i > \tilde{d}_{max}$, then after graph projection, $\tilde{d}_i^{'} = \tilde{d}_i = \tilde{d}_{max}$. As the same way in **Case-2a**, we can show that $|wc_i^{'} - wc_i| \leq 2(n-2)$ and $|wf_i^{'} - wf_i| < 2\tilde{d}_{max}$. If $d_i = \tilde{d}_{max}$, then as the same way in **Case-1**, we can show that $|wc_i^{'} - wc_i| \leq 2(n-2)$ and $|wf_i^{'} - wf_i| < 2\tilde{d}_{max}$.

Then, for any user v_i, the global sensitivity of $GS_{(wc_i, wf_i)^\top}$ is $2(n-2) + 2\tilde{d}_{max}$.

F. Proof of Theorem 8

The local differential private releasing algorithm first uses a randomized response to perturb the input-directed graph G into noisy graph $\overline{G}$. According to the Theorem 2, the Noisy Graph Generation satisfies ϵ_1-edge-LDP for each user. Then, each user adds the Laplacian noise to the local estimation. By Theorem 1 and Property 1, Local Estimation satisfy ϵ_2-edge-LDP for each user. By Property 2, the local differential private releasing algorithm satisfies $(\epsilon_1 + \epsilon_2)$-edge-LDP for each user.

G. Proof Of Theorem 9

According to Theorem 6,

$$\begin{aligned}
&\mathbb{E}[(\hat{f}_{c\triangle}(G,\epsilon_1,\epsilon_2),\hat{f}_{f\triangle}(G,\epsilon_1,\epsilon_2))]\\
=&\mathbb{E}[(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^{n}\hat{wc}_i,\frac{1}{1-2p_1}\sum_{i=1}^{n}\hat{wf}_i)]\\
=&\mathbb{E}[(\frac{1}{3(2p_1-1)^2}\sum_{i=1}^{n}wc_i,\frac{1}{1-2p_1}\sum_{i=1}^{n}wf_i)\\
=&(f_{c\triangle}(G),f_{f\triangle}(G)).
\end{aligned}$$

H. Proof Of Theorem 10

(1) For the released number of cycle triangles,

$$\begin{aligned}
&\mathbb{V}[\hat{f}_{c\triangle}(G,\epsilon_1,\epsilon_2)]\\
=&\mathbb{V}[\frac{1}{3(2p_1-1)^2}\sum_{i=1}^{n}\hat{wc}_i]\\
=&\frac{1}{9(2p_1-1)^4}\mathbb{V}[\sum_{i=1}^{n}\hat{wc}_i]\\
=&\frac{1}{9(2p_1-1)^4}(\mathbb{V}[\sum_{i=1}^{n}(t_i-p_1t_i^{(1)}-p_1t_i^{(2)}+p_1^2l_i)]\\
&+\mathbb{V}[\sum_{i=1}^{n}\mathrm{Lap}(\frac{2(n-2)+2\tilde{d}_{max}}{\epsilon_2})]).
\end{aligned}$$

We consider the term $\mathbb{V}[\sum_{i=1}^{n}(t_i-p_1t_i^{(1)}-p_1t_i^{(2)}+p_1^2l_i)]$.

$$\begin{aligned}
&\mathbb{V}[\sum_{i=1}^{n}(t_i-p_1t_i^{(1)}-p_1t_i^{(2)}+p_1^2l_i)]\\
=&\mathbb{V}[\sum_{i=1}^{n}t_i]+p_1^2\mathbb{V}[\sum_{i=1}^{n}t_i^{(1)}]+p_1^2\mathbb{V}[\sum_{i=1}^{n}t_i^{(2)}]\\
&-2p_1\mathrm{Cov}(\sum_{i=1}^{n}t_i,\sum_{i=1}^{n}t_i^{(1)})-2p_1\mathrm{Cov}(\sum_{i=1}^{n}t_i,\sum_{i=1}^{n}t_i^{(2)})\\
&+2p_1^2\mathrm{Cov}(\sum_{i=1}^{n}t_i^{(1)},\sum_{i=1}^{n}t_i^{(2)}).
\end{aligned}$$

