arXiv:2211.10887v1 [cs.DS] 20 Nov 2022

Differential Privacy from Locally Adjustable Graph Algorithms: k-Core Decomposition, Low Out-Degree Ordering, and Densest Subgraphs

Laxman Dhulipala, Quanquan C. Liu, Sofya Raskhodnikova,
Jessica Shi, Julian Shun, Shangdi Yu

Abstract

Differentially private algorithms allow large-scale data analytics while preserving user privacy. Designing such algorithms for graph data is gaining importance with the growth of large networks that model various (sensitive) relationships between individuals. While there exists a rich history of important literature in this space, to the best of our knowledge, no results formalize a relationship between certain parallel and distributed graph algorithms and differentially private graph analysis. In this paper, we define *locally adjustable* graph algorithms and show that algorithms of this type can be transformed into differentially private algorithms.

Our formalization is motivated by a set of results that we present in the central and local models of differential privacy for a number of problems, including k-core decomposition, low out-degree ordering, and densest subgraphs. First, we design an ε-edge differentially private (DP) algorithm that returns a subset of nodes that induce a subgraph of density at least $\frac{D^*}{1+\eta} - O\left(\text{poly}(\log n)/\varepsilon\right)$, where D^* is the density of the densest subgraph in the input graph (for any constant $\eta > 0$). This algorithm achieves a two-fold improvement on the multiplicative approximation factor of the previously best-known private densest subgraph algorithms while maintaining a near-linear runtime.

Then, we present an ε-locally edge differentially private (LEDP) algorithm for k-core decompositions. Our LEDP algorithm provides approximates the core numbers (for any constant $\eta > 0$) with $(2+\eta)$ multiplicative and $O\left(\text{poly}\left(\log n\right)/\varepsilon\right)$ additive error. This is the first differentially private algorithm that outputs private k-core decomposition statistics. We also modify our algorithm to return a differentially private low out-degree ordering of the nodes, where orienting the edges from nodes earlier in the ordering to nodes later in the ordering results in out-degree at most $O\left(d + \text{poly}\left(\log n\right)/\varepsilon\right)$ (where d is the degeneracy of the graph). A small modification to the algorithm also yields a ε-LEDP algorithm for $(4+\eta, O\left(\text{poly}\left(\log n\right)/\varepsilon\right))$-approximate densest subgraph (which returns both the set of nodes in the subgraph and its density). Our algorithm uses $O(\log^2 n)$ rounds of communication between the curator and individual nodes.

Contents

1 Introduction

The *k-core decomposition* and related objects—densest subgraph and low out-degree ordering—are among the most important and widely used graph statistics in a variety of communities including databases [CZL+20, LZZ+19, ESTW19, BGKV14, MMSS20], machine learning [AHDBV05, ELM18, GLM19], graph analytics [KBST15, KM17, DBS17, DBS18], graph visualization [AHDBV05, CHK+07, YL15, ZZC+10], theoretical computer science [CSS21, ELM18, GLM19, MPM13, SCS20], and other communities [GBGL20, LYL+19, SGJS+13]. The k-core decomposition assigns a number to each node in a network which captures how well-connected it is to the rest of the network; it is useful in applications where one wants to find "influential" nodes, or to partition a network based on each node's influence. Concrete applications include diffusion protocols in epidemiological studies (where nodes with large core numbers tend to be "super-spreaders") [CGB+20, KGH+10, LTZY15, MRV16], community detection and computing network centrality measures (where centers tend to have larger core numbers) [DGP09, HJMA06, MPP+15, WCL+18, ZZQ+17],network visualization and modeling [AHDBV05, CHK+07, YL15, ZZC+10], protein interactions [ASM+06, BH03], and clustering (where k-core decompositions are often used for preprocessing to achieve faster performance) [GMTV14, LRJA10]. Because of these applications, finding fast and scalable algorithms for exact and approximate k-core decompositions is an active research area with many results spanning the past decade (see [MGPV20] for a survey of these results). and a plethora of results published just in the past year [CSS21, GPC21, GZE+21, LZL+21b, LZL+21a, LZHX21, LYC+21, VAGK21, VKBK21, WZL+22, ZHH+21].

