Practical and Accurate Local Edge Differentially Private Graph Algorithms

Pranay Mundra
Yale University
New Haven, CT, USA
pranay.mundra@yale.edu

Charalampos Papamanthou
Yale University
New Haven, CT, USA
charalampos.papamanthou@yale.edu

Julian Shun
MIT CSAIL
Cambridge, MA, USA
jshun@mit.edu

Quanquan C. Liu
Yale University
New Haven, CT, USA
quanquan.liu@yale.edu

ABSTRACT

The rise of massive networks across diverse domains necessitates sophisticated graph analytics, often involving sensitive data and raising privacy concerns. This paper addresses these challenges using *local differential privacy (LDP)*, which enforces privacy at the individual level, where *no third-party entity is trusted*, unlike centralized models that assume a trusted curator.

We introduce novel LDP algorithms for two fundamental graph statistics: k-core decomposition and triangle counting. Our approach leverages previously unexplored input-dependent private graph properties, specifically the degeneracy and maximum degree of the graph, to improve theoretical utility. Unlike prior methods, our error bounds are determined by the maximum degree rather than the total number of edges, resulting in significantly tighter theoretical guarantees. For triangle counting, we improve upon the previous work of Imola, Murakami, and Chaudhury [37, 38], which bounds error in terms of the total number of edges. Instead, our algorithm achieves error bounds based on the graph's degeneracy by leveraging a differentially private out-degree orientation, a refined variant of Eden et al.'s randomized response technique [22], and a novel, intricate analysis, yielding improved theoretical guarantees over prior state-of-the-art.

Beyond theoretical improvements, we are the first to evaluate the practicality of local DP algorithms in a distributed simulation environment, unlike previous works that tested on a single processor. Our experiments on real-world datasets demonstrate substantial accuracy improvements, with our k-core decomposition achieving errors within **3x** the exact values—far outperforming the **131x** error in the baseline of Dhulipala et al. [15] . Additionally, our triangle counting algorithm reduces multiplicative approximation errors by up to **six orders of magnitude** compared to prior methods, all while maintaining competitive runtime performance.

PVLDB Reference Format:
Pranay Mundra, Charalampos Papamanthou, Julian Shun, and Quanquan C. Liu. Practical and Accurate Local Edge Differentially Private Graph Algorithms. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.

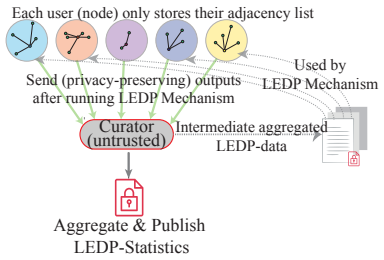


Figure 1: Local edge differential privacy (LEDP) Model

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

1 INTRODUCTION

Graph statistics such as the k-core decomposition and triangle count provide important characteristics about the underlying graph, such as its well-connected communities. These analytics, often performed on sensitive and private graphs, are regularly shared with a wide audience, including researchers, companies, governments, and the public. As such, it is vital to investigate techniques that can safeguard these published graph statistics from privacy attacks.

The *k-core decomposition* assigns an "importance" value to each node, roughly representing its influence within the graph. It is widely used to analyze the structure of real-world graphs, including social, email, and disease transmission networks. Formally, a k-core of a graph is a maximal subgraph where the induced degree of every vertex in the subgraph is at least k. The k-core decomposition (see Definition 2.16 and Figure 2) assigns a number, denoted as $core(v)$, to each vertex v. This number, $core(v)$, represents the largest value of k for which the k-core still includes vertex v. Unfortunately, these values pose privacy risks.

Consider the application of k-core decomposition to COVID transmission data [10, 29, 63, 67, 70] and other disease transmission networks [11] such as HIV [27, 35]. The core numbers are generated and published, sometimes even with location data [69]. Revealing the precise core numbers of every individual can lead to privacy breaches. Consider a scenario where exactly c individuals have a core number of $c - 1$. This implies they form a clique of c vertices,

doi:XX.XX/XXX.XX

all connected. In disease transmission graphs, this directly exposes a cluster of sensitive disease transmissions! Therefore, it's essential to release privacy-preserving core numbers.

Similarly, *triangle counting* is widely used in applications that process sensitive data. The triangle count, which measures the number of three-node cycles in a graph, is a fundamental metric in community detection [54, 61, 62, 72], recommendation systems [52, 86], and clustering [78]. In databases, triangle counting is essential for graph analytics frameworks [58] and is leveraged in query optimization [3, 6] and fraud detection [74]. However, exposing triangle counts without privacy guarantees can lead to inference attacks that compromise user confidentiality. Recent works in security and privacy [37, 38, 48, 50, 51] have highlighted the risks associated with publishing triangle counts, reinforcing the importance of privacy-preserving graph analytics.

As databases increasingly incorporate graph-based queries and analytics, ensuring privacy in fundamental graph statistics such as k-core decomposition and triangle counting becomes critical. This motivates the need for rigorous privacy mechanisms that protect structural graph information while preserving utility.

Central and local differential privacy. Differential privacy [19] is often considered the "gold standard" in protecting the privacy of individuals. Differential privacy has traditionally been studied in the *central* model where *all* of the private information is revealed to a *trusted curator* who is responsible for parsing the data and producing privacy-preserving outputs. But such an assumption is often unrealistic. Today's world requires computation over *decentralized* networks involving the transfer of sensitive personal information; hence, preserving *local* privacy is crucial.

The *local model of differential privacy*, introduced by the seminal works of Evfimievski et al. [23] and Kasiviswanathan et al. [40], has recently gained much attention in the theoretical computer science [15, 16, 22, 31], cryptography and security [26, 37, 38, 45, 48, 49, 79, 87], data mining [7, 33, 34, 50, 51, 56, 59], query answering [25, 42, 66, 77, 80], and machine learning [28, 30, 32, 39, 46, 55, 68, 81, 83, 88] communities. In this model, users independently apply differential privacy mechanisms before sharing their (privacy-preserving) outputs with an *untrusted* curator, who aggregates the data to compute statistics without accessing the original sensitive information. The local model ensures *stronger privacy guarantees* by preventing direct data sharing. Because of both its strong privacy guarantees and its easy adaptation to distributed applications, the local model of differential privacy has been used in a variety of prominent cases including federated learning [41, 53, 57], the 2020 U.S. Census [2], and by companies such as Apple [75].

Local Edge Differential Privacy and Randomized Response. Despite growing interest in local privacy, *local differentially private (LDP) graph algorithms* have only recently gained significant attention. The *local edge differential privacy (LEDP) model*, introduced in recent works [15, 37, 64], provides a novel framework for ensuring local privacy in *graph outputs* (Fig. 1). Unlike traditional databases, where privacy mechanisms focus on protecting individual records, graph data introduces the additional challenge of safeguarding *sensitive relationships* between entities. Graph data is increasingly integral to modern database systems, underpinning applications in knowledge graphs, social network analysis, cybersecurity, and financial fraud detection. Many relational databases now support graph extensions (e.g., SQL-based graph queries), while specialized graph databases [5, 58] are widely deployed in industry. These systems rely on efficient graph analytics, yet integrating LEDP mechanisms remains challenging due to the high computational overhead and large errors induced by strict privacy constraints. Unlike the setting where individual data points can be perturbed using noise, LEDP graph algorithms involve perturbing the edges within a graph, which is inherently more complex.

All previous implementations of local differentially private graph algorithms [36, 38, 64] use *Randomized Response (RR)* [82] to perturb graph edges. This mechanism systematically examines every pair of nodes in the input graph: for each existing edge, it deletes it with probability $1/(e^{\varepsilon}+1)$, and for each non-existent edge, it inserts an edge with the same probability, where ε is the privacy parameter. For small ε (in particular $0 < \varepsilon \leq 1$), the *density*[1] of the input graph increases by a significant amount. Thus, while Randomized Response is an easy and convenient method for obtaining LEDP, it comes at a cost of large error and computation time.

Beyond Randomized Response. Dhulipala et al. [15] were the first to study LEDP algorithms that used privacy mechanisms beyond Randomized Response by using the geometric mechanism on small-round parallel algorithms for k-core decomposition. However, their results are purely theoretical and they do not discuss how to translate their theoretical guarantees to practice. In particular, they give additive error bounds $\geq \frac{\log_2^3(n)}{\varepsilon}$ (where n is the number of vertices); on a graph with $n = 10^5$ nodes and $\varepsilon = 0.5$, this translates to an additive error of 9164. Most real-world graphs of this size have max core numbers of 10^2 magnitude, so the additive error itself leads to a ≥ 91-multiplicative approximation factor—much too large for any practical use.

A major goal of this work is to develop new theoretical and implementation techniques that, together with Randomized Response, enable provably private, accurate, and computationally efficient LEDP algorithms. We propose a novel LEDP k-core decomposition algorithm that incorporates a degree-thresholding technique to [15], improving error guarantees by bounding them in terms of the *maximum degree* rather than the total number of nodes. As demonstrated in Table 1 (in Section 5), the maximum degree is often significantly smaller than the number of nodes or edges, leading to improved accuracy in practical settings.

Building on our k-core decomposition algorithm, we introduce a novel LEDP edge orientation technique, ensuring that the out-degree is approximately bounded by the graph's maximum core number (degeneracy). We further refine this approach by formulating a variant of the Randomized Response (RR) algorithm of Eden et al. [22] and integrating it with our LEDP out-degree orientation method. This results in a new triangle counting algorithm with error proportional to the maximum core number, improving upon the best-known error bounds [37, 38] for a broad class of graphs. As shown in Table 1, the maximum core number is typically even smaller than the maximum degree, further enhancing accuracy. Since the maximum core number is an input-dependent private property of the graph, previously unexplored in local graph algorithms, accurately computing it while preserving privacy presents

[1]The density is the ratio of the number of edges to the number of nodes.

significant challenges. We present a detailed analysis of our triangle counting algorithm's utility in Section 4.2.

To bridge the gap between theory and practice, we conduct the first evaluation of LEDP algorithms in a distributed simulation environment. While maintaining the same privacy model as prior works [15, 37, 38, 64], our evaluation differs by simulating a distributed execution rather than relying on a single-processor implementation. We partition the graph across multiple processors, each independently running LEDP algorithms on its assigned subgraph while communicating privacy-preserving outputs via message passing over multiple rounds. This closely models real-world distributed environments. We apply this evaluation to both k-core decomposition and triangle counting, demonstrating the scalability and practicality of our algorithms in large-scale distributed settings.

Our approach integrates techniques from parallel and distributed algorithms, combining algorithmic and engineering insights to enhance practicality. Our algorithms scale to significantly larger graphs than those considered in prior private algorithms, handling up to billions of edges. We implement all our algorithms in Golang [17]. In summary, we make the following contributions:

- We design a *novel* LEDP k-core decomposition algorithm that does not use Randomized Response and provides provable privacy and error guarantees. By leveraging the input-dependent maximum degree property of the graph, we achieve improved theoretical bounds over the LEDP k-core decomposition algorithm of Dhulipala et al. [15]. A key innovation lies in thresholding the maximum number of levels a node can move up based on its noisy (private) degree. Since a node's core number is upper bounded by its degree, our algorithm offers stronger theoretical guarantees for most real-world graphs, where the maximum degree is significantly smaller than the number of nodes.
- We present the *first* implementation of a private k-core decomposition algorithm and demonstrate through empirical evaluation that it achieves an average error of **3x** the exact values, markedly improving upon prior approaches. Furthermore, our LEDP implementation attains error rates that closely align with the theoretical approximation bounds of the best *non-private* algorithms, underscoring its practical efficiency and accuracy.
- We design a *novel* LEDP triangle counting algorithm that modifies our k-core decomposition to construct a low out-degree ordering, minimizing each node's out-degree. Leveraging this ordering, our algorithm achieves improved theoretical error bounds over the best-known methods of Imola et al. [37, 38] and Eden et al. [22] for bounded degeneracy graphs, common in real-world networks. Our implementation reduces relative error by up to **89x** and improves the multiplicative approximation by up to **six orders of magnitude** over the best previous implementation [38], while maintaining competitive runtime performance.
- We conduct the first evaluation of LEDP graph algorithms in a distributed simulation setting with actual message passing. Unlike prior studies that relied on a single processor, we simulate a distributed environment by partitioning the graph across multiple processors. This approach provides a more realistic assessment of both computational and communication overhead in large-scale distributed scenarios. We demonstrate the scalability and practicality of this evaluation by applying it to our k-core decomposition and triangle counting algorithms.
- We present the first LEDP graph algorithm implementations that scale to ***billion***-edge graphs, whereas prior implementations were tested on graphs with millions of edges [37, 38]. Our evaluation framework serves as a valuable tool for designing and testing other LEDP algorithms. Our source code is available at [1].

1.1 Related Work

Local differential privacy (LDP) for graph data has been extensively studied [15, 22, 33, 37, 38, 64, 73, 84, 85], focusing on tasks such as synthetic graph generation and subgraph counting. Some works [73] explore an *extended local view*, where nodes observe their neighbors' edges to improve triangle counting accuracy. However, this model is unrealistic in many settings, as users in social networks, for instance, may not wish to reveal their friend lists.

The LEDP model was introduced by Qin et al. [64] and Imola, Murakami, and Chaudhury [37], with subsequent theoretical expansions [15, 22, 38]. Imola et al. [37, 38] provided the first practical LEDP implementations for triangle and subgraph counting. Recently, Hillebrand et al. [33] improved LEDP triangle counting using hash functions, though their method does not scale to large graphs. All prior LEDP triangle counting algorithms rely on Randomized Response. Imola et al. [37] introduced an LEDP triangle counting algorithm in both non-interactive (single-round) and interactive (multi-round) settings, bounding the standard deviation of the additive error by $O\left(\frac{n^2}{\varepsilon} + \frac{n^{3/2}}{\varepsilon^2}\right)$. In a subsequent work, they reduce the protocol's communication cost [38] by using a combination of sampling and clipping techniques, and refined their standard deviation analysis by using the number of 4-cycles, C_4. Their new theoretical standard deviation is $O\left(\frac{\sqrt{C_4}}{\varepsilon} + \frac{n^{3/2}}{\varepsilon^2}\right)$ for the interactive setting and $O(n^2)$ for the non-interactive setting. Eden et al. [22] further enhanced triangle counting accuracy with an improved post-processing analysis, achieving a standard deviation of $O\left(\frac{\sqrt{C_4}}{\varepsilon^2} + \frac{n^{3/2}}{\varepsilon^3}\right)$ for the non-interactive setting and establishing lower bounds of $\Omega(n^2)$ and $\Omega\left(\frac{n^{3/2}}{\varepsilon}\right)$ for the non-interactive and interactive settings, respectively. Despite these advancements, prior work has neither combined Randomized Response with other privacy mechanisms to improve error bounds nor accounted for input-dependent properties of graphs in theoretical analyses.