According to Cauchy-Schwarz inequality,

$$\begin{aligned}
-\mathrm{Cov}(\sum_{i=1}^{n}t_i,\sum_{i=1}^{n}t_i^{(1)})&\le\sqrt{\mathbb{V}[\sum_{i=1}^{n}t_i]\cdot\mathbb{V}[\sum_{i=1}^{n}t_i^{(1)}]},\\
-\mathrm{Cov}(\sum_{i=1}^{n}t_i,\sum_{i=1}^{n}t_i^{(2)})&\le\sqrt{\mathbb{V}[\sum_{i=1}^{n}t_i]\cdot\mathbb{V}[\sum_{i=1}^{n}t_i^{(2)}]},\\
\mathrm{Cov}(\sum_{i=1}^{n}t_i^{(1)},\sum_{i=1}^{n}t_i^{(2)})&\le\sqrt{\mathbb{V}[\sum_{i=1}^{n}t_i^{(1)}]\cdot\mathbb{V}[\sum_{i=1}^{n}t_i^{(2)}]}.
\end{aligned}$$

Now, we bound the term $\mathbb{V}[\sum_{i=1}^{n}t_i]$.

$$\begin{aligned}
&\sum_{i=1}^{n}t_i\\
=&\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E}).
\end{aligned}$$

For any fixed $i,j\in[n]$, let,

$$a_{i,j}=\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E}),$$

It is easy to show that if i,j are fixed, then for every k, $\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})$ are independent random variables. Then,

$$\begin{aligned}
\mathbb{V}[a_{i,j}]&=\mathbb{V}[\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})]\\
&=\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})].
\end{aligned}$$

For fixed i, j and k, to bound the any term, we consider three cases: **Case-1:** $\lambda_{j,k}=\lambda_{k,i}=0$; **Case-2:** $\lambda_{j,k}=1$, $\lambda_{k,i}=0$ or $\lambda_{j,k}=0$, $\lambda_{k,i}=1$; **Case-3:** $\lambda_{j,k}=\lambda_{k,i}=1$. For **Case-1**, $\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})]=p_1^2(1-p_1^2)$. For **Case-2**, $\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})]=p_1(1-p_1)(1-p_1(1-p_1))$. For **Case-3**, $\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})]=(1-p_1)^2(1-(1-p_1)^2)$. Since $0<p_1=\frac{1}{e^{\epsilon_1}+1}\le\frac{1}{2}$, for all cases, $\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E}\wedge<v_k,v_i>\in\overline{E})]\le(1-p_1)^2(1-(1-p_1)^2)$.

Thus, for any $i,j\in[n]$, where $i\neq j$, $\mathbb{V}[a_{i,j}]\le(n-2)(1-p_1)^2(1-(1-p_1)^2)$.

According to Cauchy-Schwarz inequality, for any $i,j,i',j'\in[n]$, where $i\neq j$ or $i'\neq j'$,

$$\begin{aligned}
\mathrm{Cov}(a_{i,j},a_{i',j'})&\le\sqrt{\mathbb{V}[a_{i,j}]\cdot\mathbb{V}[a_{i',j'}]}\\
&\le(n-2)(1-p_1)^2(1-(1-p_1)^2).
\end{aligned}$$

Let $\mathrm{Cov}(a_{i,j},a_{i,j})=\mathbb{V}[a_{i,j}]$. Then,

$$\begin{aligned}
&\mathbb{V}[\sum_{i=1}^{n}t_i]=\mathbb{V}[\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}a_{i,j}]\\
&=\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{i'=1}^{n}\sum_{j'\in[n]:\tilde{\lambda}_{i',j'}=1}\mathrm{Cov}(a_{i,j},a_{i',j'})\\
&\le n^2\tilde{d}_{max}^2(n-2)(1-p_1)^2(1-(1-p_1)^2).
\end{aligned}$$

Then, we bound the term $\mathbb{V}[\sum_{i=1}^{n}t_i^{(1)}]$,

$$\sum_{i=1}^{n}t_i^{(1)}=\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_j,v_k>\in\overline{E}).$$

It is easy to show that for every $j,k\in[n]$ and $j\neq k$, $\mathbb{I}(<v_j,v_k>\in\overline{E})$ are independent random variables. Then,

$$\begin{aligned}\mathbb{V}[t_i^{(1)}] &= \mathbb{V}[\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_j,v_k>\in\overline{E})]\\ &= \sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E})]\\ &\leq (n-2)\tilde{d}_{max}p_1(1-p_1).\end{aligned}$$

According to Cauchy-Schwarz inequality, for any $i, i^{'} \in [n]$, where $i \neq i^{'}$,

$$\begin{aligned}\mathrm{Cov}(t_i^{(1)}, t_{i'}^{(1)}) &\leq \sqrt{\mathbb{V}[t_i^{(1)}]\cdot\mathbb{V}[t_{i'}^{(1)}]}\\ &\leq (n-2)\tilde{d}_{max}p_1(1-p_1).\end{aligned}$$