However, one increasingly important topic that has been overlooked thus far in the community is the privacy of the individuals whose data are used for computing the core numbers. Privacy measures are particularly important for statistics such as k-core numbers, since such statistics output a value for each individual in the data set, allowing for more effective attacks that can decipher individual links and also the information of one or more individuals in the data set. Such ominous possibilities call for a formal study of k-core decomposition algorithms that are *privacy preserving*, which is the focus of this paper.

Given an undirected graph G with n nodes and m edges, the k-core of the graph is the maximal subgraph $H \subseteq G$ such that the induced degree of every node in H is at least k. The *k-core decomposition* of the graph is a partition of the nodes of the graph into layers such that layer k contains all the nodes that belong to the k-core but not the $(k+1)$-core. We illustrate the kind of attack on a (private) input data set that can occur provided the (anonymized) k-core decomposition of a social network graph. In particular, the k-core decomposition gives much more information about the structure of the graph than many other graph statistics. Consider a social network graph consisting of a k-clique and a k-ary tree. Suppose the attacker does not know the graph but has access to the exact core numbers of individuals in the data set. From the core numbers, the attacker can determine the individuals in the clique and the edges between them. Such an attack is not possible, for example, if instead of the core numbers, the attacker has access to the degree distribution of the graph. In this case, the attacker cannot differentiate between the clique and the non-leaf nodes of the k-ary tree.

Consider a graph G that, instead of friendships, represent sensitive information such as HIV transmissions [LPA+14, WWM+15, WPF+17] or cryptocurrency payments [MB19]. Two members of a clique in G, users A and B, may not want others to know that they share an edge; however, the k-core decomposition of G reveals that they do indeed share an edge. Such an attack emphasizes the need for k-core decomposition algorithms that provide privacy for individuals.

With this goal in mind, we present novel *differentially private* (DP) algorithms for the k-core decomposition and related objects—densest subgraph and low out-degree ordering—that match the multiplicative approximation bounds of the known non-private algorithms. Differential privacy [DMNS06] is the gold standard for privacy in data analysis. We present results in two differential privacy models: *central* [DMNS06] and *local* [KLN+11]. The central model assumes a trusted curator that gathers and processes non-private data from users. In contrast, the local differential privacy model assumes *no* trusted third-party. Each node represents an individual device of a user (e.g., phone) which releases privatized data. A third-party can act as an *untrusted curator* for computing statistics on the released data. Such models are particularly important as individuals become more wary of central authorities; and it is also a liability for companies to keep such sensitive data. Local differential privacy is a stronger notion of privacy than central differential privacy because nobody besides the owner touches any private data. Furthermore, privacy in the local model

implies privacy in the central model. On the other hand, local differential privacy is more restrictive for the algorithm designer, and, for some problems, locally differentially private algorithms must have larger error than DP algorithms. In fact, several known error lower bounds exhibit gaps between the central and local models that could be as large as polynomial in the size of the input (see, e.g., [BNO08, CSS12, DMNS06]).

Outline We provide all definitions and notation for this paper in Section 2. We summarize our contributions and provide a technical overview in Section 3. In Section 4, we present our locally differentially private k-core decomposition, low out-degree ordering, and densest subgraph algorithms. In Section 5, we provide our DP densest subgraph algorithm that achieves a better multiplicative factor approximation than our densest subgraph algorithm in the local model. Finally, in Section 6, we provide our privacy framework that allows us to convert a class of algorithms that we call *locally adjustable* into differentially private algorithms. It is open whether one can show that, in general, the utility of locally adjustable algorithms does not degrade significantly during the transformation.

Related Works DP algorithms have been developed for graph statistics such as subgraph counts [KRSY14, BBDS13, CZ13, IMC22, IMC21, KNRS13, SXK+19, ZCP+15], degree distribution [HLMJ09, DLL16, RS16, ZNF21], minimum spanning tree and clustering [NRS07, HL18, KS18, NSV16], spectral properties [WWW13, AU19], cut problems [GLM+10, AU19, EKKL20, KL10, Sea16, Sta21], and parameter estimation [LM14, YHA+20a, YHA+20b].