For LEDP k-core decomposition, all known algorithms remain theoretical [15]. The algorithm by Dhulipala et al. [15] uses a *level data structure*, where nodes ascend levels based on their noisy induced degrees, with noise drawn from a symmetric geometric distribution to ensure privacy. However, this noise scales with the number of nodes rather than adapting to input structure, leading to significant errors in large graphs. Recent concurrent and independent work by Dhulipala et al. [14] introduces a generalized sparse vector technique to avoid cumulative privacy budget costs, achieving improved theoretical guarantees. However, implementing this approach in a distributed setting presents challenges, as it relies on a peeling algorithm that is difficult to distribute in practice. Additionally, the practical performance of these algorithms remains unexplored.

2 PRELIMINARIES

Differential privacy on graphs is defined for *edge-neighboring* inputs. Edge-neighboring inputs are two graphs which differ in exactly one edge. Here, we consider *undirected* graphs.

Definition 2.1 (Edge-Neighboring [60]). *Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are edge-neighboring if they differ in one edge, namely, if $V_1 = V_2$ and the size of the symmetric difference of E_1 and E_2 is 1.* [2]

With high probability (***whp***) is used in this paper to mean with probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$.

The local edge differential privacy (LEDP) model assumes that each node in the input graph keeps their adjacency list private. The model is defined in terms of ε-DP algorithms, called *ε-local randomizers (ε-LR)*, that are run individually by every node. The ε-LRs are guaranteed to be ε-DP when the neighboring inputs are adjacency lists that differ in one element. Following [38], we assume that the curator and all nodes act as honest-but-curious adversaries.

Definition 2.2 (ε-Edge Differential Privacy [19, 60]). *Algorithm $\mathcal{A}(G)$, that takes as input a graph G and outputs some value in $Range(\mathcal{A})$,* [3] *is ε-**edge differentially private** (ε-edge DP) if for all $S \subseteq Range(\mathcal{A})$ and all edge-neighboring graphs G and G',*

$$\frac{1}{e^{\varepsilon}} \leq \frac{\Pr[\mathcal{A}(G') \in S]}{\Pr[\mathcal{A}(G) \in S]} \leq e^{\varepsilon}.$$

Definition 2.3 (Local Randomizer (LR) [15]). *An ε-**local randomizer** $R : \mathbf{a} \to \mathcal{Y}$ for node v is an ε-edge DP algorithm that takes as input the set of its neighbors $N(v)$, represented by an adjacency list $\mathbf{a} = (b_1, \ldots, b_{|N(v)|})$. In other words,*

$$\frac{1}{e^{\varepsilon}} \leq \frac{\Pr\left[R(\mathbf{a}') \in Y\right]}{\Pr\left[R(\mathbf{a}) \in Y\right]} \leq e^{\varepsilon}$$

for all $\mathbf{a}$ and $\mathbf{a}'$ where the symmetric difference is 1 and all sets of outputs $Y \subseteq \mathcal{Y}$. The probability is taken over the random choices of R (but not over the choice of the input).

The previous definitions of LEDP [15, 37, 64] are satisfied by the following Definition 2.4. [15] gives a slightly more general and complex definition of LEDP in terms of *transcripts* but all of the algorithms in our paper satisfy our definition below, which is also guaranteed to satisfy their more general transcript-based definition.

Definition 2.4 (Local Edge Differential Privacy (LEDP) [15]). *Given an input graph $G = (V, E)$, for any edge $\{u, v\}$, let algorithm $\mathcal{A}$ assign $\left((R_1^u(\mathbf{a}_u, p_1), \varepsilon_1^u), \ldots, (R_r^u(\mathbf{a}_u, p_r), \varepsilon_r^u)\right)$ to be the set of ε_i^u-local randomizers called by vertex u during each interactive round and $\left((R_1^v(\mathbf{a}_v, p_1), \varepsilon_1^v), \ldots, (R_s^v(\mathbf{a}_v, p_s), \varepsilon_s^v)\right)$ be the set of ε_i^v-LRs called by v. The private adjacency lists of u and v are given by $\mathbf{a}_u$ and $\mathbf{a}_v$, respectively, and p_i are the new public information released in each round. Algorithm $\mathcal{A}$ is ε-**local edge differentially private (LEDP)** if for every edge, $\{u, v\}$:*

$$\varepsilon_1^u + \cdots + \varepsilon_r^u + \varepsilon_1^v + \cdots + \varepsilon_s^v \leq \varepsilon.$$

For intuition, each LR takes as input the private adjacency list of the node v and public information released in previous rounds; then, it releases new public information for v which will inform the computation of other nodes in the next round. Hence, the algorithm is *interactive*. Each time v releases, it loses some amount of privacy indicated by ε_i^v for the i-th LR. Since edge-neighboring graphs differ in exactly one edge, to ensure the privacy of the system, it is sufficient to ensure that the privacy loss of every edge sums up to ε. Thus, ε-LEDP algorithms also satisfy ε-DP (proven in [15]).

[2] The *symmetric difference* of two sets is the set of elements that are in either set, but not in their intersection.

[3] *$Range(\cdot)$* denotes the set of all possible outputs of a function.

2.1 Privacy Tools

We make use of the following privacy tools and primitives. We define all definitions below in terms of edge-neighboring adjacency lists since our tools will be applied to ε-local randomizers.

Definition 2.5 (Global Sensitivity [19]). *For a function $f : \mathcal{D} \to \mathbb{R}^d$, where $\mathcal{D}$ is the domain of f and d is the dimension of the output, the ℓ_1-**sensitivity** of f is $GS_f = \max_{\mathbf{a},\mathbf{a}'} \|f(\mathbf{a}) - f(\mathbf{a}')\|_1$ for all pairs of $\{\mathbf{a}, \mathbf{a}'\}$ of neighboring adjacency lists (differing in one neighbor).*

Our algorithms and implementations in this paper use the *symmetric geometric distribution* defined in previous papers [4, 9, 19, 20, 24, 71]. The symmetric geometric distribution is also often referred to as the "discrete Laplace distribution." Using this distribution is quite crucial in practice in order to avoid the numerical errors associated with real-valued outputs from continuous distributions.

Definition 2.6 (Symmetric Geometric Distribution [4, 71]). *The **symmetric geometric distribution**, denoted $\text{Geom}(b)$, with input parameter $b \in (0, 1)$, takes integer values i where the probability mass function at i is given by $\frac{e^b - 1}{e^b + 1} \cdot e^{-|i| \cdot b}$.*

We denote a random variable drawn from this distribution by $X \sim \text{Geom}(b)$. ***With high probability*** (***whp***) is used in this paper to mean with probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$. As with all DP algorithms, privacy is *always* guaranteed and the approximation factors are guaranteed whp. We can upper bound the symmetric geometric noise whp using the following lemma.

Lemma 2.7 ([4, 9, 15, 19, 20, 24, 71]). *With probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$, we can upper bound $X \sim \text{Geom}(x)$ by $|X| \leq \frac{c \ln n}{x}$.*

The ***geometric mechanism*** is defined as follows.

Definition 2.8 (Geometric Mechanism [4, 9, 19, 20]). *Given any function $f : \mathcal{D} \to \mathbb{Z}^d$, where $\mathcal{D}$ is the domain of f and GS_f is the ℓ_1-sensitivity of f, the geometric mechanism is defined as $\mathcal{M}(\mathbf{a}, f(\cdot), \varepsilon) = f(\mathbf{a}) + (Y_1, \ldots, Y_d)$, where $Y_i \sim \text{Geom}(\varepsilon / GS_f)$ are independent and identically distributed (i.i.d.) random variables drawn from $\text{Geom}(\varepsilon / GS_f)$ and $\mathbf{a}$ is a private input adjacency list.*

Definition 2.9 (Laplace Distribution). *The probability density function of the Laplace distribution on $X \in \mathbb{R}$ is $Lap(b) = 2b \cdot \exp(-(|X| \cdot b))$*

Lemma 2.10 (Laplace Mechanism [19]). *Given a function $f : \mathcal{G} \to R$ with sensitivity Δ_f, $f(\mathcal{G}) + Lap\left(\frac{\varepsilon}{\Delta_f}\right)$ is ε-differentially private.*

Lemma 2.11 (Privacy of the Geometric Mechanism [4, 9, 19, 20]). *The geometric mechanism is ε-DP.*

In addition to the Geometric Mechanism, our paper also uses Randomized Response (RR). Randomized Response (RR) when applied to graphs flips the bit indicating the existence of an edge in the graph. We define RR in terms of the way it is used on graphs.

Definition 2.12 (Randomized Response). *Randomized response on input graph $G = (V, E)$, represented as an upper triangular adjacency*

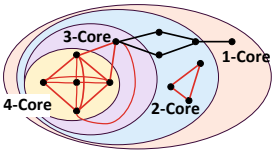


Figure 2: Example k-core decomposition and triangles in a 4-degenerate graph. Nodes are assigned core numbers based on the highest value core they belong to; e.g., a node in the 1-core but not in the 2-core is given the core number of 1. Larger valued cores are contained within all smaller valued cores; e.g., the 3-core is contained in the 1 and 2-core. Red edges show the triangles, i.e., 3-cycles in the graph. The degeneracy of this graph is 4.

matrix M which only contains entries $M[i, j]$ where $i < j$, flips every bit $M[i, j]$ (where $i < j$) in the matrix with probability $\frac{1}{e^{\varepsilon}+1}$.

It is well-known that randomized response is ε-DP.

Lemma 2.13 ([19]). *Randomized response is ε-DP.*

The *composition theorem* guarantees privacy for the *composition* of multiple algorithms with privacy guarantees of their own. In particular, this theorem covers the use case where multiple LEDP algorithms are used on the *same* dataset.

Theorem 2.14 (Composition Theorem [18, 19, 21]). *A sequence of DP algorithms, $(\mathcal{A}_1, \ldots, \mathcal{A}_k)$, with privacy parameters $(\varepsilon_1, \ldots, \varepsilon_k)$ form at worst an $(\varepsilon_1 + \cdots + \varepsilon_k)$-DP algorithm under* adaptive composition *(where the adversary can adaptively select algorithms after seeing the output of previous algorithms).*

Finally, the post-processing theorem states that the result of post-processing on the output of an ε-LEDP algorithm is ε-LEDP.

Theorem 2.15 (Post-Processing [8, 19]). *Let $\mathcal{M}$ be an ε-LEDP algorithm and h be an arbitrary (randomized) mapping from Range$(\mathcal{M})$ to an arbitrary set. The algorithm $h \circ \mathcal{M}$ is ε-LEDP.*[4]

We use implementations by the Google privacy team [76], which also guarantee cryptographic security.

2.2 Problem Definitions

Below, we define the k-core decomposition, low out-degree ordering, and triangle counting problems that we study.

In this paper, we consider undirected graphs $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. We use $[n]$ to denote $\{1, \ldots, n\}$. For ease of indexing, we set the IDs of V to be $V = [n]$. The set of neighbors of a node $i \in [n]$ is denoted by $N(i)$, and the degree of node i is denoted $\deg(i)$.

Definition 2.16 (k-Core Decomposition). *Given an input graph, $G = (V, E)$, a k-**core** is a maximal induced subgraph in G where every node has degree at least k. A k-**core decomposition** assigns a **core number** to each node $v \in V$ equal to κ if v is in the κ-core but not the $(\kappa + 1)$-core. Let $k(v)$ be the core number of v.*

See Fig. 2 for an example. No *exact* k-core decomposition algorithm satisfies the definition of DP (or LEDP). Hence, our algorithms take an input graph G and output an *approximate* core number for each node in the graph (Definition 2.17) and an approximate ***low out-degree ordering*** (Definition 2.19).

Definition 2.17 ((ϕ, ζ)-Approximate Core Number [15]). *Let $\hat{k}(v)$ be an approximation of the core number of v, and let $\phi \geq 1, \zeta \geq 0$. The core estimate $\hat{k}(v)$ is a (ϕ, ζ)-**approximate core number** of v if $k(v) - \zeta \leq \hat{k}(v) \leq \phi \cdot k(v) + \zeta$.*

[4] $\circ$ is notation for applying h on the outputs of $\mathcal{M}$.

We define the related concept of an ***approximate low out-degree ordering*** based on the definition of ***degeneracy***.

Definition 2.18 (Degeneracy). *An undirected graph $G = (V, E)$ is d-degenerate if every induced subgraph of G has a node with degree at most d. The* degeneracy *of G is the smallest value of d for which G is d-degenerate.*

It is well known that degeneracy $d = \max_{v \in V}\{k(v)\}$.

Definition 2.19 ((ϕ, ζ)-Approximate Low Out-Degree Ordering). *Let $D = [v_1, v_2, \ldots, v_n]$ be a total ordering of nodes in a graph $G = (V, E)$. The ordering D is an (ϕ, ζ)-**approximate low out-degree ordering** if orienting edges from earlier nodes to later nodes in D produces out-degree at most $\phi \cdot d + \zeta$.*

Definition 2.20 (Triangle Count). *Given an undirected input graph $G = (V, E)$, the triangle count returns the number of 3-cycles in G.*

2.3 Distributed Simulation Model

LEDP algorithms require that each node and its adjacency list be assigned to a separate machine or processor. However, given the infeasibility of this approach for large-scale graphs in real-world distributed systems, we simulate it using a coordinator-worker model. In our setup, each worker (machine) is assigned a subset of nodes along with their respective adjacency lists. Each worker independently runs LEDP algorithms on the private adjacency lists stored in its memory and communicates only LEDP, i.e privacy-preserving, outputs to a central coordinator (acting as a curator). The coordinator aggregates these outputs and publishes new public information for all workers, ensuring iterative updates over multiple synchronous communication rounds. We assume that each worker has sufficient storage capacity for the adjacency lists of its assigned nodes. This simulation provides a realistic evaluation of the computational and communication costs of LEDP algorithms in a distributed setting while ensuring strict privacy guarantees, as only LEDP outputs are communicated over the network.