Let $\mathrm{Cov}(b_i, b_i) = \mathbb{V}(b_i)$. Then,

$$\begin{aligned}&\mathbb{V}[\sum_{i=1}^{n}t_i^{(1)}]\\ =&\sum_{i=1}^{n}\sum_{i'=1}^{n}\mathrm{Cov}(t_i^{(1)}, t_{i'}^{(1)})\\ \leq& n^2(n-2)\tilde{d}_{max}p_1(1-p_1).\end{aligned}$$

Similarly, we bound the term $\mathbb{V}[\sum_{i=1}^{n}t_i^{(2)}]$,

$$\sum_{i=1}^{n}t_i^{(2)} = \sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_k,v_i>\in\overline{E}).$$

For any fixed $i, j \in [n]$, let

$$c_{i,j} = \sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_k,v_i>\in\overline{E}).$$

It is easy to show that if i is fixed, then for every k, $\mathbb{I}(<v_k, v_i>\in\overline{E})$ are independent random variables. Then,

$$\begin{aligned}\mathbb{V}[c_{i,j}] &= \mathbb{V}[\sum_{k\in[n]:k\neq i,k\neq j}\mathbb{I}(<v_k,v_i>\in\overline{E})]\\ &= \sum_{k\in[n]:k\neq i,k\neq j}\mathbb{V}[\mathbb{I}(<v_k,v_i>\in\overline{E})]\\ &= (n-2)p_1(1-p_1).\end{aligned}$$

According to Cauchy-Schwarz inequality, for any $i, j, i^{'}, j^{'} \in [n]$, where $i \neq j$ or $i^{'} \neq j^{'}$,

$$\begin{aligned}\mathrm{Cov}(c_{i,j}, c_{i',j'}) &\leq \sqrt{\mathbb{V}[c_{i,j}]\cdot\mathbb{V}[c_{i',j'}]}\\ &\leq (n-2)p_1(1-p_1).\end{aligned}$$

Let $\mathrm{Cov}(c_{i,j}, c_{i,j}) = \mathbb{V}[c_{i,j}]$. Then,

$$\begin{aligned}\mathbb{V}[\sum_{i=1}^{n}t_i^{(2)}] &= \mathbb{V}[\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}c_{i,j}]\\ &= \sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{i'=1}^{n}\sum_{j'\in[n]:\tilde{\lambda}_{i',j'}=1}\mathrm{Cov}(c_{i,j}, c_{i',j'})\\ &\leq n^2\tilde{d}_{max}^2(n-2)p_1(1-p_1).\end{aligned}$$

Thus,

$$-\mathrm{Cov}(\sum_{i=1}^{n}t_i, \sum_{i=1}^{n}t_i^{(1)}) \leq n^2(n-2)\tilde{d}_{max}^{\frac{3}{2}}(1-p_1)^{\frac{3}{2}}p_1(2-p_1)^{\frac{1}{2}},$$

$$-\mathrm{Cov}(\sum_{i=1}^{n}t_i, \sum_{i=1}^{n}t_i^{(2)}) \leq n^2(n-2)\tilde{d}_{max}^{2}(1-p_1)^{\frac{3}{2}}p_1(2-p_1)^{\frac{1}{2}},$$

$$\mathrm{Cov}(\sum_{i=1}^{n}t_i^{(1)}, \sum_{i=1}^{n}t_i^{(2)}) \leq n^2(n-2)\tilde{d}_{max}^{\frac{3}{2}}p_1(1-p_1).$$

Then, since $0 < p_1 \leq \frac{1}{2}$,

$$\mathbb{V}[\sum_{i=1}^{n}(t_i - p_1t_i^{(1)} - p_1t_i^{(2)} + p_1^2l_i)] \leq O((1-p_1)^4n^3\tilde{d}_{max}^2).$$

Then,

$$\begin{aligned}\mathbb{V}[\hat{f}_{c\triangle}(G,\epsilon_1,\epsilon_2)] &\leq O(\frac{(1-p_1)^4}{(2p_1-1)^4}n^3\tilde{d}_{max}^2 + \frac{n^3}{(2p_1-1)^4\epsilon_2^2})\\ &= O(\frac{e^{4\epsilon_1}}{(1-e^{\epsilon_1})^4}n^3(\tilde{d}_{max}^2 + \frac{1}{\epsilon_2^2})).\end{aligned}$$

(2) For the released number of flow triangles,

$$\begin{aligned}\mathbb{V}[\hat{f}_{f\triangle}(G,\epsilon_1,\epsilon_2)] &= \mathbb{V}[\frac{1}{1-2p_1}\sum_{i=1}^{n}\hat{w}f_i]\\ &= \frac{1}{(1-2p_1)^2}\mathbb{V}[\sum_{i=1}^{n}\hat{w}f_i]\\ &= \frac{1}{(1-2p_1)^2}(\mathbb{V}[\sum_{i=1}^{n}(r_i - p_1s_i)]\\ &+ \mathbb{V}[\sum_{i=1}^{n}\mathrm{Lap}(\frac{2(n-2)+2\tilde{d}_{max}}{\epsilon_2})]).\end{aligned}$$