DP Densest Subgraph. The currently best DP algorithms for densest subgraphs are due to Nguyen and Vullikanti [NV21] and Farhadi et al. [FHS22]. We describe them in detail in Section 3.

Non-Private Algorithms for k-Core Decomposition and Related Problems. In the non-DP, fully-dynamic setting, Bhattacharya et al. [BHNT15], Henzinger et al. [HNW20], and Sawlani and Wang [SW20] provide sequential algorithms that use $\text{poly}(\log n)$ update time and obtain a $(4+\eta)$-approximate densest subgraph, $O(1)$-approximate low out-degree ordering, and $(1+\eta)$-approximate densest subgraph, respectively, for any $\eta > 0$. Sun et al. [SCS20] provide the first dynamic $(4+\eta)$-approximate k-core decomposition in the sequential model, using a peeling algorithm. A similar technique is used by Chan et al. [CSS21] in the distributed setting. Ghaffari et al. [GLM19] give various algorithms for $(1+\varepsilon)$-approximate k-core decomposition in the massively parallel computation (MPC) model. Finally, Liu et al. [LSY+22] formulate a parallel, batch-dynamic $(4+\eta)$-approximate k-core decomposition algorithm which we use in our work[1].

2 Preliminaries

We provide both DP and LDP algorithms with a focus on the k-core decomposition, low out-degree ordering, and the densest subgraph. We define these problems here. All privacy tools presented in this section are used in both the DP and LDP settings.

2.1 Graph Definitions

We use $[n]$ to denote $\{1, \ldots, n\}$. We consider undirected graphs $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. For ease of indexing, we set $V = [n]$. The set of neighbors of a node $i \in [n]$ is denoted $N(i)$, and the degree of node i is denoted $\deg(i)$. Our algorithms take an input graph G and output an approximate ***core number*** for each node in the graph (Definition 2.1), an approximate ***densest subgraph*** (Definition 2.4), and an approximate ***low out-degree ordering*** (Definition 2.3).

Definition 2.1 ((ϕ, ζ)-Approximate Core Number)**.** *The* k-core *of a graph $G = (V, E)$ is a maximal subgraph H of G such that the induced degree of every node in H is at least k. A node $v \in V$ has* ***core number*** *κ if v is part of the κ-core but not the $(\kappa+1)$-core. Let $k(v)$ be the core number of v and $\hat{k}(v)$ 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 $k(v) - \zeta \leq \hat{k}(v) \leq \phi \cdot k(v) + \zeta$.*

[1] We only consider one-sided multiplicative error. Thus, we translate the approximation factors of [SCS20] and [LSY+22], which give *two-sided* error, to instead give one-sided error, resulting in an additional factor of 2 in our statement of their approximation bounds.

Whereas an algorithm that outputs the *exact* k-core decomposition does not satisfy the definition of DP (or LDP), we obtain an LDP algorithm for *approximate* k-core decomposition which gives $(2+\eta, O(\log^3 n/\varepsilon))$-approximate core numbers for any constant $\eta > 0$. We define the related concept of an ***approximate low out-degree ordering*** based on the definition of ***degeneracy***.

Definition 2.2 (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 2.3 ((ϕ, ζ)-Approximate Low Outdegree 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 outdegree at most $\phi \cdot d + \zeta$.*

We denote the ***density*** of a graph $G = (V, E)$ by $\rho(G) := \frac{|E|}{|V|}$. The ***densest subgraph*** problem is defined as follows.

Definition 2.4 (Densest Subgraph)**.** *The densest subgraph $S_{\max}$ of a graph $G = (V, E)$ is a maximal induced subgraph with maximum density.*

When defining an approximate densest subgraph, we remove the condition on maximality of the subgraph and require the density to be within the specified approximation factors.