3 PRACTICAL k-CORE DECOMPOSITION ALGORITHM

We present a novel k-core decomposition algorithm, k-CoreD, that builds upon the ϵ-LEDP algorithm proposed in [15]. The original algorithm, while theoretically robust, faces practical challenges such as a large number of communication rounds and significant additive error, both scaling with the total number of nodes in the graph. To overcome these issues, we introduce two key ideas: degree thresholding and bias terms. These techniques shift the dependency from the graph size to an input-dependent parameter, the maximum degree, significantly reducing the number of rounds and improving the additive error bounds.

3.1 Algorithm Description

The k-core decomposition algorithm operates synchronously over $O(\log(n) \log(D_{\max}))$ rounds, where $D_{\max}$ is the maximum degree of the graph. The algorithm outputs $\left(2 + \eta, O\left(\frac{\log(D_{\max}) \log^2(n)}{\epsilon}\right)\right)$-approximate core numbers with high probability, as well as a low out-degree ordering with the same approximation guarantee. Throughout this section, the term $\log(n)$ denotes $\log_{1+\psi}(n)$, unless explicitly stated otherwise (where ψ is a constant). The algorithm

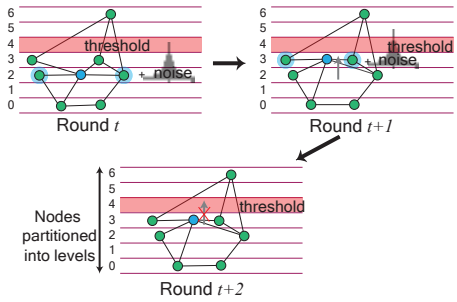


Figure 3: Node's (blue) movement across levels in the coordinator's level data structure, updated each round based on the noisy neighbor counts released by worker processes. The node doesn't move up a level in round *t+2*, due to thresholding, which upper-bounds the level for a node.

uses a level data structure (LDS) [15], where nodes are assigned levels that are updated iteratively. Levels are partitioned into groups of equal size, with each group g_i containing $\frac{\lceil \log(n) \rceil}{4}$ consecutive levels. We limit the number of rounds a node participates in, based on its noisy degree, which we refer to as degree thresholding. This significantly reduces the number of rounds, from $O(\log^2(n))$ [15] to $O(\log(n)\log(D_{\max}))$. In each round, the algorithm processes nodes to determine whether they should move up a level in the LDS, based on a noisy count of its neighbors at the same level. After processing all nodes in a round, the updated LDS is published for use in subsequent rounds. Once all rounds are complete, the algorithm estimates the core numbers of the nodes based on their final levels, using Algorithm 3.4. Additionally, a low out-degree ordering is determined by sorting nodes from smaller to larger levels, breaking ties using node IDs.[5]. The algorithm is implemented in a distributed setting, where computation is divided between a coordinator and multiple workers. The pseudocode is structured to reflect this division. The following parapgraphs describe the roles of the coordinator and the workers.

Coordinator As detailed in Algorithm 3.1, it receives the graph size n, the number of workers, M, constant parameter $\psi > 0$, privacy parameter $\varepsilon \in (0, 1]$, constant privacy split fraction $f \in (0, 1)$, and the bias term b as input. It computes privacy parameters ε_1 and ε_2 for degree thresholding and noisy neighbor count based on ε and f as shown in Line 5. The coordinator maintains the *level data structure* (*LDS*) and a communication channel, *channel*, for receiving data from the workers. All nodes start at level 0 as indicated in Line 7; the nodes then move up levels according to bits released by the corresponding workers (discussed below). $LDS[i]$ contains the current level of node i. The coordinator signals the workers to load their graphs (Line 10) in parallel, and uses the threshold values sent by the workers to compute the number of rounds (Line 11). In each round r, the coordinator calculates the group index of that round (Line 13), and launches M asynchronous worker processes (Line 15). Each worker sets a bit for each node in its subgraph to 1, if the node should move up a level, or 0 (if not) and sends this data back to the coordinator. The coordinator waits for all workers to complete (Line 16), processes incoming data, and moves nodes based on worker-computed bits. After each round, it publishes the updated *LDS* (Line 21) for use in the next round. Once all rounds are complete, it estimates the core numbers for the nodes using the LDS.

Worker (Degree Thresholding) As shown in Algorithm 3.2, workers initialize their respective subgraphs, and compute the threshold for a node by adding symmetric geometric noise to the original degree d_v of the node, producing $\widetilde{d_v} = d_v + X$, where $X \sim \text{Geom}(\frac{\epsilon}{2})$ (Line 6). An additional bias term is subtracted from $\widetilde{d_v}$ to account for large negative noise; this bias term is calculated using the variance of the noise distribution. The threshold value is then calculated as $\lceil \log_2(\widetilde{d_v}) \rceil \cdot L$, where L is the number of levels per group in the LDS (Line 8). Nodes store their thresholds to determine whether they should participate in a given round. Once all thresholds are computed, the worker identifies the maximum threshold value for its subgraph and sends this information back to the coordinator (Line 11).

Worker (Level Moving) In each round, a worker process determines for each node in its respective subgraph if it should move up a level in the LDS. As shown in Algorithm 3.3, for a node v, if the threshold matches the current round r, the node is skipped (Line 5). Otherwise, if the node's level is r, the algorithm calculates the number of neighbors at the same level (Line 9). This count is perturbed to ensure privacy, producing a noisy count $\widehat{\mathcal{U}_v} = \mathcal{U}_v + X + B$ (Line 13), where $\mathcal{U}_v$ is the original count, X is noise drawn from a symmetric geometric distribution with parameter $s = \frac{\epsilon}{2 \cdot (v.threshold)}$, and B is a bias term introduced to optimize error bounds in practice, calculated using the variance of the noise distribution. A node moves up a level if and only if $\widehat{\mathcal{U}_v} > (1 + \eta/5)^{\mathcal{F}(r)}$, where $\mathcal{F}(r)$ is the group index of the current level (Line 15). After processing all nodes, the worker sends this information back to the coordinator (Line 18).

Bias Terms. We introduce two different bias terms, one in the degree thresholding procedure, and one in the level moving procedure. For the bias terms, we use the approximate standard deviation of the symmetric geometric distribution. The first bias term is subtracted from the computed threshold to account for situations where a large positive noise is chosen. If a large positive noise is chosen, we lose privacy proportion to the new threshold in the level moving step. Hence, our bias term biases the result to smaller thresholds, resulting in less privacy loss in the level moving step.

The second bias term is added to the computed induced noisy degree to account for situations where a large negative noise prevents nodes from moving up the first few levels of the structure. Our added bias allows nodes with non-zero degrees to move up the first levels of the structure. Since the degree bounds increase exponentially, this additional bias term accounts for smaller errors as nodes move up levels. Our bias terms are derived from theoretical analysis optimizing the trade-off between utility, privacy, and computational cost, rather than being manually tuned hyperparameters.

3.2 Theoretical Analysis

Memory Analysis & Communication Cost. Let M be the number of workers and n the graph size. Each worker processes S nodes, where $S = \lfloor n/M \rfloor$ for $M - 1$ workers, and the last worker handles $n-(M-1)\lfloor n/M \rfloor$. The coordinator maintains the level data structure (LDS) and a communication channel, both requiring $O(n)$ space, resulting in a total memory usage of $O(n)$. Each worker processes

[5]This doesn't leak privacy, as IDs are assigned to nodes and not edges and reveal no information about the sensitive edge data.

Algorithm 3.1: k-Core Decomposition and Ordering (Coordinator)

Input: graph size n; number of workers M; approx constant $\psi \in (0,1)$; privacy parameter $\varepsilon \in (0,1]$; split fraction $f \in (0,1)$; bias term b.
Output: Approximate core numbers and low out-degree ordering of each node in G.
Function k-CoreD$(n, \psi, \varepsilon, f, b)$
  Set $\lambda = \frac{(5-2\eta)\eta}{(\eta+5)^2}$; $L = \frac{\lceil \log n \rceil}{4}$
  Set $\varepsilon_1 = f \cdot \varepsilon$ and $\varepsilon_2 = (1-f) \cdot \varepsilon$
  Set $C \leftarrow$ new Coordinator $(LDS, channel)$
  Coordinator initializes $C.LDS$ with $C.LDS[i] \leftarrow 0 \ \forall i \in [n]$.
  Set $maxDegreeThresholds \leftarrow [\,]$
  parfor $w = 1$ *to* M **do**
    $maxDegreeThresholds[w] =$ DegreeThresholdWorker(w, ε_1, L, b)
  Set $numOfRounds =$ $\min\left(4\log(n)\log(\widetilde{d}_{max}) - 1, \max(maxDegreeThresholds)\right)$
  for $r = 0$ *to* $numOfRounds$ **do**
    Set $\mathcal{F}(r) \leftarrow \lfloor \frac{r}{L} \rfloor$
    parfor $w = 1$ *to* M **do**
      LevelMovingWorker$(w, r, \varepsilon_2, \psi, \mathcal{F}(r), C.LDS)$
    C.wait() ▷ coordinator waits for workers to finish
    $nextLevels \leftarrow C.channel$
    for $i = 1$ *to* n **do**
      if $nextLevels[i] = 1$ **then**
        $C.LDS$.levelIncrease (i)
    Coordinator publishes updated $C.LDS$
  Coordinator calls $cores \leftarrow C$.EstimateCoreNumbers$(C.LDS, L, \lambda, \psi)$
  Coordinator produces D, a total order of all nodes, using levels from $C.LDS$ (from smaller to larger) breaking ties by node ID
  Return $(cores, D)$

Algorithm 3.2: Degree Thresholding (Worker)

Input: worker ID w; privacy parameter $\varepsilon \in (0,1]$; levels per group L; bias term b.
Function DegreeThresholdWorker(w, ε, L, b)
  Set $maxThreshold \leftarrow 0$
  for *node* $v :=$ *localGraph* **do**
    Sample $X \sim$ Geom$\left(\frac{\varepsilon}{2}\right)$
    $\widetilde{d}_v \leftarrow d_v + X$ ▷ noised degree
    $\widetilde{d}_v \leftarrow \widetilde{d}_v + 1 - \min\left(b \cdot \frac{2 \cdot e^{\varepsilon}}{e^{2\varepsilon}-1}, \widetilde{d}_v\right)$
    $v.threshold \leftarrow \left\lceil \log_2(\widetilde{d}_v) \right\rceil \cdot L$ ▷ thresholding
    $v.permZero \leftarrow 1$
    $maxThreshold =$ max $(maxThreshold, v.threshold)$
  w.send $(maxThreshold)$

$O(S)$ nodes, requiring $O(Sn)$ space for the graph and an additional $O(S)$ space for auxiliary structures, leading to a total of $O(Sn)$. In terms of communication, workers send one bit per node per round, incurring a per-worker cost of $O(S)$ and an overall round cost of $O(n)$. The coordinator receives and distributes the updated LDS, adding another $O(n)$ cost. Thus, the total communication overhead for the algorithm is $O\left(n \log(n) \log\left(D_{\max}\right)\right)$.

Algorithm 3.3: Level Moving (Worker)

Input: worker id w; round number r; privacy parameter $\varepsilon \in (0,1]$; constant ψ; group index $\mathcal{F}(r)$; pointer to the coordinator LDS.
Function LevelMovingWorker$(w, r, \varepsilon, \psi, \mathcal{F}(r), LDS)$
  Set $nextLevels \leftarrow [0, \ldots, 0]$
  for *node* $v :=$ *localGraph* **do**
    if $v.threshold = r$ **then**
      $v.permZero = 0$
    $vLevel \leftarrow LDS$.getLevel (v)
    if $vLevel = r$ *and* $v.permZero \neq 0$ **then**
      Let $\mathcal{U}_v$ be the number of neighbors $j \in \mathbf{a}_v$ where LDS.getLevel $(j) = r$.
      Set scale $s \leftarrow \frac{\varepsilon}{2 \cdot (v.\text{threshold})}$
      Sample $X \sim$ Geom(s).
      Set extra bias $B \leftarrow \frac{6e^s}{(e^{2s}-1)^3}$
      Compute $\widehat{\mathcal{U}_v} \leftarrow \mathcal{U}_v + X + B$.
      if $\widehat{\mathcal{U}_i} > (1+\eta/5)^{\mathcal{F}(r)}$ **then**
        $nextLevels[v] = 1$
      else
        $v.permZero = 0$
  w.send $(w, nextLevels)$
  w.done ()

Algorithm 3.4: Estimate Core Number (Coordinator) [47]

Function EstimateCoreNumbers(LDS, L, λ, η)
  for $i = 1$ *to* n **do**
    $\hat{k}(i) \leftarrow (2+\lambda)(1+\eta/5)^{\max\left(\left\lfloor \frac{LDS[i]+1}{L} \right\rfloor - 1, 0\right)}$.
  Return $\{(i, \hat{k}(i)) : i \in [n]\}$

Privacy Guarantees. Our privacy guarantees depend on the following procedures. First, we perform degree-based thresholding, which upper bounds the number of levels a node can move up. Second, we subtract and add bias terms to the results of our mechanisms. And finally, we scale our noise added in Line 10 of Algorithm 3.3 by the noisy threshold. We show that our algorithm can be implemented using local randomizers (Definition 2.3). Then, we show that the local randomizers have appropriate privacy parameters to satisfy ε-LEDP (Definition 2.4).