We consider the term $\mathbb{V}[\sum_{i=1}^{n}(r_i - p_1s_i)]$,

$$\begin{aligned}&\mathbb{V}[\sum_{i=1}^{n}(r_i - p_1s_i)]\\ =&\mathbb{V}[\sum_{i=1}^{n}r_i]\\ =&\mathbb{V}[\sum_{i=1}^{n}\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:\tilde{\lambda}_{i,k}=1,j\neq k}\mathbb{I}(<v_j,v_k>\in\overline{E})].\end{aligned}$$

It is easy to show that for every $j, k \in [n]$, $j \neq k$, $\mathbb{I}(<v_j, v_k>\in\overline{E})$ are independent random variables. Then,

$$\begin{aligned}\mathbb{V}[r_i] &= \mathbb{V}[\sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:\tilde{\lambda}_{i,k}=1,j\neq k}\mathbb{I}(<v_j,v_k>\in\overline{E})]\\ &= \sum_{j\in[n]:\tilde{\lambda}_{i,j}=1}\sum_{k\in[n]:\tilde{\lambda}_{i,k}=1,j\neq k}\mathbb{V}[\mathbb{I}(<v_j,v_k>\in\overline{E})]\\ &\leq \tilde{d}_{max}(\tilde{d}_{max}-1)p_1(1-p_1).\end{aligned}$$

According to Cauchy-Schwarz inequality, for any $i, i^{'} \in [n]$, where $i \neq i^{'}$,

$$\begin{aligned}\mathrm{Cov}(r_i, r_{i'}) &\leq \sqrt{\mathbb{V}[r_i] \cdot \mathbb{V}[r_{i'}]} \\ &\leq \tilde{d}_{max}(\tilde{d}_{max} - 1)p_1(1 - p_1).\end{aligned}$$

Let $\mathrm{Cov}(r_i, r_i) = \mathbb{V}[r_i]$. Then,

$$\begin{aligned}\mathbb{V}[\sum_{i=1}^{n}(r_i - p_1 s_i)] &= \mathbb{V}[\sum_{i=1}^{n} r_i] \\ &= \sum_{i=1}^{n}\sum_{i'=1}^{n} \mathrm{Cov}(r_i, r_{i'}) \\ &\leq n^2 \tilde{d}_{max}(\tilde{d}_{max} - 1)p_1(1 - p_1).\end{aligned}$$

Then,

$$\mathbb{V}[\hat{f}_{f\triangle}(G, \epsilon_1, \epsilon_2)] \leq O(\frac{e^{\epsilon_1}}{(1 - e^{\epsilon_1})^2} n^2 (\tilde{d}_{max}^2 + \frac{e^{\epsilon_1} n}{\epsilon_2^2})).$$

I. Missing Experimental Results

1) Impact of Privacy Budget on CDP: Varying privacy budget, the performance of CDP in terms of running time, relative error, and L_2 loss is shown in Fig. 16 and Fig. 17.

2) Impact of Projection Degree on CDP: Varying projection degree, the performance of CDP in terms of running time, relative error, and L_2 loss is shown in Fig. 18 and Fig. 19.

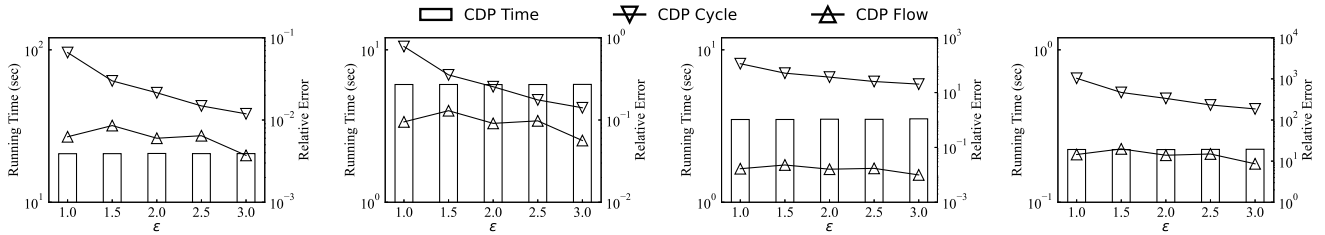
3) Impact of Graph Size on CDP: Varying graph size, the performance of CDP in terms of running time, relative error, and L_2 loss is shown in Fig. 20 and Fig. 21.