Definition 2.5 ((ϕ, ζ)-Approximate Densest Subgraph)**.** *Let the density of the densest subgraph in G be D^* and $\phi \geq 1, \zeta \geq 0$. A (ϕ, ζ)-**approximate densest subgraph** S has density $\rho(S)$ at least $\frac{D^*}{\phi} - \zeta$.*

2.2 Differential Privacy

We consider two models of differential privacy: central [DMNS06] and local [KLN+11]. In the central model, there is a *trusted* curator that has direct access to the input, whereas in the local model, the curator is not trusted and gets access only to outputs of private algorithms, called *randomizers*. Both notions of privacy require a definition of *neighboring* inputs. We focus on ***edge-neighboring*** graphs, defined next.

Definition 2.6 (Edge-Neighboring [NRS07])**.** *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.*

Definition 2.7 (ε-Edge Differential Privacy [NRS07])**.** *Algorithm $\mathcal{A}(G)$, that takes as input a graph G and outputs some value in $Range(\mathcal{A})$[2], is ε-**edge differentially private** (ε-edge DP) if for all $S \subseteq Range(\mathcal{A})$ and all edge-neighboring graphs G and G',*

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

The primary complexity measure for ε-edge DP algorithms is the *running time*.

2.3 Local Edge Differential Privacy (LEDP)

The LEDP model is an extension of the local differential privacy (LDP) model originally introduced by [KLN+11]. Below we use the definitions given in [ELRS22] which are based on definitions in [JMNR19] for non-graph data. The LEDP was also defined in [IMC22, IMC21] for the special case of one round, and [IMC22] additionally provides a proposition on the sequential composition of their LEDP algorithms.

Our LEDP algorithms are described in terms of an (untrusted) *curator*, who does not have access to the graph's edges, and individual nodes. During each round, the curator first queries a set of nodes for information. Individual nodes, which have access only to their own (private) adjacency lists, then *release* information via *local randomizers,* defined next.

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

Definition 2.8 (Local Randomizer (LR)). *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,*

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

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 coins of R (but not *over the choice of the input).*

The information released via local randomizers is public to all nodes and the curator. The curator performs some computation on the released information and makes the result public. The overall computation is formalized via the notion of the transcript.

Definition 2.9 (LEDP). *A **transcript** π is a vector consisting of 4-tuples $(S_U^t, S_R^t, S_\varepsilon^t, S_Y^t)$ – encoding the set of parties chosen, set of randomizers assigned, set of randomizer privacy parameters, and set of randomized outputs produced – for each round t. Let S_π be the collection of all transcripts and S_R be the collection of all randomizers. Let $\perp$ denote a special character indicating that the computation halts. A **protocol** is an algorithm $\mathcal{A} : S_\pi \to (2^{[n]} \times 2^{S_R} \times 2^{\mathbb{R}^{\geq 0}} \times 2^{\mathbb{R}^{\geq 0}}) \cup \{\perp\}$ mapping transcripts to sets of parties, randomizers, and randomizer privacy parameters. The length of the transcript, as indexed by t, is its round complexity.*

*Given $\varepsilon \geq 0$, a randomized protocol $\mathcal{A}$ on (distributed) graph G is ε-**locally edge differentially private (ε-LEDP)** if the algorithm that outputs the entire transcript generated by $\mathcal{A}$ is ε-edge differentially private on graph G. If $t = 1$, that is, if there is only one round, then $\mathcal{A}$ is called **non-interactive**. Otherwise, $\mathcal{A}$ is called **interactive**.*

Since LEDP algorithms operate in the distributed setting, the complexity measures that we care about for our LEDP algorithms is the ***number of rounds*** of communication and the ***node communication complexity***, or the maximum size of each message sent from a user to the curator. We assume each user can see the public information for each round on a public "bulletin board".

2.4 Differential Privacy Tools

We define these techniques for DP but the same cited sources also prove they hold under LDP. Also for simplicity, we give these techniques in graph theoretic terms but they also hold for broader applications (e.g., for databases, etc.). Our algorithms use the notion of *sensitivity*, defined as follows.