Lemma 3.1 (Degree Threshold Local Randomizer). *Our degree thresholding procedure run with privacy parameter ε' is a $(\varepsilon'/2)$-local randomizer.*

Proof. Our degree-thresholding procedure upper bounds the number of levels that we iterate through using the (private) degree of each node. Specifically, it adds symmetric geometric noise to the degree $\widetilde{d}_u = d_u + \text{Geom}(\varepsilon'/2)$ and then computes $\lceil \log_{1+\eta}(\widetilde{d}_u) \rceil \cdot L$, where L is the number of levels per group. The sensitivity of the degree of any node is 1 and by the privacy of the geometric mechanism (Lemma 2.11), the output $\widetilde{d}_u$ is $(\varepsilon'/2)$-DP. Then, producing the final level upper bound uses post-processing (Theorem 2.15) where privacy is preserved. Hence, our output is $(\varepsilon'/2)$-DP and the algorithm can be implemented as a $(\varepsilon'/2)$-local randomizer. □

Using Lemma 3.1, we prove Theorem 3.2.

Theorem 3.2. *Algorithm 3.1 is ε-LEDP.*

Proof. Our algorithm calls the local randomizers in Lemma 3.1 with $\varepsilon_1 = \varepsilon \cdot f$, where $f \in (0,1)$ is a fraction which splits some portion of the privacy budget, and then iterates through the levels one-by-one while adding noise to the induced degree of each node consisting of all neighbors of the node on the same or higher level. We showed in Lemma 3.1 that the degree thresholding procedure can be implemented as $(\varepsilon_1/2)$-local randomizers.

The key to our better error bounds is our upper bound on the number of levels we iterate through, bounded by our threshold. Since the thresholds are public outputs from the local randomizers, we can condition on these outputs. Let the threshold picked for node v be denoted as t_v. Then, we add symmetric geometric noise to the induced degree of the node (among the neighbors at or above v's current level) drawn from Geom $(\varepsilon_2/(2 \cdot t_v))$ where $\varepsilon_2 = \varepsilon \cdot (1-f)$. Conditioning on the public levels of each node, the sensitivity of the induced degree of any node is 1. By the privacy of the geometric mechanism, we obtain a $(\varepsilon_2/(2 \cdot t_v))$-local randomizer for v. By composition (Theorem 2.14) over at most t_v levels, the set of all local randomizers called on v, is $(\varepsilon_2/2)$-differentially private. For any edge, the sum of the privacy parameters of the set of all local randomizers called on the endpoints of the edge is $2 \cdot \varepsilon_1/2 + 2 \cdot \varepsilon_2/2 = \varepsilon_1 + \varepsilon_2 = f \cdot \varepsilon + (1-f) \cdot \varepsilon$. By Definition 2.4, this is ε-LEDP.

Finally, our bias terms, added or subtracted after applying the geometric mechanism, preserve privacy due to the post-processing invariance of differential privacy (Theorem 2.15). □

Approximation Guarantees. Our algorithm given in the previous section contains several changes that results in better theoretical bounds and optimizes the practical performance on real-world datasets. To prove our approximation factors, we first show Invariant 1 and Invariant 2 hold for our modified algorithm. $D_{\max}$ is the graph's max degree.

Invariant 1 (Degree Upper Bound [15]). *If node $i \in V_r$ (where V_r contains nodes in level r) and $r < 4\log^2 n - 1$, then i has at most $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} + \frac{c\log(D_{\max})\log^2(n)}{\varepsilon}$ neighbors in levels $\geq r$, with high probability, for constant $c > 0$.*

Invariant 2 (Degree Lower Bound [15]). *If node $i \in V_r$ (where V_r contains nodes in level r) and $r > 0$, then i has at least $(1+\eta/5)^{\lfloor (r-1)/(2\log n)\rfloor} - \frac{c\log(D_{\max})\log^2(n)}{\varepsilon}$ neighbors in levels $\geq r-1$, with high probability, for constant $c > 0$.*

There are two parts to our analysis: first, we prove that our degree thresholding procedure does not keep the node at too low of a level, with high probability; second, we show that our new procedure for moving up levels adds at most the noise used in [15] and not more. Together, these two arguments maintain the invariants.

Lemma 3.3. *Degree-thresholding satisfies Invariant 1.*

Proof. By Lemma 2.7, the noise we obtain for thresholding is upper bounded by $\frac{c'\ln n}{(\varepsilon_1/2)} = \frac{2c'\ln n}{f\cdot\varepsilon}$ with probability at least $1 - \frac{1}{n^{c'}}$. Thus, the noisy degree $\tilde{d_v}$ we obtain in **??** of **??** follows $\tilde{d_v} \geq d_v - \frac{2c'\ln n}{f\cdot\varepsilon}$ with probability at least $1 - \frac{1}{n^{c'}}$. Let r be the level we output as the threshold level. Then, we can compute the upper degree bound of this level to be at least $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} = (1+\eta/5)^{\log_{1+\eta/5}(\tilde{d_v})} \geq (1+\eta/5)^{\log_{1+\eta/5}\left(d_v - \left(\frac{2c'\ln n}{f\varepsilon}\right)\right)} = d_v - \frac{2c'\ln n}{f\varepsilon}$. The maximum induced degree of v on any level is at most d_v. Let $d_{v,r}$ be the induced degree of v on the thresholded level r, it must hold that $d_{v,r} \leq \left(d_v - \frac{2c'\ln n}{f\varepsilon}\right) + \frac{4c'\ln n}{f\varepsilon} + \frac{2c_3\ln n}{f\varepsilon} \leq (1+\eta/5)^{\lfloor r/(2\log n)\rfloor} + \frac{(4c'+2c_3)\ln n}{f\varepsilon}$. Since f and c' are both constants and Invariant 1 allows for picking a large enough constant $c > 0$, we can pick $c \geq 2(c'+c_3)/f$ and Invariant 1 is satisfied where c' and c_3 are fixed constants ≥ 1. □

We do not have to prove that our thresholded level satisfies Invariant 2 since we use the threshold level as an *upper bound* of the maximum level that a node can be on. Hence, a node will not reach that level unless the procedure for moving the node up the levels satisfies Invariant 2. We now prove that our level movement procedure satisfies both invariants.

Lemma 3.4. *Our level moving algorithm satisfies Invariant 1 and Invariant 2.*

Proof. Our level moving algorithm is similar to [15] except that we pick noise based on the threshold. Thus, by Lemma 2.7, our algorithm picks noise that is at most $\frac{2c_1\cdot t_v\ln n}{\varepsilon}$ with probability at least $1 - \frac{1}{n^{c_1}}$ where t_v is the released threshold for v and $c_1 \geq 1$ is a fixed constant. Also, by Lemma 2.7, it holds that $t_v \leq \log\left(D_{\max} + \frac{2c_2\ln n}{f\varepsilon} - \frac{2c_3\ln n}{f\varepsilon}\right)\cdot L \leq 2\log(D_{\max})\log n$ with probability at least $1 - \frac{1}{n^{c_2}}$. A node moves up a level from level r if its induced degree plus the noise exceeds the threshold $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor}$. Thus, using our computed noise, the degree of a node must be at least $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} - 2\log(D_{\max})\left(\frac{2c_1\log^2 n}{\varepsilon}\right)$ with probability at least $1 - \frac{1}{n^{c_1}} - \frac{1}{n^{c_2}}$ when it moves up from level r. By choosing large enough constants $c_1, c_2 > 0$ and $c \geq 4c_1c_2$, we satisfy Invariant 2. Similarly, the node does not move up from level r when its induced degree plus noise is at most $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor}$. By a symmetric argument to the above, we show that Invariant 1 is satisfied. □

Finally, using Lemma 3.3 and Lemma 3.4, we can prove the final approximation factor for our algorithm.

Theorem 3.5. *Our algorithm returns $(2+\eta, O(\log(D_{\max})\log^2(n)/\varepsilon))$-approximate k-core numbers, with high probability, in $O(\log(n)\log(D_{\max}))$ rounds of communication.*

Proof. By Lemma 3.3 and Lemma 3.4, our algorithm satisfies Invariant 1 and Invariant 2, hence, our algorithm returns our desired approximation using Theorem 4.1 of [15]. Our algorithm iterates to value at most $O(\log n\log(D_{\max}))$ for $D_{\max}$ (the maximum threshold), resulting in a total of $O(\log n\log(D_{\max}))$ rounds. □

4 TRIANGLE COUNTING USING LOW OUT-DEGREE ORDERING

We present our novel triangle counting algorithm, EdgeOrient$_\Delta$, which leverages the low out-degree ordering obtained from the k-CoreD algorithm along with randomized response (RR). While previous approaches rely on all neighboring edges after applying RR or on sampling subsets of edges, these methods often suffer from error bounds that scale poorly with the graph size. To address these limitations, our algorithm leverages a key observation: the number

of oriented 4-cycles (Fig. 5) for a node is upper bounded by n^2d^2, where d is the maximum core number (degeneracy). By utilizing this bound, the algorithm exploits a new input-dependent graph property, the degeneracy, to significantly improve error bounds in theory and practice.

Algorithm 4.1: ε-LEDP Triangle Counting (Coordinator)

1 **Input:** graph size n; number of workers M; constant $\psi \in (0,1)$; privacy parameter $\varepsilon \in (0,1]$; split fraction $f \in (0,1)$; bias term b;
2 **Output:** Noisy Triangle Count
3 **Function** $\texttt{EdgeOrient}_{\Delta}(n, M, \psi, \varepsilon, f, b)$
4 Set $Z \leftarrow k\text{-}\texttt{CoreD}(n, \psi, \frac{\varepsilon}{4}, f, b)$ (Algorithm 3.1)
5 Set $C \leftarrow$ new Coordinator $(cRR, cTCount, cMaxOut, X)$
6 ▷ Round 1: Randomized Response
7 **parfor** $w = 1$ *to* M **do**
8 $\texttt{RRWorker}\left(w, n, \frac{\varepsilon}{4}\right)$
9 C.wait() ▷ coordinator waits for workers to finish
10 C.publishNoisyEdges(cRR, X)
11 ▷ Round 2: Max Out-degree
12 **parfor** $w = 1$ *to* M **do**
13 $\texttt{MaxOutDegreeWorker}\left(w, \frac{\varepsilon}{4}, Z\right)$
14 C.wait()
15 $\widetilde{d}_{\max} \leftarrow \max\left(\{C.cMaxOut[i] \mid i \in [M]\}\right) + \frac{12\log(n)}{\varepsilon}$
16 ▷ Round 3: Count Triangles
17 **parfor** $w = 1$ *to* M **do**
18 $\texttt{CountTrianglesWorker}\left(w, \frac{\varepsilon}{4}, Z, X, \widetilde{d}_{\max}\right)$
19 C.wait()
20 $\widetilde{\Delta} \leftarrow \sum_{i=1}^{n} C.cTCount[i]$
21 **Return** $\widetilde{\Delta}$

Algorithm 4.2: Randomized Response (Worker)

1 **Input:** worker id w; graph size n; privacy parameter $\varepsilon \in (0,1]$;
2 **Function** $\texttt{RRWorker}(w, n, \varepsilon)$
3 Set $neighborsRR \leftarrow [][]$
4 **for** *node* $v :=$ *localGraph* **do**
5 For all $ngh_v \leftarrow \{j : j \in [n] \wedge j > v\}$
6 $neighborsRR[v] = \text{RandomizedResponse}_{\varepsilon}(ngh_v)$
7 w.sendRR(w, $neighborsRR$)
8 w.done ()

Algorithm 4.3: Noisy Max Out-Degree (Worker)

1 **Input:** worker id w; graph size n; privacy parameter $\varepsilon \in (0,1]$;
2 **Function** $\texttt{MaxOutDegreeWorker}(w, \varepsilon, Z)$
3 Set $out_{\max} \leftarrow 0$.
4 **for** *node, adjacency list* $v, ngh :=$ *localGraph* **do**
5 Set $ngh_v \leftarrow \{j : j \in ngh \wedge Z[j] > Z[v]\}$
6 $B \sim \text{Geom}(\varepsilon)$
7 $out_{\max} \leftarrow \max(out_{\max}, |ngh_v| + B)$
8 w.sendMaxOutdegree$(out_{\max})$
9 w.done ()

4.1 Algorithm Description

The triangle counting algorithm consists of three additional computation rounds after the low out-degree ordering (Z) is computed using k-CoreD. In the first round, each node perturbs its adjacency list of each node using Randomized Response (RR) [82], producing a privacy-preserving set of noisy edges for subsequent computations. In the second round, we calculate the maximum noisy out-degree, $\widetilde{d}_{\max}$, by determining each node's outgoing edges based on Z. While the first two rounds can be combined, we separate them for clarity. In the final round, we compute the number of triangles incident to each node, using the noisy edges and the maximum noisy out-degree, $\widetilde{d}_{\max}$. The algorithm is implemented in a distributed setting, where computation is divided between a coordinator and multiple workers. The following sections describe this division, present the pseudocode and expand on each round of computation, and detail the roles of the coordinator and workers.

Algorithm 4.4: Triangle Counting (Worker)

1 **Input:** worker id w; privacy parameter $\varepsilon \in (0,1]$; low out-degree ordering Z, published noisy edges X; public maximum noisy out-degree $\widetilde{d}_{\max}$;
2 **Function** $\texttt{CountTrianglesWorker}(w, \varepsilon, Z, X, \widetilde{d}_{\max})$
3 Set $workerTCount \leftarrow 0.0$
4 **for** *node, adjacency list* $v, ngh :=$ *localGraph* **do**
5 Set $\widetilde{\Delta} \leftarrow 0.0$
6 $\text{OutEdges}_v = \{j : j \in ngh \wedge Z[j] > Z[v]\}$
7 **for** $i_1 \in \{1, \ldots, \min(\widetilde{d}_{\max}, |OutEdges_v|)\}$ **do**
8 **for** $i_2 \in \{i_1 + 1, \ldots, \min(\widetilde{d}_{\max}, |OutEdges_v|)\}$ **do**
9 $j \leftarrow \text{OutEdges}_v[i_1]$
10 $k \leftarrow \text{OutEdges}_v[i_2]$
11 $\widetilde{\Delta} \leftarrow \widetilde{\Delta} + \frac{X_{\{j,k\}} \cdot (e^{\varepsilon}+1)-1}{e^{\varepsilon}-1}$
12 Sample $R \sim \text{Lap}\left(\frac{\varepsilon}{2 \cdot \widetilde{d}_{\max}}\right)$
13 $\widetilde{\Delta} \leftarrow \widetilde{\Delta} + R$
14 $workerTCount \leftarrow workerTCount + \widetilde{\Delta}$
15 w.sendTCount (w, $workerTCount$)
16 w.done ()

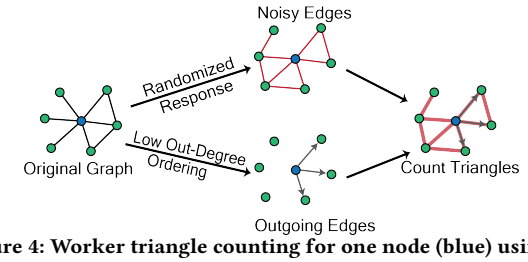


Figure 4: Worker triangle counting for one node (blue) using Randomized Response and low out-degree ordering.