4) Impact of Privacy Budget on LDP: Varying privacy budget, the performance of LDP in terms of running time, relative error, and L_2 loss is shown in Fig. 22 and Fig. 23.

5) Impact of Projection Degree on LDP: Varying projection degree, the performance of LDP in terms of running time, relative error, and L_2 loss is shown in Fig. 24 and Fig. 25.

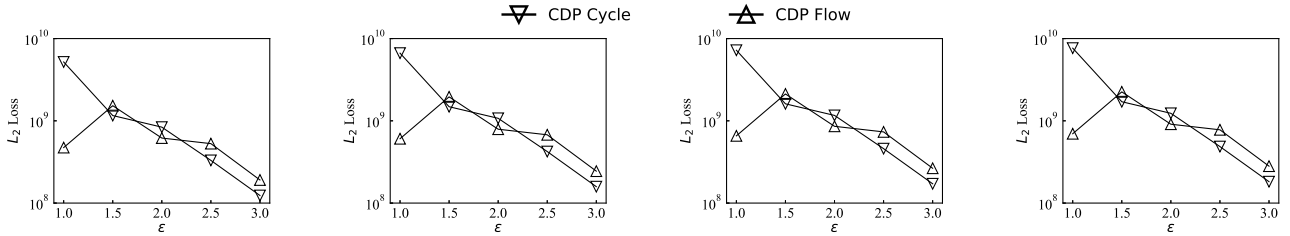
6) Impact of Graph Size on LDP: Varying graph size, the performance of LDP in terms of running time, relative error, and L_2 loss is shown in Fig. 26 and Fig. 27.

7) Impact of Privacy Budget Allocation on LDP: Varying privacy budger allocation ratio, the performance of LDP in terms of running time, relative error, and L_2 loss is shown in Fig. 28 and Fig. 29.



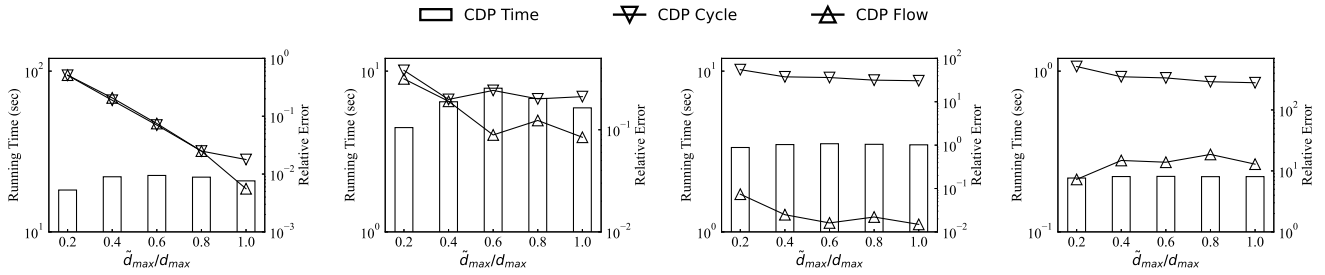
(a) MO (b) AC (c) CH (d) PG

Fig. 16: Running time and relative error on varying ϵ (CDP)



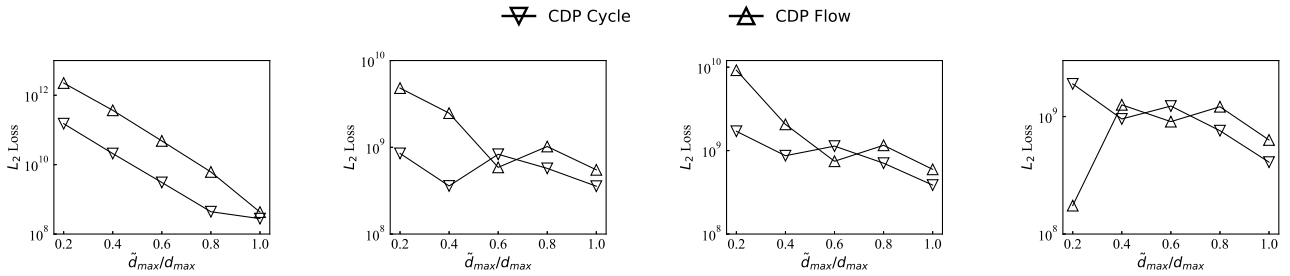
(a) MO (b) AC (c) CH (d) PG

Fig. 17: L_2 loss on varying ϵ (CDP)