Definition 2.10 (Global Sensitivity [DMNS06]). *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 $GS_f = \max_{G,G'} \|f(G) - f(G')\|_1$ for all pairs of (G, G') of edge-neighboring graphs.*

Our algorithms use the *symmetric geometric distribution* given in previous papers [BV18, CSS11, DMNS06, DNPR10, FHS22, SCR+11]. One can think of the symmetric geometric distribution as the "discrete Laplace distribution." The advantage of such a distribution is that we avoid the rounding issues of continuous distributions and this distribution is sufficient for our needs.

Definition 2.11 (Symmetric Geometric Distribution [BV18, SCR+11]). *The **symmetric geometric distribution**, denoted $\mathsf{Geom}(b)$, with input parameter $b \in (0, 1)$, takes integer values i where the probability mass function at i is $\frac{e^b-1}{e^b+1} \cdot e^{-|i| \cdot b}$.*

We denote a random variable drawn from the distribution as $X \sim \mathsf{Geom}(b)$.

With high probability or ***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 private 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 2.12. *With probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$, we can upper bound $X \sim \mathsf{Geom}(x)$ by $|X| \leq \frac{c \ln n}{x}$.*

Proof. The probability that $|X| > \frac{c \ln n}{x}$ is given by $2 \sum_{i=\frac{c \ln n}{x}+1}^{\infty} \frac{e^x-1}{e^x+1} \cdot e^{-|i| \cdot x} = 2 \cdot \frac{\exp\left(-x \cdot \frac{c \ln n}{x}\right)}{e^x+1} = 2 \cdot \frac{\exp(-c \ln n)}{e^x+1} \leq \frac{1}{n^c}$. Thus, the probability that $|X| \leq \frac{c \ln n}{x}$ is at least $1 - \frac{1}{n^c}$. ∎

The ***geometric mechanism*** uses the symmetric geometric distribution.

Definition 2.13 (Geometric Mechanism [BV18, CSS11, DMNS06, DNPR10])**.** *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 $\mathcal{M}_G(x, f(\cdot), \varepsilon) = f(x) + (Y_1, \ldots, Y_d)$, where $Y_i \sim \mathsf{Geom}(\varepsilon/GS_f)$ are i.i.d. random variables drawn from $\mathsf{Geom}(\varepsilon/GS_f)$ and x is a data set.*

Lemma 2.14 (Privacy of the Geometric Mechanism [BV18, CSS11, DMNS06, DNPR10])**.** *The geometric mechanism is ε-edge 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 DP algorithms are used on the *same* dataset. We also use a theorem for group differential privacy.

Theorem 2.15 (Composition Theorem [DMNS06, DL09, DRV10])**.** *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).*

Lemma 2.16 (Group Privacy [DMNS06])**.** *Let $k \in \mathbb{N}$. Every ε-differentially private algorithm $\mathcal{A}$ is $(k\varepsilon)$-differentially private for groups of size k. That is, for all data sets X, X' such that $\|X - X'\|_0 \leq k$ and all subsets $Y \subseteq \mathcal{Y}$,*

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

For completeness, we provide a proof of Theorem 2.15 in Appendix A. Finally, the post-processing theorem states that the result of post-processing on the output of an ε-edge DP algorithm is ε-edge DP.

Theorem 2.17 (Post-Processing [DMNS06, BS16])**.** *Let $\mathcal{M}$ be an ε-edge DP mechanism and h be an arbitrary (randomized) mapping from Range$(\mathcal{M})$ to an arbitrary set. The algorithm $h \circ \mathcal{M}$ is ε-edge DP.*[3]

2.5 Notations

All notations used in our paper are included in Table 1; notation used in a specific section are labeled with that section.

3 Our Contributions and Technical Overview

Our main contributions in this paper are our locally private approximation algorithms for k-core decomposition, low out-degree ordering, and densest subgraphs that achieve bicriteria approximations where the multiplicative factors *exactly match* the multiplicative approximations of the original non-private algorithms, and they only incur additional small $\frac{\text{poly}(\log n)}{\varepsilon}$ additive error. Our key observation is that the release of noisy *levels* in the recent *level data structures* of [BHNT15, HNW20, LSY+22] used for these problems allows for privacy at the cost of only a $\text{poly}(\log n)$ increase in the additive error.