Coordinator As shown in Algorithm 4.1, the coordinator receives the graph size n, number of workers M, constant parameter $\psi > 0$, privacy parameter $\varepsilon \in (0,1]$, privacy split fraction $f \in (0,1)$, and bias term b. It initializes three channels, $cRR, cMaxOut, cTCount$, to receive Randomized Response noisy edges, maximum noisy out-degrees, and local noisy triangle counts from workers (Line 5). It also initializes X to store noisy edges for the entire graph. The coordinator manages the algorithm's execution, collecting worker outputs and publishing updates each round. It first computes the low out-degree ordering, Z, using k-CoreD. In the first round, it launches M asynchronous worker processes (Line 6), each computing and sending noisy edges after Randomized Response. Before the second round, the coordinator aggregates and stores them in

X, then publishes X for global access (Line 10), enabling workers to utilize the noisy public edges in subsequent computations. In the second round (Line 11), workers compute noisy maximum out-degrees for their subgraphs using Algorithm 4.3. The coordinator then determines $\widetilde{d}_{\max}$, the maximum noisy out-degree across all workers (Line 15). Finally, in the third round (Line 16), each worker counts the triangles incident to its nodes using the low out-degree ordering, published noisy edges, and $\widetilde{d}_{\max}$. Workers send noisy local triangle counts to the coordinator, which aggregates them to compute the overall noisy triangle count (Line 20).

Worker (Randomized Response) As specified in Algorithm 4.2, workers maintain a *neighborsRR* data structure to store noisy edges. For each node v, noisy edges are computed via Randomized Response (RR) with parameter ε, processing only the upper triangular part of the adjacency matrix (Line 5), as the graph is undirected. Specifically, for a node v, all indices greater than v are processed using $\text{RandomizedResponse}_{\varepsilon}(ngh_v)$, which flips the existence of each edge (v, ngh_v) with probability $\frac{1}{e^{\varepsilon}+1}$ (Line 6). Once computed, workers send noisy edges to the coordinator (Line 7).

Worker (Noisy Max Out-Degree) In Algorithm 4.3, workers maintain a variable out_{max} which stores the maximum noisy out-degree of any node in their subgraph. For each node v, the worker first computes the out-degree d_v using the order provided in Z, where an edge (v, j) is considered outgoing if $Z[j] > Z[v]$ (Line 5). The out-degree d_v is the number of outgoing edges from v. Then, the worker adds symmetric geometric noise with parameter ε to d_v. Finally, the worker maintains the maximum noisy out-degree computed in this way in $out_{\max}$ and sends $out_{\max}$ back to the coordinator (Line 8).

Worker (Count Triangles) The pseudocode for our algorithm is outlined in Algorithm 4.4. Each worker computes the number of triangles incident to each node in its respective subgraph. For each node v, the outgoing edges are identified and sorted in ascending order by node IDs. The triangle count is determined by iterating through all unique pairs of outgoing neighbors $\{j, k\}$ of v, up to $\widetilde{d}_{\max}$ (Line 7,8). For each pair, the triangle contribution is calculated using the formula: $\frac{X_{\{j,k\}} \cdot (e^{\varepsilon}+1)-1}{e^{\varepsilon}-1}$, where $X_{\{j,k\}}$ represents the noisy presence (1) or absence (0) of an edge between j and k (Line 11). To ensure privacy, additional noise is added to the triangle counts using the Laplace distribution with parameter $\frac{\varepsilon}{2 \cdot \widetilde{d}_{\max}}$, where $\widetilde{d}_{\max}$ is the global maximum noisy out-degree. We use Laplace noise here because it offers a smoother tradeoff for smaller parameters. Upon completing the computation, each worker aggregates and returns the noisy triangle count for its entire subgraph (Line 15).

4.2 Theoretical Analysis

Memory Analysis & Communication Cost. Let M be the number of workers and n the graph size. Each worker processes S nodes, where $S = \lfloor n/M \rfloor$ for $M-1$ workers, and the last worker handles $n - (M-1)\lfloor n/M \rfloor$ nodes. The coordinator manages three communication channels and publishes the noisy edges for the entire graph. The *cRR* structure, which aggregates noisy edges, requires $O(n^2)$ space, while *cTCount* and *cMaxOut*, which collect triangle counts and maximum noisy out-degree, require $O(M)$ space each. Storing published noisy edges further adds $O(n^2)$ space, resulting in a total coordinator memory requirement of $O(n^2 + M)$. Each worker processes $O(S)$ nodes, requiring $O(Sn)$ space for the graph. The *neighborsRR* structure for storing noisy edges demands $O(Sn)$ space, while computing the maximum noisy out-degree requires $O(S)$. The final triangle count computation takes $O(S \cdot \widetilde{d}_{\max})$ space, where $\widetilde{d}_{\max}$ is the maximum noisy out-degree, leading to an overall worker memory requirement of $O(Sn)$. The algorithm runs three communication rounds beyond those for low out-degree ordering. Workers first send noisy edges, incurring $O(Sn)$ communication cost, followed by sending the maximum noisy out-degree and triangle counts, each requiring $O(M)$ communication. Thus, the total communication overhead for the algorithm is $O(n^2 + M)$.

Privacy Guarantees. As before, our privacy guarantees are proven by implementing our triangle counting algorithm using local randomizers.

Lemma 4.1. *Our triangle counting algorithm is ε-LEDP.*

Proof. Our triangle counting algorithm calls Algorithm 3.1, which by Theorem 3.2 is $(\varepsilon/4)$-LEDP. Additionally, we release three sets of information, each of which we show to be $(\varepsilon/4)$-LEDP.

First, each node applies Randomized Response to the upper triangular adjacency matrix to generate a privacy-preserving set of edges. By [19], this adjacency list output is a $(\varepsilon/4)$-local randomizer.

Second, each node releases its privacy-preserving out-degree. By Line 7, the sensitivity of the out-degree (conditioning on Z) is 1 for neighboring adjacency lists. By the privacy of the geometric mechanism ([4, 9, 19, 20]), each node uses a $(\varepsilon/4)$-local randomizer to output its noisy degree.

Third, each node releases a privacy-preserving triangle count using its outgoing edges from the low out-degree ordering. To bound the sensitivity, we truncate the outgoing adjacency list (computed using Z) of each node by $\widetilde{d}_{\max}$. Given neighboring adjacency lists $\mathbf{a}$ and $\mathbf{a}'$, assume $\mathbf{a}'$ contains one additional neighbor w (without loss of generality). Let $\overline{\mathbf{a}}$ and $\overline{\mathbf{a}'}$ be the truncated adjacency lists. In the worst case, $\overline{\mathbf{a}}$ contains a node u not in $\overline{\mathbf{a}'}$, while $\overline{\mathbf{a}'}$ contains w (not in $\overline{\mathbf{a}}$). Let j be defined as in Line 9. If $j = u$, the first for-loop (Line 7) counts at most $\widetilde{d}_{\max}$ additional triangles for u (symmetrically for w). Assuming u returns $\widetilde{d}_{\max}$ triangles and w returns none, then for all other nodes $j \neq u$, the second for-loop (Line 8) encounters at most $\widetilde{d}_{\max}$ additional triangles. Thus, the total difference in counted triangles between $\overline{\mathbf{a}}$ and $\overline{\mathbf{a}'}$ is $2\widetilde{d}_{\max}$, giving a sensitivity of $2\widetilde{d}_{\max}$. By the privacy of the Laplace mechanism ([19]), outputting local triangle counts is an $(\varepsilon/4)$-local randomizer.

Since the differing edge between neighboring graphs G and G' affects at most one node's out-degree, applying composition ([18, 19, 21]) over all four $(\varepsilon/4)$-local randomizers results in an ε-LEDP triangle counting algorithm. □

Approximation Guarantees. One of the major novelties in our proofs is via a new intricate use of the Law of Total Expectation and Law of Total Variance for the events where the out-degrees of each node is upper bounded by the noisy maximum out-degree $\widetilde{d}_{\max}$ (which is, in turn, upper bounded by the degeneracy $\widetilde{O}(d)$). Such use cases were unnecessarily in [22, 38] because they did not use oriented edges.

Figure 5: Oriented cycle of length 4; two non-adjacent black nodes have edges oriented toward the remaining red nodes.

Lemma 4.2. *In expectation, our algorithm returns a 2-approximation of the true triangle count:* $\frac{(n^3-1)\cdot T}{n^3} \leq \mathbb{E}[\widetilde{\Delta}] \leq T$.

Proof. We first prove that, in expectation, $\mathbb{E}\left[\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right] = \mathbb{1}_{j,k}$ where $\mathbb{1}_{j,k}$ is the indicator variable for whether edge $\{j,k\}$ exists in the original private graph. Since randomized response flips the bit indicating the existence of an edge between each pair of vertices with probability $\frac{1}{e^{\varepsilon/4}-1}$, we can simplify the expression to be:

$$\mathbb{E}\left[\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right] = \frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}.$$

When $\mathbb{1}_{j,k}=1$, the probability that we obtain a bit of 1 is $\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}$ and our expression simplifies to

$$\frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = \frac{\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = 1.$$

When $\mathbb{1}_{j,k}=0$, the probability that we obtain a bit of 1 is $\frac{1}{e^{\varepsilon/4}+1}$ then our expression simplifies to

$$\frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = \frac{\frac{1}{e^{\varepsilon/4}+1}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = 0.$$

The expected value of the random variable obtained for each edge e using our randomized response procedure is equal to $\mathbb{1}_e$.

Now, we condition on the event that $\widetilde{d}_{\max}$ upper bounds the out-degree of every vertex after computing and conditioning on D (the ordering). Let U be this event.

Then, for each node, we compute all pairs of its outgoing neighbors and use the random variable indicating existence of the edge spanned by the endpoints to count every triangle composed of a pair of outgoing edges. Let $T_{v,j,k}$ be the random variable representing the existence of the queried triangle for node v and outgoing edges (v,j) and (v,k). Then, $\mathbb{E}[T_{v,j,k} \mid U] = \mathbb{E}[X_{j,k}]$ since both outgoing edges (v,j) and (v,k) exists. By what we showed above, it then follows that $\mathbb{E}[T_{v,j,k}] = \mathbb{1}_{j,k}$.

We must account for the symmetric geometric noise. Since the expectation of the symmetric geometric noise is 0 and we add together all of the values for each of the drawn noises, the expected total noise is 0 by linearity of expectations. Each triangle has a unique node that queries it since every triangle has a unique orientation of edges where two edges are outgoing from a vertex in the triangle. Hence, the expected sum of all queried triangles is T, conditioned on U, by linearity of expectations.

Finally, we remove the conditioning on the event U. Recall that $\mathbb{E}[W] = \sum_y \mathbb{E}[W \mid Y=y]\cdot\Pr(Y=y)$. Since truncation can only decrease the number of queried triangles (and hence the expectation), we upper and lower bound $0 \leq \mathbb{E}[\widetilde{\Delta} \mid \neg U] \leq T$. Hence we only need to figure out the probability $P(U)$ to upper and lower bound $\mathbb{E}[\widetilde{\Delta}]$. The probability of event U is the probability that $\max(\{d_v + \text{Geom}(\varepsilon/4) \mid v \in V\}) + \frac{\log(n)}{\varepsilon} \geq \max(\{d_v \mid v \in V\})$ where d_v is the out-degree of node v given order D. This probability is lower bounded by the probability that $\max(\{d_v \mid v \in V\}) + \text{Geom}(\varepsilon/4) + \frac{c\log(n)}{\varepsilon} \geq \max(\{d_v \mid v \in V\})$ for a fixed constant $c \geq 1$; this means we want to lower bound $\Pr\left(\left(Q + \frac{c\log(n)}{\varepsilon}\right) \geq 0\right)$ where $Q \sim \text{Geom}(\varepsilon/4)$. By concentration of random variables chosen from the symmetrical geometric distribution, we know that $\Pr\left(\left(Q + \frac{c\log(n)}{\varepsilon}\right) \geq 0\right) \geq 1 - \frac{1}{n^3}$ for large enough constant $c \geq 1$. Specifically, setting $c = 3$ gives us this bound. Hence, we can upper and lower bound $\frac{(n^3-1)\cdot T}{n^3} \leq \mathbb{E}[\widetilde{\Delta}] \leq T$. □

To calculate the variance of the triangle count obtained from our algorithm, we use a quantity denoted by $\overrightarrow{C}_4$ which is the number of oriented 4-cycles where there exists two non-adjacent nodes with outgoing edges to the other two nodes in the cycle. See Fig. 5 for an example. The number of oriented cycles of length 4 is upper bounded by n^2d^2 where d is the degeneracy (maximum core number) in the graph, resulting in significant gains in utility over previous results which use the number of total (not oriented) 4-cycles which could be as large as $\Omega(n^4)$.

Lemma 4.3. *Our triangle counting algorithm returns a count with variance* $O\left(\frac{nd^2\log^6 n}{\varepsilon^4} + \overrightarrow{C}_4\right)$ *where d is the degeneracy (max core number) of the graph,* $\overrightarrow{C}_4$ *is the number of oriented cycles of length 4, and T is the number of (true) triangles in the private graph.*

Proof. As before, we first calculate the variance conditioned on the event U that $\widetilde{d}_{\max}$ upper bounds the out-degrees of every node. Then, we use the law of total variance to remove this condition.