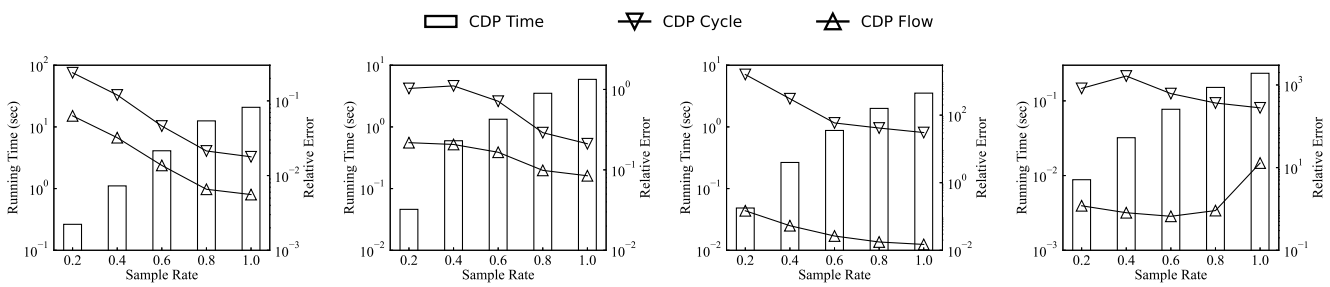
(a) MO (b) AC (c) CH (d) PG

Fig. 18: Running time and relative error on varying $\tilde{d}_{max}$ (CDP)



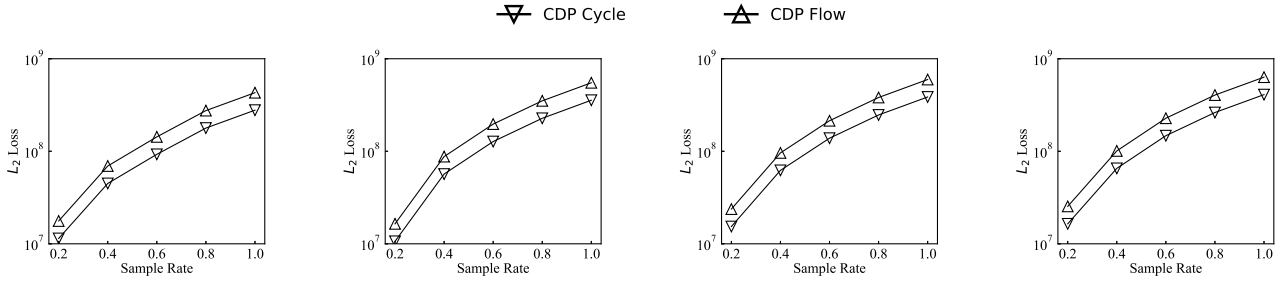
(a) MO (b) AC (c) CH (d) PG

Fig. 19: L_2 loss on varying $\tilde{d}_{max}$ (CDP)



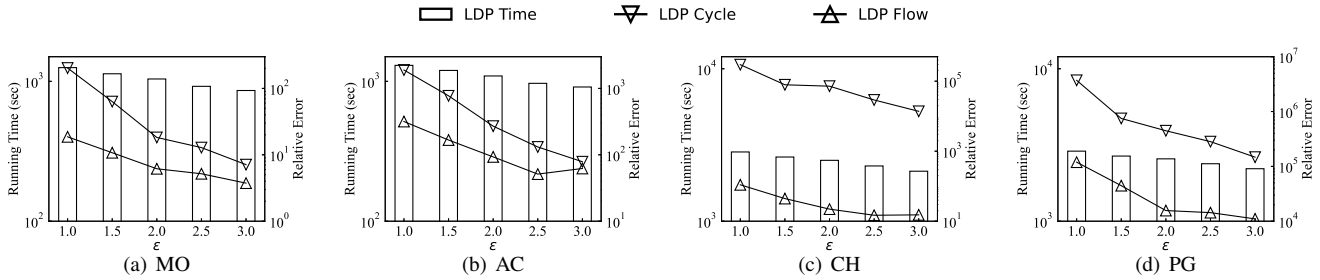
(a) MO (b) AC (c) CH (d) PG

Fig. 20: Running time and relative error on varying $|V|$ (CDP)