We first give ε-LEDP algorithms using a number of recent non-private algorithms for static and dynamic orientation and k-core decomposition algorithms [BHNT15, HNW20, CSS21, LSY+22]. Our algorithms are the first ε-LEDP algorithms for approximate k-core decomposition, low out-degree orientation and densest subgraph. For the approximate k-core decomposition problem, we must return an approximate core number for *every* node in the graph. Even though a single edge addition can cause the *exact* core number of every node to change, we are able to give a $\left(2 + \eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate ε-LEDP algorithm.

One of the challenges in adapting known techniques to obtain LEDP algorithms for the k-core decomposition is that the vector of k-core numbers has high sensitivity. Specifically, its global sensitivity is n, as *one* edge update can cause the k-core number of *every* node to increase or decrease by 1. E.g., all nodes in a cycle have core number 2, and the deletion of any edge decreases the core number of *every* node by 1. [4]

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

[4] More details on these challenges can be found in Appendix C.

Notation	Meaning
$G = ([n], E)$	initial input graph
$\varepsilon > 0$	privacy parameter
$\eta > 0$	approximation parameter
i	each node i has a unique index $i \in [n]$
$N(i)$	i's set of neighbors
$[x]_a^b$ for $x, a, b \in \mathbb{R}$, where $a < b$	a if $x < a$; x if $x \in [a, b]$; and b if $x > b$
$G[V']$	induced subgraph by nodes in V'
$\rho(G[V'])$	density of induced subgraph by nodes in V'
$\mathsf{Geom}(b)$	symmetric geometric distribution (Definition 2.11) with parameter b
$[n]$	integers in $\{1, \ldots, n\}$
	Notation in Section 5
$\ell(e)$	load on edge e
$\hat{\alpha}_{eu}$	new load added to edge e from node u
T	number of phases
	Notation in Section 4
$\lambda \in (0, 1)$	constant parameter used in upper bounds on induced degree
$\psi \in (0, 1)$	constant parameter used to determine number of levels
L_r	array of node levels publicized by curator for round r
C	array containing core number estimates of each node
$\mathcal{F}(r)$	function that outputs $\lfloor r/(2\log_{(1+\psi)} n)\rfloor$
$\text{level}(i)$	the level of node i
$\delta(G[V'])$	sum of degrees in induced subgraph of V' divided by $\lvert V'\rvert$, i.e. $\frac{\sum_{v \in V'} \deg_{G[V']}(v)}{\lvert V'\rvert}$.
$\mathcal{U}_i(r)$	number of neighbors of node i at level r or higher

Table 1: Notation

The key to our results is to instead consider a function that returns the *induced degree* of each node in a special subgraph of the input graph. The sensitivity of this function is 2, since one additional edge can increase the degree of at most two nodes, each by 1. We bound the number of times the curator queries the value of this function by $O(\log^2 n)$ in the worst case, thus eliminating the need for the sparse vector technique (SVT) used in previous works [FHS22, FHO21].

Our algorithm runs in $O(\log^2 n)$ rounds where each node sends messages with $O(1)$ bits to the curator. Our algorithm is based on a level data structure investigated by a number of works [BHNT15, HNW20, LSY+22]. In these data structures, nodes are partitioned into levels. The key insight that allowed us to achieve privacy in the local model is that each node in our algorithm only requires knowledge of the levels their neighbors are on. Thus, nodes can publish a *noisy level* for each phase. Using the noisy level in multiple phases directly leads to core number estimates. Our algorithm matches the multiplicative approximation factor of the best known algorithm for this problem by Chan et al. [CSS21]. We also give a version of our algorithm that uses only $O(\log n)$ rounds (with messages of size $O(\log n)$ sent to the curator). We present this algorithm and its analysis in Section 4.