For notation simplicity, we omit the condition on U from the right hand sides of the below equations. First, the variance of $\text{Var}[X_{\{i,j\}}] = \frac{e^{\varepsilon}}{(e^{\varepsilon}+1)^2}$, since $X_{\{i,j\}}$ is a Bernoulli variable. Then, let $\hat{T}$ be our returned triangle count. The variance

$$\text{Var}\left[\hat{T} \mid U\right] = \text{Var}\left[\sum_{v\in[n]}\left(\sum_{j,k\in Out(v)}\left(\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right) + \text{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right)\right].$$

Recall $\widetilde{d}_{\max}$ is our noisy maximum out-degree of any vertex. Then, our variance simplifies to

$$\left(\frac{e^{\varepsilon/4}+1}{e^{\varepsilon/4}-1}\right)^2 \cdot \text{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right] + \text{Var}\left[\sum_{v\in[n]}\text{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right].$$

By the variance of the Laplace distribution, we can compute

$$\text{Var}\left[\sum_{v\in[n]}\text{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right] \leq n\cdot\frac{8\widetilde{d}_{\max}^2}{\varepsilon^2}.$$

Now, it remains to compute $\text{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right]$. The covariance is 0 if two queried pairs do not query the same $X_{j,k}$. The covariance is non-zero only in the case of queries which overlap in $X_{i,j}$. This occurs only in the case of an oriented 4-cycle where $X_{i,j}$ is shared between two queries. Let P_2 be the set of all pairs of such queries that share $X_{j,k}$. In this case, the covariance is upper

bounded by $\mathbb{E}[X_{j,k}^2]$. Hence, we can simplify our expression to be

$$\text{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right] \tag{1}$$

$$\leq \sum_{v\in[n]}\sum_{j,k\in Out(v)} \text{Var}\left[X_{j,k}\right] + 2\cdot \sum_{T_{v,j,k},T_{w,j,k}\in P_2}\left(\mathbb{E}[X_{j,k}^2]\right) \tag{2}$$

$$\leq n\cdot \widetilde{d}_{\max}^2\cdot\frac{e^{\varepsilon/4}}{(e^{\varepsilon/4}+1)^2} + \overrightarrow{C}_4\cdot\left(\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}\right), \tag{3}$$

where $\overrightarrow{C}_4$ indicates the number of directed cycle of length 4 (Fig. 5).

Hence, our total variance is upper bounded by

$$\frac{8n\widetilde{d}_{\max}^2}{\varepsilon^2} + n\cdot\widetilde{d}_{\max}^2\cdot\frac{e^{\varepsilon/4}}{(e^{\varepsilon/4}+1)^2} + \overrightarrow{C}_4\cdot\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}.$$

Finally, by the guarantees of our k-core decomposition algorithm, the maximum out-degree $d_{\max}$ is bounded by $d_{\max}\leq(2+\eta)\cdot d + O\left(\frac{\log(D_{\max})\log^2 n}{\varepsilon}\right)$, with high probability, where d is the degeneracy of the input graph and $D_{\max}\leq n$ is the maximum degree in the graph. Finally, we also know that the noise generated for $\widetilde{d}_{\max}$ is upper bounded by $O\left(\frac{\log n}{\varepsilon}\right)$. Thus, $\widetilde{d}_{\max}\leq(2+\eta)\cdot d + O\left(\frac{\log^3 n}{\varepsilon}\right)$ Hence, our variance is upper bounded by $O\left(\frac{nd^2\log^6 n}{\varepsilon^4}+\overrightarrow{C}_4\right)$.

We now remove our condition on U and use the law of total variance to compute our unconditional variance. Recall the law of total variance states $\text{Var}[Y] = \mathbb{E}[\text{Var}[Y\mid X]] + \text{Var}[\mathbb{E}[Y\mid X]]$. We computed $\text{Var}\left[\hat{T}\mid U\right]$ above. We now compute $\text{Var}\left[\hat{T}\mid\neg U\right]$. The main difference between when the event U occurs and does not occur is that some adjacency lists of the outgoing neighbors will be truncated. Consequently, we sum over fewer $X_{j,k}$ variables in Eqs. (1) and (2). Thus, the variance when U does not occur is upper bounded by the variance when U does occur.

Now we calculate $\text{Var}\left[\mathbb{E}[\hat{T}\mid U]\right] = \mathbb{E}[\mathbb{E}[\hat{T}\mid U]^2] - \mathbb{E}[\mathbb{E}[\hat{T}\mid U]]^2$. By our calculation in the proof of Lemma 4.2, we can calculate $\mathbb{E}[\hat{T}\mid U] = T$ and let $\frac{(n^3-1)\cdot T}{n^3}\leq Y = \mathbb{E}[\hat{T}\mid\neg U]\leq T$. Then,

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[\hat{T}\mid U]^2] - \mathbb{E}[\mathbb{E}[\hat{T}\mid U]]^2 &= \left(\frac{T^2}{2}+\frac{Y^2}{2}\right) - \left(\frac{T}{2}+\frac{Y}{2}\right)^2\\
&= \frac{T^2+Y^2}{2} - \frac{T^2+2TY+Y^2}{4}\\
&= \frac{T^2+Y^2-2TY}{4}\\
&= \left(\frac{T-Y}{2}\right)^2\\
&\leq \left(\frac{T-\frac{(n^3-1)\cdot T}{n^3}}{2}\right)^2\\
&\leq \left(\frac{T}{2n^3}\right)^2\\
&\leq \frac{1}{4}.
\end{aligned}$$

Thus, our final variance is $O\left(\frac{nd^2\log^6 n}{\varepsilon^4}+\overrightarrow{C}_4\right)$. □

Theorem 4.4. *With high constant probability, our triangle counting algorithm returns a* $\left(1+\eta, O\left(\frac{\sqrt{n}d\log^3 n}{\varepsilon^2}+\sqrt{\overrightarrow{C}_4}\right)\right)$*-approximation of the true triangle count.*

Table 1: Graph size, maximum core number, and number of triangles.

Graph Name	Num. Vertices	Num. Edges	Max. Degree	Max. Core Num. (d)	Num. Triangles
email-eu-core	986	1,329,336	345	34	105,461
wiki	7115	100,761	1065	35	608,387
enron	36,692	183,830	1,383	43	727,044
brightkite	58,228	214,078	1,134	52	494,728
ego-twitter	81,306	1,342,296	3,383	96	13,082,506
gplus	107,614	12,238,285	20,127	752	1,073,677,742
stanford	281,903	1,992,635	38,625	71	11,329,473
dblp	317,080	1,049,866	343	113	2,224,385
brain	784,262	267,844,669	21,743	1200	–
orkut	3,072,441	117,185,083	33,313	253	–
livejournal	4,846,609	42,851,237	20,333	372	–
twitter	41,652,230	1,202,513,046	2,997,487	2488	–
friendster	65,608,366	1,806,067,135	5214	304	–

Proof. We use Chebyshev's inequality with the standard deviation calculated from Lemma 4.3. The $(1+\eta)$-approximation comes from our $\left(1-\frac{1}{n^3}\right)$-approximation of the expectation. □

See Table 1 for the d values of real-world graphs; when $d = O(1)$ is constant, as is the case for real-world graphs, then $\overrightarrow{C}_4 = O(n^2)$ and $T = O(n)$. We improve previous theoretical additive errors from $O\left(\frac{\sqrt{C_4}}{\varepsilon}+\frac{n^{3/2}}{\varepsilon^2}\right)$ [38] to $O\left(\sqrt{\overrightarrow{C}_4}+\frac{\sqrt{n}\log^3 n}{\varepsilon^2}\right)$, an improvement of at least a $\Omega(\sqrt{n})$ factor, translating to massive practical gains.

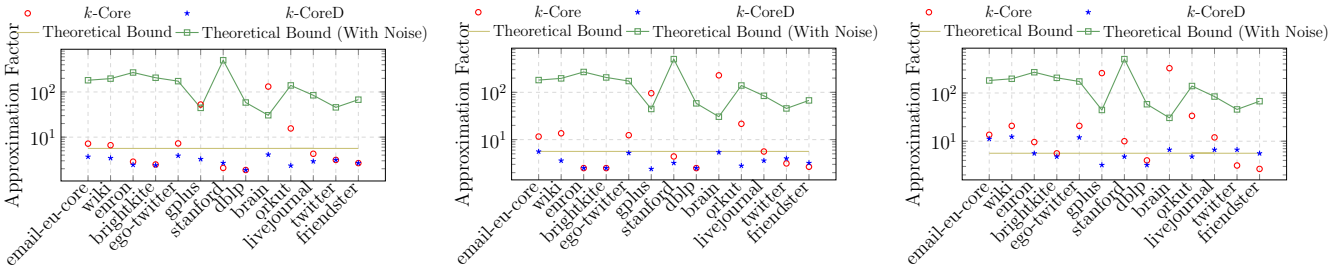
5 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance and accuracy of our k-core decomposition (k-CoreD) and triangle counting (EdgeOrient$_\Delta$) algorithms in a distributed simulation, benchmarking runtime and accuracy. For k-core decomposition, we compare our algorithm with the LEDP k-**Core** decomposition algorithm of [15], which we implement. For triangle counting, we also compare against **ARROneNS**$_\Delta$ (**Lap**) [38] and **GroupRR** [33]. Our results show that our algorithms scale to graphs with over a billion edges while consistently achieving significantly better approximations than theoretical bounds and prior work.

For k-core decomposition, our algorithm reduces the number of rounds by nearly ***two orders of magnitude*** over [15] across all graphs while improving accuracy. For triangle counting, our approach improves multiplicative approximation accuracy by nearly ***six orders of magnitude*** over previous methods and achieves speed-ups on large graphs.

Experimental Setup To evaluate our algorithms in a distributed simulation environment, we partition the input graph across M **worker processors** and a **single coordinator processor**. Each worker is responsible for a subset of the nodes and their complete adjacency lists and run LEDP algorithms locally while preserving privacy. The workers communicate their LEDP outputs to the coordinator, which aggregates the data and publishes new public information to all workers. This process is executed over multiple synchronous rounds of communication, simulating a real-world distributed environment. Unless otherwise specified, our experimental setup consists of **80 worker processors** and **a single coordinator**. Each reported value is the mean across **5 runs**, with a **4-hour** wall-clock limit per run.

Parameters We use the following values for the experiments $\varepsilon = 1.0$; bias term $= 8$, the approximation factor $(2+\eta)$ set to be 5.625 (the same approximation factor as used in non-private k-core decomposition experiments [47]), and privacy split fraction $f = 0.8$ where we use $0.8\cdot\varepsilon$ for thresholding and $0.2\cdot\varepsilon$ for adding



(a) Average Approximation Factor (b) 80^{th} Percentile Approximation Factor (c) 95^{th} Percentile Approximation Factor

Figure 6: k-core Decomposition Results.

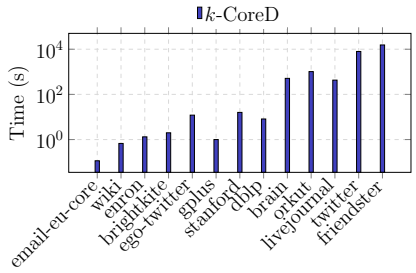


Figure 7: k-Core Decomposition Avg. Response Time

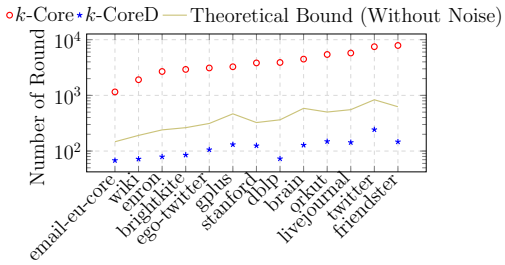


Figure 8: k-Core Decomposition Number of rounds

noise to the neighbor count. Additionally, we conduct an ablation study to analyze the impact of key parameters, ε, f, on the utility of our algorithms. Our theoretical proofs show that our approximation falls within a $(2+\eta)$-multiplicative factor.

Compute Resources We run experiments on a Google Cloud `c3-standard-176` instance (3.3 GHz Intel Sapphire Rapids CPUs, 88 physical cores, 704 GiB RAM) with hyper-threading disabled. The code, implemented in Golang [17], is publicly available [1].

Datasets We test our algorithms on a diverse set of 13 real-world undirected graphs from SNAP [44], the DIMACS Shortest Paths Challenge road networks [12], and the Network Repository [65], namely ***email-eu-core***, ***wiki***, ***enron***, ***brightkite***, ***ego-twitter***, ***gplus***, ***stanford***, ***dblp***, ***brain***, ***orkut***, ***livejournal***, and ***friendster***. We also used ***twitter***, a symmetrized version of the Twitter network [43]. *Brain* is a highly dense human brain network from NeuroData (https://neurodata.io/). We remove duplicate edges, zero-degree vertices, and self-loops. Table 1 reflects the graph sizes *after* this removal and gives the exact maximum core number and the number of triangles. Exact triangle counts for some graphs are omitted due to time or memory constraints.

5.1 k-Core Decomposition

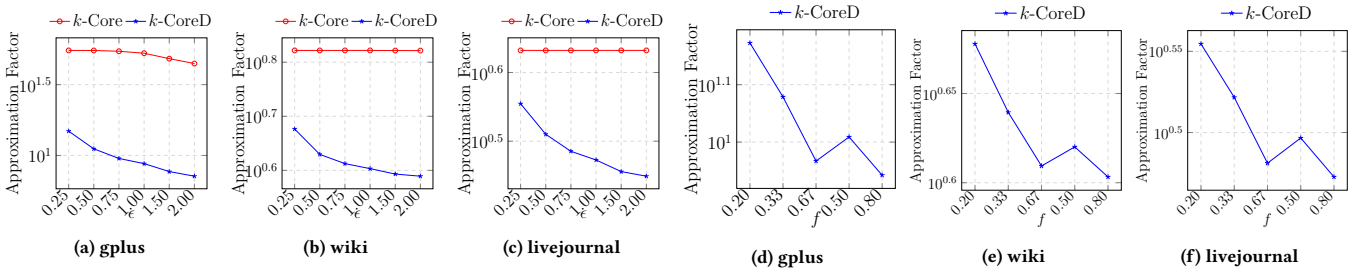
Response Time Fig. 8 shows the number of communication rounds required by our k-core decomposition algorithm, k-CoreD, compared to the baseline, k-Core, [15]. Our approach reduces the number of rounds by ***two orders of magnitude***, well within our theoretical bound (without noise) of $O(\log(n)\cdot\log(D_{\max}))$, demonstrating its scalability and efficiency for large-scale graphs.