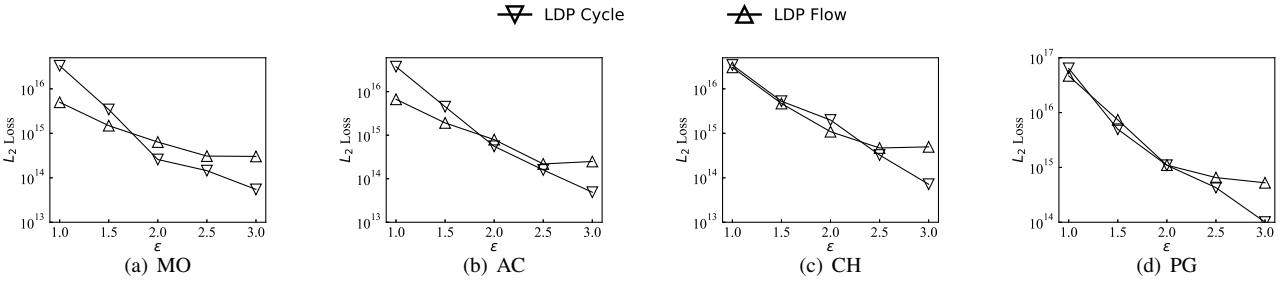
(a) MO (b) AC (c) CH (d) PG

Fig. 21: L_2 loss on varying $|V|$ (CDP)



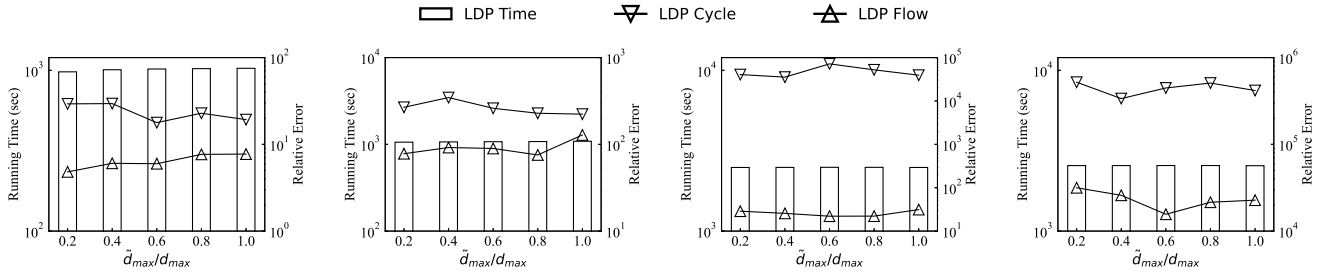
(a) MO (b) AC (c) CH (d) PG

Fig. 22: Running time and relative error on varying ϵ (LDP)



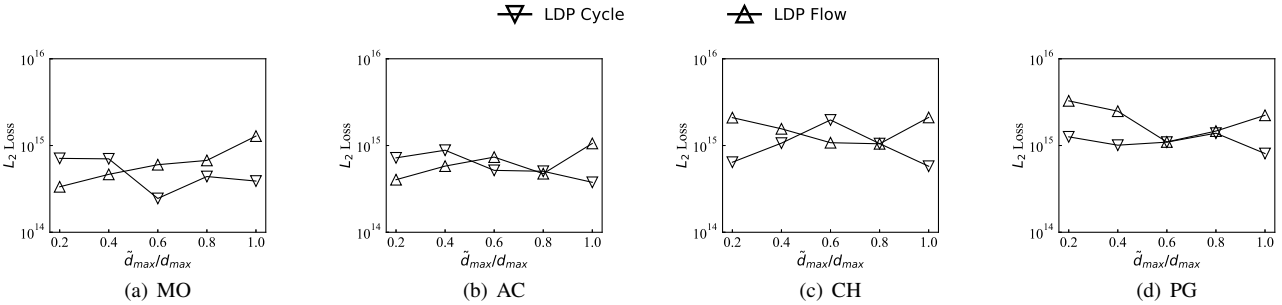
(a) MO (b) AC (c) CH (d) PG

Fig. 23: L_2 loss on varying ϵ (LDP)



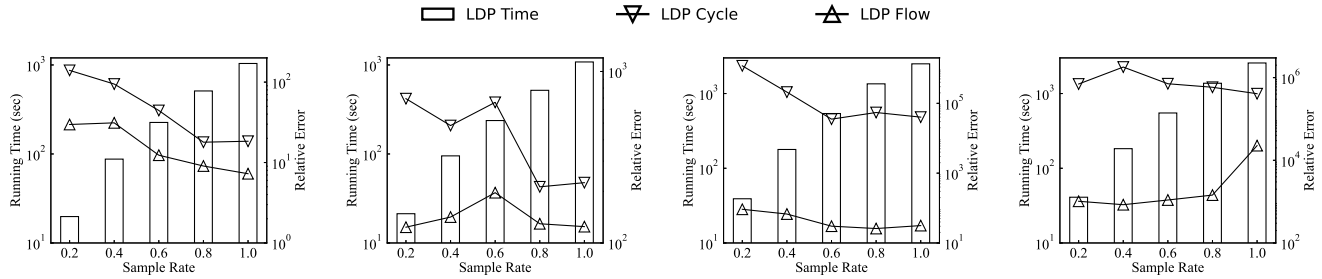
(a) MO (b) AC (c) CH (d) PG

Fig. 24: Running time and relative error on varying $\tilde{d}_{max}$ (LDP)