Theorem 3.1 (ε-LEDP k-Core Decomposition)**.** *There exists an $O(\log n)$ round ε-LEDP algorithm that, given a constant $\eta > 0$, returns $\left(2 + \eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate core numbers with high probability.*

As a consequence of our algorithm, we give an ε-LEDP algorithm for *approximate low out-degree ordering*. Edge orientations obtained from such orderings are useful in graph algorithm design and contribute to a variety of faster algorithms for fundamental problems, such as coloring [MB83], matching [CHS09], triangle counting [CN85], independent set [OSSW20], dominating set [AG09], and many others. Furthermore, many real-world graphs have small degeneracy [BS20, ELS13, ST15, DBS18], making such algorithms practically useful. Note that (perhaps obviously) an algorithm that explicitly outputs an orientation for every edge cannot be edge DP. Instead, the output of our private algorithm is an ordering on the vertices, and the edge orientation can be obtained by orienting each edge from the lower to the higher endpoint in the ordering. Private orderings have been considered in previous works: e.g., Gupta et al. [GLM+10] give an ε-edge DP ordering for the vertex cover problem (where the earliest endpoint of an edge covers that edge).

Theorem 3.2 (ε-LEDP Low Out-Degree Ordering)**.** *There exists an $O(\log^2 n)$-round ε-LEDP algorithm that, given a constant $\eta > 0$ and a graph of degeneracy d, returns a total ordering of the nodes such that*

orienting edges from nodes earlier in the ordering to nodes later in the ordering results in out-degree at most $(4+\eta)d + O\left(\frac{\log^3 n}{\varepsilon}\right)$*, with high probability.*

With an additional tweak, our algorithm also yields the first result for the densest subgraph problem in the local model.

Theorem 3.3 (ε-LEDP Densest Subgraph)**.** *There exists an* $O(\log n)$*-round* ε*-LEDP algorithm that returns a set of nodes that induce a* $\left(4+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$*-approximate densest subgraph. Our algorithm returns both the nodes in the densest subgraph as well as the approximate density.*

Interestingly, our *static* LEDP algorithms are inspired by a body of non-private works in the dynamic setting [BHNT15, HNW20, LSY+22, SCS20]. We show that the locality of these dynamic algorithms and the bounded sequential dependency paths allow us to provide our LEDP guarantees as well as minimize the number of rounds of communication. Our techniques may potentially lead to other LEDP algorithms inspired by non-private dynamic algorithms.

There are currently no known $(1+\eta)$-approximate k-core decomposition algorithm (even in the non-private setting) that use $\text{poly}(\log n)$ phases. (Non-private $(1+\eta)$-approximate algorithms that use $\omega(\text{poly}(\log n))$ phases do exist.) This bound on the number of phases is essential for obtaining our additive approximation guarantees. It is open whether we can obtain a framework without additive error dependence on the number of phases. Also, independently, it is interesting to see whether non-private $(1+\eta)$-approximate k-core decomposition algorithms that take $\text{poly}(\log n)$ phases exist.

Then, we give an ε-edge DP densest subgraph algorithm based on the *non-private* parallel algorithm of Bahmani et al. [BGM14] together with the modifications made by Su and Vu [SV20] in their non-private distributed algorithm. Using this algorithm and our mechanism, we present a $\left(1+\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$-approximate ε-edge DP algorithm for the densest subgraph problem with runtime $O((n+m)\log^3 n)$ when $\eta > 0$ is constant. This improves on the multiplicative approximation factors of the previous (ε, δ)-edge DP $O\left(2+\eta, \frac{\log(n)\log(1/\delta)}{\varepsilon}\right)$-approximation results of Nguyen and Vullikanti [NV21] and the more recent ε-edge DP $\left(2, O\left(\frac{\log^{2.5}(n)\log(1/\sigma)}{\varepsilon}\right)\right)$-approximation algorithm of Farhadi et al. [FHS22] that obtains this approximation guarantee with probability $1-\sigma$. We achieve this improvement in the approximation factor with only an $O(\log^3 n)$ increase in the runtime. Furthermore, when $\eta > 0$ is constant, our algorithm matches the $((n+m)\log^2 n)$ runtime up to a $O(\log n)$ factor of the best known non-private algorithm of Chekuri et al. [CQT22], which obtains a $(1+\eta)$-approximate densest subgraph algorithm. We present our algorithm and its analysis in Section 5.