While the comparison of the number of rounds clearly demonstrates the improvements achieved by k-CoreD, a direct comparison of runtime between the two algorithms is not possible. In the baseline algorithm, the absence of bias terms prevents nodes from progressing through the levels in the LDS, effectively resulting in no computations being performed during each round. Conversely, k-CoreD ensures node movement in every round, thereby increasing the computational workload per round. For this reason, we focus on presenting and analyzing the runtime results of k-CoreD.

Fig. 7 shows the response times of k-CoreD across all datasets. Our algorithm efficiently processes large-scale graphs, including billion-edge datasets like *twitter* and *friendster*, within **four hours**. These results validate the scalability and practicality of k-CoreD, demonstrating the impact of degree thresholding and bias terms in delivering both theoretical and practical improvements.

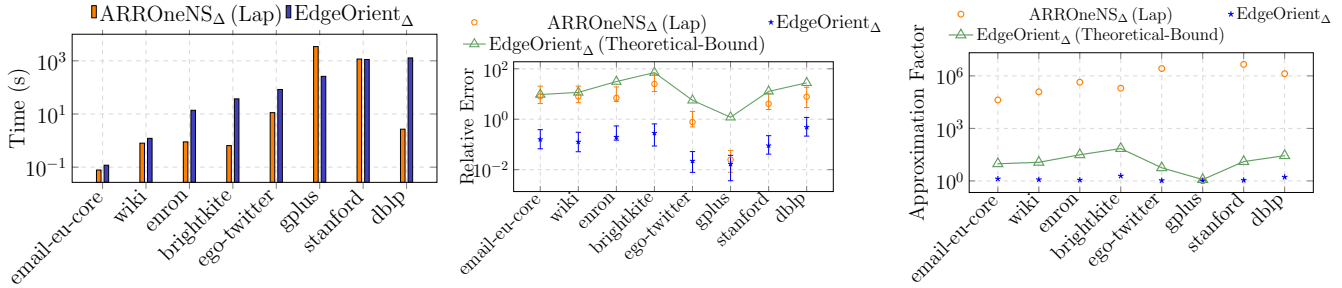
Accuracy We calculate the approximation factor for each node as $a_v = \frac{\max(s_v,t_v)}{\min(s_v,t_v)}$, where s_v is the approximate core number and t_v is the true core number. We use this metric to be consistent with the best-known *non-private* k-core decomposition implementations [13, 47]. These individual node approximation factors facilitate the computation of aggregate metrics: the average, maximum, 80^{th}, and 95^{th} percentile approximation factors for each graph. The theoretical approximation bound in the absence of noise is calculated as $(2+\eta)$, which is 5.625 for all graphs (labeled Theoretical Bound). Additionally, we adjust this bound to account for noise by incorporating the additive error term $\frac{\log^3_{1+\eta/5}(D_{max})}{\varepsilon}$, where n is the number of nodes, D_{max} is the maximum degree (labeled Theoretical Bound (With Noise)). For this bound, we compute the effect of the additive noise on the multiplicative factor by adding $\frac{\log^3_{1+\eta/5}(D_{max})}{\varepsilon\cdot k_{\max}}$, where $k_{\max}$ is the maximum core number, to 5.625. Note that such a theoretical bound is a *lower bound* on the effect of the additive error on the multiplicative factor; for smaller core numbers, e.g. $k_{\min} << k_{\max}$, the additive error leads to a *much greater* factor.

Figs. 6a to 6c present the approximation factors achieved by our approach and the baseline algorithm [15], alongside the theoretical bounds across various datasets. On average, our method maintains



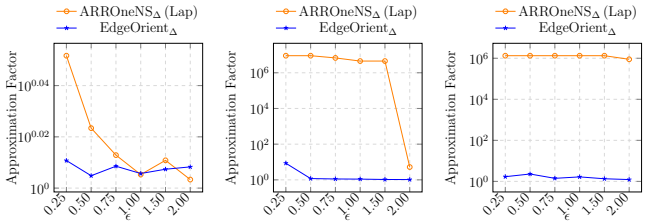
(a) gplus (b) wiki (c) livejournal (d) gplus (e) wiki (f) livejournal

Figure 9: Avg. approximation factor for k-Core decomposition vs. epsilon (ε) and split fraction (f).



(a) Average Response Time (b) Relative Error (c) Average Approximation Factor

Figure 10: Triangle Counting Results.



(a) gplus (b) stanford (c) dblp

Figure 11: Avg. approx factor for triangle counting vs. epsilon (ε).

approximation factors below **4x** across all datasets, with the 80$^{\text{th}}$ percentile staying under **5.5x**, as illustrated in Fig. 6a and Fig. 6b, demonstrating significantly lower variance compared to the baseline. These results remain well within the theoretical bounds without noise. Compared to the baseline, k-CoreD consistently achieves better or comparable performance. Notably, for graphs such as *brain* and *gplus*, k-CoreD reduces the approximation factors from 131.55 to 4.11 and from 52.71 to 3.27, improvements of over **31x** and **16x**, respectively. Similarly, for *orkut* and *wiki*, our algorithm improves the approximation factors by **6.6x** and **1.9x**, respectively.

However, for many graphs, the difference between the baseline and our approach is less pronounced. This is due to the baseline algorithm's inability to move nodes up levels in the LDS, which results in an approximate core number of 2.5 for most nodes. Given that real-world graphs often exhibit small core numbers (Table 1), the average approximation factor for the baseline becomes skewed, particularly for graphs with a significant proportion of low-core nodes. In contrast, for graphs with larger core numbers, such as *gplus*, *brain*, and *orkut*, our method demonstrates significant improvements. The advantages of our algorithm become more evident when examining the 80$^{\text{th}}$ and 95$^{\text{th}}$ percentile approximation factors. As shown in Fig. 6b, our method consistently achieves a notable reduction in the 80$^{\text{th}}$ percentile error compared to the baseline, with reductions of up to **42x**, **40x**, and **7.7x** for graphs with large core numbers, such as *brain*, *gplus*, and *orkut*, respectively. Similarly, in the 95$^{\text{th}}$ percentile error (Fig. 6c), our method achieves reductions of nearly **49x**, **81x**, and **7x** for the same graphs. These results underscore the robustness and scalability of our algorithm in handling diverse graph structures with varying core number distributions.

Ablation Study We analyze the effect of varying the privacy parameter, ε, and the privacy split fraction, f, on the utility of the k-core decomposition algorithm by plotting the average approximation factor across different datasets: *gplus*, *wiki*, and *livejournal*, for our algorithm and the baseline. From the results shown in Fig. 9, we observe that the approximation factor improves as ε increases, which aligns with the theoretical expectations of differential privacy where higher ε allows for less noise and greater utility.

When varying the privacy split fraction, f (Fig. 9), we note that an optimal value of 0.8 consistently minimizes the approximation factor across all datasets. This is the case since degree thresholding affects the amount of noise that will be added per level for later computations. This demonstrates that allocating $0.8 \cdot \varepsilon$ for degree thresholding and $0.2 \cdot \varepsilon$ for the level moving step strikes a balance between the two steps, ensuring better overall performance. Consequently, we use $f = 0.8$ for all other experiments.

5.2 Triangle Counting

Response Time Fig. 10a shows the average response times of our LEDP triangle counting algorithm, EdgeOrient$_\Delta$, implemented in our distributed (LEDP-DS) framework. We compare our algorithm to **ARROneNS$_\Delta$ (Lap)** from [38]. While ARROneNS$_\Delta$ (Lap) is implemented in C++, our Golang implementation demonstrates comparable performance across most datasets, with notable speedups for large graphs. Specifically, for *gplus*, our algorithm achieves a **speedup of 3.45x**, while for other graphs such as *email-eu-core, stanford*, and *wiki*, our performance is comparable despite the communication overhead. However, for the *enron, brightkite*, and *dblp* dataset, our algorithm is slower. This discrepancy is likely due to

the smaller sizes of the graphs so a centralized algorithm will perform better than a distributed algorithm. These results emphasize the scalability and practicality of EdgeOrient$_\Delta$ for large-scale graph analysis, highlighting its ability to handle diverse graphs.

Accuracy Following the evaluation methodology in [38], we compute relative error for a graph using $\frac{|\widetilde{\Delta}-\Delta|}{\Delta}$, where $\widetilde{\Delta}$ represents the approximated triangle count and Δ the true triangle count. Additionally, we apply the theoretical bounds from Theorem 4.4 to our analysis. According to Fig. 10b, our algorithm, EdgeOrient$_\Delta$, consistently achieves relative errors ranging from $\mathbf{10^{-1}}$ to $\mathbf{10^{-2}}$ across all datasets, remaining well within the theoretical bounds.

Compared to ARROneNS$_\Delta$ (Lap), our algorithm gives better relative errors by **53x - 89x**, for all graphs except *gplus*, where we achieve slightly better but comparable accuracy. In contrast to GroupRR [33], which we could only run on the *wiki* dataset due to the 4-hour timeout limit, our algorithm not only matches accuracy but also has a response time ***two orders of magnitude faster***.

To further analyze the limitations of prior approaches, we compute the multiplicative approximation factor of the triangle count, defined as $\frac{\max\left(\widetilde{\Delta},\Delta\right)}{\max\left(1,\min\left(\widetilde{\Delta},\Delta\right)\right)}$. Unlike relative error, this metric explicitly accounts for cases where algorithms, such as the one in [38], produce negative triangle counts, which severely undermines their utility in real-world scenarios. As shown in Fig. 10c, our algorithm achieves consistently small approximation factors across all graphs, remaining within $[\mathbf{1.01}, \mathbf{1.93}]$, and reducing the factor by **six orders of magnitude** compared to [38]. These results highlight the robustness of our approach, EdgeOrient$_\Delta$, in maintaining low and stable approximation factors across diverse graph structures, demonstrating superior performance compared to previous methods that fail to deliver consistent utility across different datasets.

Ablation Study To evaluate the impact of the privacy parameter ε on utility, we analyze and plot the approximation factors for varying values of ε across three representative graphs: *gplus*, *dblp*, and *stanford*. The approximation factors are compared against those reported in [38]. Our results demonstrate that for *stanford* and *dblp*, our algorithm achieves a significant reduction in approximation factor by up to **six orders of magnitude**. For *gplus*, while the approximation factors are comparable for higher values of ε, our algorithm achieves better utility for smaller values of ε. Specifically, at $\varepsilon = 0.25$, our algorithm achieves an approximation factor of 1.025, compared to 1.126 for ARROneNS$_\Delta$ (Lap), an improvement of **9.8%**. Similarly, at $\varepsilon = 0.50$, our algorithm achieves a factor of 1.011, compared to 1.055, an improvement of **4.2%** This highlights the ability of our approach to offer better utility even under stricter privacy constraints, underscoring its advantage over [38].

Additionally, we observe that the utility of our algorithm consistently improves as the privacy parameter ε increases, which aligns with the theoretical expectations of differential privacy, where higher values of ε results in less noise.

6 CONCLUSION

Large-scale network analysis (social, biological, etc.) often involves privacy concerns when analyzing sensitive data. This paper addresses this using local edge differential privacy (LEDP), where nodes protect their private information without sending it to a trusted central authority. We introduce novel LEDP algorithms for key graph statistics, k-core decomposition and triangle counting, improving upon prior work, and present the first distributed implementation framework to simulate distributed LEDP algorithms on one machine and evaluate them on real-world data. Experiments show significant improvements, where the k-core decomposition approximations are bounded by the theoretical guarantees of non-private algorithms on average and our triangle count errors are reduced by nearly two orders of magnitude over previous work, with comparable runtimes. In ongoing work, we are extending this framework to multi-machine testing with network communication, acknowledging additional latency and overhead, and to support a broader range of algorithms. We encourage the community to utilize our open-source framework (available at [1] for evaluating their LEDP graph algorithms, fostering further advancements in this exciting field.

REFERENCES

[1] 2024. DistributedLEDPGraphAlgos. https://github.com/mundrapranay/DistributedLEDPGraphAlgos.

[2] John M Abowd, Robert Ashmead, Ryan Cumings-Menon, Simson Garfinkel, Micah Heineck, Christine Heiss, Robert Johns, Daniel Kifer, Philip Leclerc, Ashwin Machanavajjhala, et al. 2022. The 2020 census disclosure avoidance system topdown algorithm. *Harvard Data Science Review* 2 (2022).

[3] Mohammad Al Hasan and Vachik S Dave. 2018. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1226.

[4] Victor Balcer and Salil P. Vadhan. 2018. Differential Privacy on Finite Computers. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*. 43:1–43:21.

[5] Bradley R Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Mike Personick, Karthik Rajan, et al. 2018. Amazon Neptune: Graph Data Management in the Cloud.. In *ISWC (P&D/Industry/BlueSky)*.

[6] Arijit Bishnu, Debarshi Chanda, and Gopinath Mishra. 2025. Arboricity and Random Edge Queries Matter for Triangle Counting using Sublinear Queries. arXiv:2502.15379 [cs.DS] https://arxiv.org/abs/2502.15379

[7] Felipe T Brito, Victor AE Farias, Cheryl Flynn, Subhabrata Majumdar, Javam C Machado, and Divesh Srivastava. 2023. Global and local differentially private release of count-weighted graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[8] Mark Bun and Thomas Steinke. 2016. Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds. In *International Conference on Theory of Cryptography*. 635–658.

[9] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Trans. Inf. Syst. Secur.* 14, 3, Article 26 (Nov. 2011), 24 pages. https://doi.org/10.1145/2043621.2043626

[10] Ho-Chun Herbert Chang and Emilio Ferrara. 2022. Comparative analysis of social bots and humans during the COVID-19 pandemic. *Journal of Computational Social Science* 5, 2 (2022), 1409–1425.

[11] Martino Ciaperoni, Edoardo Galimberti, Francesco Bonchi, Ciro Cattuto, Francesco Gullo, and Alain Barrat. 2020. Relevance of temporal cores for epidemic spread in temporal networks. *Scientific reports* 10, 1 (2020), 12529.