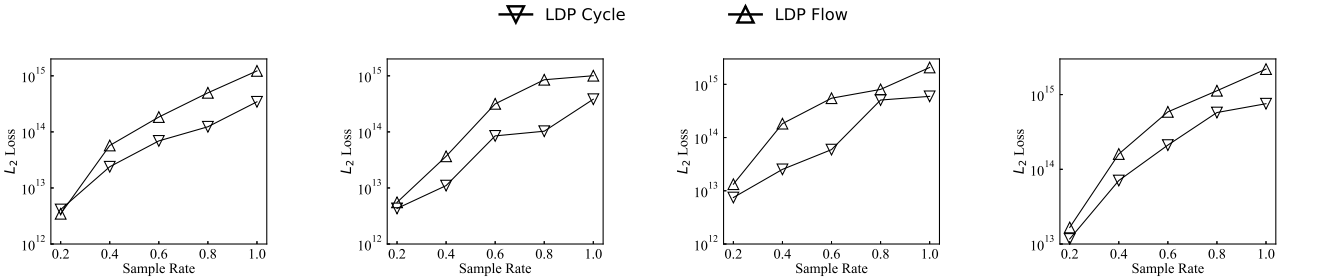
(a) MO (b) AC (c) CH (d) PG

Fig. 25: L_2 loss on varying $\tilde{d}_{max}$ (LDP)



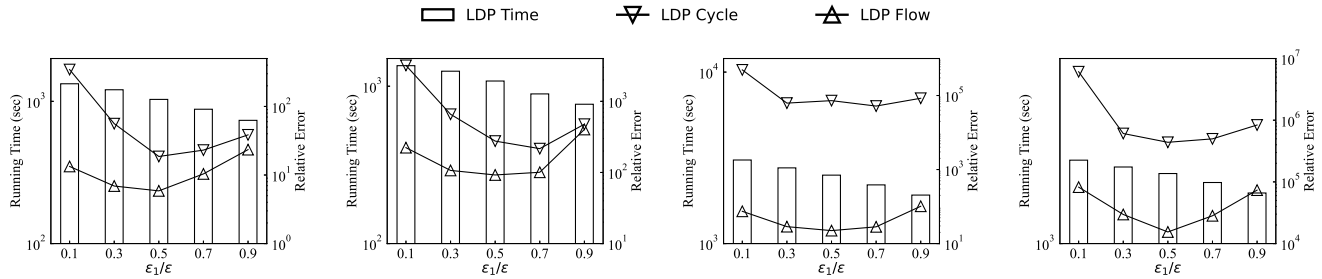
(a) MO (b) AC (c) CH (d) PG

Fig. 26: Running time and relative error on varying $|V|$ (LDP)



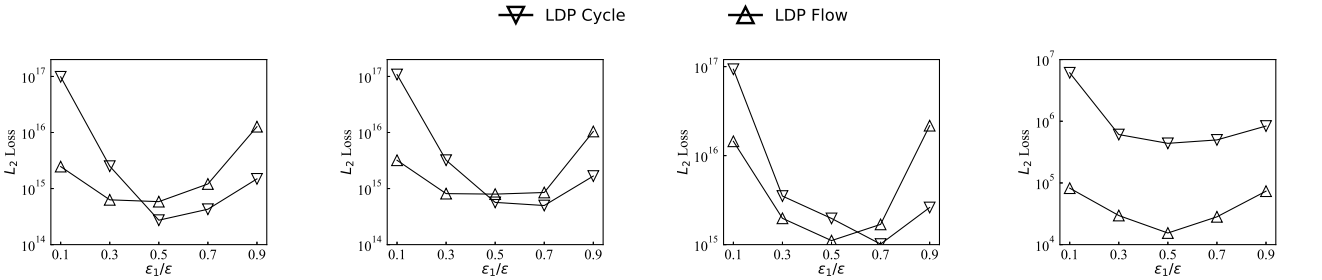
(a) MO (b) AC (c) CH (d) PG

Fig. 27: L_2 loss on varying $|V|$ (LDP)



(a) MO (b) AC (c) CH (d) PG

Fig. 28: Running time and relative error on varying $\frac{\epsilon_1}{\epsilon}$ (LDP)



(a) MO (b) AC (c) CH (d) PG

Fig. 29: L_2 loss on varying $\frac{\epsilon_1}{\epsilon}$ (LDP)