[12] Camil Demetrescu, Andrew V Goldberg, David S Johnson, et al. 2008. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*. Springer US, 395–398.

[13] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. 2017. Julienne: A Framework for Parallel Graph Algorithms Using Work-efficient Bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 293–304.

[14] Laxman Dhulipala, George Z. Li, and Quanquan C. Liu. 2024. Near-Optimal Differentially Private k-Core Decomposition. arXiv:2312.07706 [cs.DS] https://arxiv.org/abs/2312.07706

[15] Laxman Dhulipala, Quanquan C. Liu, Sofya Raskhodnikova, Jessica Shi, Julian Shun, and Shangdi Yu. 2022. Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 754–765.

[16] Michael Dinitz, Satyen Kale, Silvio Lattanzi, and Sergei Vassilvitskii. 2024. Almost Tight Bounds for Differentially Private Densest Subgraph. arXiv:2308.10316 [cs.DS]

[17] Alan A.A. Donovan and Brian W. Kernighan. 2015. *The Go Programming Language* (1st ed.). Addison-Wesley Professional.

[18] Cynthia Dwork and Jing Lei. 2009. Differential Privacy and Robust Statistics. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. 371–380.

[19] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography*. 265–284.

[20] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential Privacy under Continual Observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. 715–724.

[21] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. 2010. Boosting and Differential Privacy. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*. 51–60.

[22] Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam D. Smith. 2023. Triangle Counting with Local Edge Differential Privacy. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 52:1–52:21. https://doi.org/10.4230/LIPICS.ICALP.2023.52

[23] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. 2003. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 211–222.

[24] Alireza Farhadi, MohammadTaghi Hajiaghayi, and Elaine Shi. 2021. Differentially Private Densest Subgraph. *CoRR* abs/2106.00508 (2021), 11581–11597. arXiv:2106.00508 https://arxiv.org/abs/2106.00508

[25] Victor AE Farias, Felipe T Brito, Cheryl Flynn, Javam C Machado, Subhabrata Majumdar, and Divesh Srivastava. 2020. Local dampening: Differential privacy for non-numeric queries via local sensitivity. *arXiv preprint arXiv:2012.04117* (2020).

[26] Nan Fu, Weiwei Ni, Sen Zhang, Lihe Hou, and Dongyue Zhang. 2023. GC-NLDP: A graph clustering algorithm with local differential privacy. *Computers & Security* 124 (2023), 102967.

[27] Kayo Fujimoto, Dimitrios Paraskevis, Jacky C Kuo, Camden J Hallmark, Jing Zhao, Andre Hochi, Lisa M Kuhns, Lu-Yu Hwang, Angelos Hatzakis, and John A Schneider. 2022. Integrated molecular and affiliation network analysis: Core-periphery social clustering is associated with HIV transmission patterns. *Social networks* 68 (2022), 107–117.

[28] Taolin Guo, Shunshun Peng, Yong Li, Mingliang Zhou, and Trieu-Kien Truong. 2023. Community-based social recommendation under local differential privacy protection. *Information Sciences* 639 (2023), 119002.

[29] Yang Guo, Fatemeh Esfahani, Xiaojian Shao, Venkatesh Srinivasan, Alex Thomo, Li Xing, and Xuekui Zhang. 2022. Integrative COVID-19 biological network inference with probabilistic core decomposition. *Briefings in Bioinformatics* 23, 1 (2022), bbab455.

[30] Jonathan Hehir, Aleksandra Slavković, and Xiaoyue Niu. 2022. Consistent spectral clustering of network block models under local differential privacy. *The Journal of privacy and confidentiality* 12, 2 (2022).

[31] Monika Henzinger, A. R. Sricharan, and Leqi Zhu. 2024. Tighter Bounds for Local Differentially Private Core Decomposition and Densest Subgraph. *CoRR* abs/2402.18020 (2024). https://doi.org/10.48550/ARXIV.2402.18020 arXiv:2402.18020

[32] Seira Hidano and Takao Murakami. 2022. Degree-preserving randomized response for graph neural networks under local differential privacy. *arXiv preprint arXiv:2202.10209* (2022).

[33] Quentin Hillebrand, Vorapong Suppakitpaisarn, and Tetsuo Shibuya. 2023. Communication Cost Reduction for Subgraph Counting under Local Differential Privacy via Hash Functions. *arXiv preprint arXiv:2312.07055* (2023).

[34] Quentin Hillebrand, Vorapong Suppakitpaisarn, and Tetsuo Shibuya. 2023. Unbiased locally private estimator for polynomials of laplacian variables. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 741–751.

[35] Petter Holme and Nelly Litvak. 2017. Cost-efficient vaccination protocols for network epidemiology. *PLoS computational biology* 13, 9 (2017), e1005696.

[36] Jacob Imola, Alessandro Epasto, Mohammad Mahdian, Vincent Cohen-Addad, and Vahab Mirrokni. 2023. Differentially private hierarchical clustering with provable approximation guarantees. In *International Conference on Machine Learning*. PMLR, 14353–14375.

[37] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2021. Locally Differentially Private Analysis of Graph Statistics. In *30th USENIX Security Symposium*. 983–1000.

[38] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Communication-Efficient Triangle Counting under Local Differential Privacy. In *31st USENIX Security Symposium*. 537–554.

[39] Linyu Jiang, Yukun Yan, Zhihong Tian, Zuobin Xiong, and Qilong Han. 2023. Personalized sampling graph collection with local differential privacy for link prediction. *World Wide Web* 26, 5 (2023), 2669–2689.

[40] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[41] Muah Kim, Onur Günlü, and Rafael F. Schaefer. 2021. Federated Learning with Local Differential Privacy: Trade-Offs Between Privacy, Utility, and Communication. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2650–2654. https://doi.org/10.1109/ICASSP39728.2021.9413764

[42] Tejas Kulkarni. 2019. Answering Range Queries Under Local Differential Privacy. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1832–1834. https://doi.org/10.1145/3299869.3300102

[43] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *www*. 591–600.

[44] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. (2014).

[45] Xiaoguang Li, Ninghui Li, Wenhai Sun, Neil Zhenqiang Gong, and Hui Li. 2023. Fine-grained poisoning attack to local differential privacy protocols for mean and variance estimation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1739–1756.

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

[47] Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2022. Parallel Batch-Dynamic Algorithms for k-Core Decomposition and Related Graph Problems. In *34th ACM Symposium on Parallelism in Algorithms and Architectures*. 191–204.

[48] Shang Liu, Yang Cao, Takao Murakami, Jinfei Liu, and Masatoshi Yoshikawa. 2023. CARGO: Crypto-Assisted Differentially Private Triangle Counting without Trusted Servers. *arXiv preprint arXiv:2312.12938* (2023).

[49] Shang Liu, Yang Cao, Takao Murakami, and Masatoshi Yoshikawa. 2022. A crypto-assisted approach for publishing graph statistics with node local differential privacy. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 5765–5774.

[50] Yuhan Liu, Tianhao Wang, Yixuan Liu, Hong Chen, and Cuiping Li. 2024. Edge-Protected Triangle Count Estimation under Relationship Local Differential Privacy. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[51] Yuhan Liu, Suyun Zhao, Yixuan Liu, Dan Zhao, Hong Chen, and Cuiping Li. 2022. Collecting triangle counts with edge relationship local differential privacy. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2008–2020.

[52] Zemin Liu, Vincent W. Zheng, Zhou Zhao, Hongxia Yang, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2018. Subgraph-Augmented Path Embedding for Semantic User Search on Heterogeneous Social Network. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1613–1622. https://doi.org/10.1145/3178876.3186073

[53] Pathum Chamikara Mahawaga Arachchige, Dongxi Liu, Seyit Camtepe, Surya Nepal, Marthie Grobler, Peter Bertok, and Ibrahim Khalil. 2022. Local Differential Privacy for Federated Learning. In *European Symposium on Research in Computer Security*. Springer, 195–216.

[54] Naoki Masuda, Michiko Sakaki, Takahiro Ezaki, and Takamitsu Watanabe. 2018. Clustering Coefficients for Correlation Networks. *Frontiers in Neuroinformatics* 12 (2018). https://doi.org/10.3389/fninf.2018.00007

[55] Tamara T Mueller, Dmitrii Usynin, Johannes C Paetzold, Daniel Rueckert, and Georgios Kaissis. 2022. SoK: Differential privacy on graph-structured data. *arXiv preprint arXiv:2203.09205* (2022).

[56] Takao Murakami and Yuichi Sei. 2023. Automatic Tuning of Privacy Budgets in Input-Discriminative Local Differential Privacy. *IEEE Internet of Things Journal* (2023).

[57] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. 2022. Local and Central Differential Privacy for Robustness and Privacy in Federated Learning. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/auto-draft-204/

[58] Neo4j. 2012. Neo4j - The World's Leading Graph Database. http://neo4j.org/

[59] Dung Nguyen, Mahantesh Halappanavar, Venkatesh Srinivasan, and Anil Vullikanti. 2024. Faster approximate subgraph counts with privacy. *Advances in Neural Information Processing Systems* 36 (2024).

[60] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth Sensitivity and Sampling in Private Data Analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. 75–84.

[61] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[62] Arnau Prat-Pérez, David Dominguez-Sal, Josep M Brunat, and Josep-Lluis Larriba-Pey. 2012. Shaping communities out of triangles. In *Proceedings of the ACM international Conference on Information and Knowledge Management*. 1677–1681.

[63] Lei Qin, Yidan Wang, Qiang Sun, Xiaomei Zhang, Ben-Chang Shia, Chengcheng Liu, et al. 2020. Analysis of the covid-19 epidemic transmission network in

mainland china: K-core decomposition study. *JMIR public health and surveillance* 6, 4 (2020), e24291.

[64] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2017. Generating Synthetic Decentralized Social Graphs with Local Differential Privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 425–438. https://doi.org/10.1145/3133956.3134086

[65] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 29.

[66] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. 2021. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 327–343.

[67] Higor S Monteiro, Shaojun Luo, Saulo DS Reis, Carles Igual, Antonio S Lima Neto, Matias Travizan, Jose Soares De Andrade Jr, Hernan Makse, et al. 2021. Superspreading k-cores at the center of Covid-19 pandemic persistence. *Bulletin of the American Physical Society* 66 (2021).

[68] Mohamed Seif, Dung Nguyen, Anil Vullikanti, and Ravi Tandon. 2022. Differentially private community detection for stochastic block models. *arXiv preprint arXiv:2202.00636* (2022).

[69] Matteo Serafino, Higor S. Monteiro, Shaojun Luo, and Hernán A. Makse. 2020. *Project COVID19 K-core tracker.* https://github.com/makselab/COVID19

[70] Matteo Serafino, Higor S Monteiro, Shaojun Luo, Saulo DS Reis, Carles Igual, Antonio S Lima Neto, Matías Travizano, José S Andrade Jr, and Hernán A Makse. 2022. Digital contact tracing and network theory to stop the spread of COVID-19 using big-data on human mobility geolocalization. *PLOS Computational Biology* 18, 4 (2022), e1009865.

[71] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *Proceedings of the Network and Distributed System Security Symposium*.

[72] Konstantinos Sotiropoulos and Charalampos E. Tsourakakis. 2021. Triangle-Aware Spectral Sparsifiers and Community Detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1501–1509. https://doi.org/10.1145/3447548.3467260

[73] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Hui Wang, and Ting Yu. 2019. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 703–717.

[74] Marco Sánchez-Aguayo, Luis Urquiza-Aguiar, and José Estrada-Jiménez. 2021. Fraud Detection Using the Fraud Triangle Theory and Data Mining Techniques: A Literature Review. *Computers* 10, 10 (2021). https://doi.org/10.3390/computers10100121

[75] Differential Privacy Team. 2017. Learning with Privacy at Scale — machinelearning.apple.com. https://machinelearning.apple.com/research/learning-with-privacy-at-scale. [Accessed 10-04-2024].

[76] Google Differential Privacy Team. [n. d.]. GitHub - google/differential-privacy: Google's differential privacy libraries. — github.com. https://github.com/google/differential-privacy. [Accessed 12-04-2024].

[77] Ekin Tire and M. Emre Gursoy. 2024. Answering Spatial Density Queries Under Local Differential Privacy. *IEEE Internet of Things Journal* 11, 10 (2024), 17419–17436. https://doi.org/10.1109/JIOT.2024.3357570

[78] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel index-based structural graph clustering and its approximation. In *Proceedings of the International Conference on Management of Data*. 1851–1864.

[79] Songlei Wang, Yifeng Zheng, Xiaohua Jia, Qian Wang, and Cong Wang. 2023. MAGO: Maliciously Secure Subgraph Counting on Decentralized Social Graphs. *IEEE Transactions on Information Forensics and Security* (2023).

[80] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering Multi-Dimensional Analytical Queries under Local Differential Privacy. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 159–176. https://doi.org/10.1145/3299869.3319891

[81] Yanling Wang, Qian Wang, Lingchen Zhao, and Cong Wang. 2023. Differential privacy in deep learning: Privacy and beyond. *Future Generation Computer Systems* (2023).

[82] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.

[83] Zihang Xiang, Tianhao Wang, and Di Wang. 2023. Preserving Node-level Privacy in Graph Neural Networks. *arXiv preprint arXiv:2311.06888* (2023).

[84] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2020. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2020), 4905–4920.

[85] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2020. Towards locally differentially private generic graph metric estimation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1922–1925.

[86] Wei Zeng, An Zeng, Hao Liu, Ming-Sheng Shang, and Tao Zhou. 2014. Uncovering the information core in recommender systems. *Scientific reports* 4 (August 2014), 6140. https://doi.org/10.1038/srep06140

[87] Da Zhong, Ruotong Yu, Kun Wu, Xiuling Wang, Jun Xu, and Wendy Hui Wang. 2023. Disparate Vulnerability in Link Inference Attacks against Graph Neural Networks. *Proceedings on Privacy Enhancing Technologies* (2023).

[88] Xiaochen Zhu, Vincent YF Tan, and Xiaokui Xiao. 2023. Blink: Link Local Differential Privacy in Graph Neural Networks via Bayesian Estimation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2651–2664.