Theorem 3.4 (ε-Edge DP Densest Subgraph)**.** *There exists an* $\left(1+\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$*-approximate algorithm for the densest subgraph problem that is* ε*-edge DP and runs in* $O((n+m)\log^3 n)$ *worst-case time for constant* $\eta > 0$ *that returns an approximation factor within the stated bounds with high probability.*

Finally, we present our general framework. This framework is a formalization of a category of graph algorithms that we call ***locally adjustable***. We present a simple mechanism for converting locally adjustable graph algorithms into ε-edge DP and ε-LEDP algorithms. Our framework and privacy proofs are given in Section 6.

Three crucial observations about locally adjustable algorithms allow us to obtain private algorithms with small error. First, such algorithms proceed in several phases where the state of each node or edge is updated via a function that uses *only* information from its immediate neighbors and incident edges. We are thus able to bound the sensitivity of such functions by a small constant in edge-neighboring graphs. Second, these local functions often only require the *number* of neighbors and incident edges that satisfy a condition without requiring knowledge of the states of these neighbors. This property allows us to easily add noise via the geometric mechanism to the *count* of neighbors/incident edges that satisfy the condition. The noise can be viewed as either hiding actual edges or representing ***dummy edges*** which may additionally satisfy the condition. In edge-neighboring graphs G and G', a dummy edge incident to node v can account for the additional neighbor of v that is present in G' but not in G. Finally, we see that the additive error of each of our algorithms depends on its total number of ***phases*** (or the number of times the algorithm is run before returning an output). We show that state-of-the-art parallel and distributed algorithms naturally run in

poly($\log n$) phases, thus leading to only a poly($\log n$) factor in the additive error. It is as an interesting open question whether general utility guarantees can be obtained for our framework.

4 Local Algorithms for Core Decomposition, Low Out-Degree Ordering, and Densest Subgraph

In this section, we give local algorithms for core decomposition, low out-degree ordering, and densest subgraph. Our local algorithms are based on extending algorithms for a recently developed non-private batch-dynamic level data structure to the private setting. We start with our local core decomposition algorithm and its privacy guarantees and accuracy, and then present our local low out-degree ordering and densest subgraph algorithms.

4.1 ε-LEDP k-Core Decomposition

We present our algorithm in this section, provide its privacy analysis in Section 4.3, and finally analyze its approximation factor and round complexity in Section 4.4. Our LEDP algorithm is inspired by the sequential level data structure algorithms of [BHNT15, HNW20] and the parallel level data structure of [LSY+22] used to obtain the densest subgraphs and bounded degeneracy orientations of a graph in the dynamic setting. We crucially use the observation made by [LSY+22] that the longest path of sequential dependencies is $O(\log^2 n)$ for *any number* of updates. We use this, along with our new private procedures for releasing *multiple* outputs *simultaneously* for a phase at once, to prove both our privacy guarantees and our $O(\log^2 n)$ rounds, $O(1)$ node communication complexity bound. A simple extension allows us to obtain an LEDP algorithm that uses $O(\log n)$ rounds and $O(\log n)$ communication complexity per node.

Finally, we are able to improve the multiplicative approximation factor from $(4+\eta)$ in [LSY+22] to $(2+\eta)$ (see Footnote 1) because [LSY+22] is a dynamic algorithm while ours is a static algorithm. The additional factor of 2 was useful in reducing the parallel work in the fully batch-dynamic algorithm (when accounting for both insertions and deletions). In our static setting, we only need to maintain insertions (from inserting all the edges of the graph as a batch) which allows us to reduce the approximation factor by 2. However, in order to ensure our privacy guarantees in the LEDP model, we compute a new noise for every node. In the sequential computation setting, this incurs an additional factor of n in the running time, which also means that we no longer need the amortized analysis from previous works, since the additional factor of n dominates the running time. But in the distributed setting, we are still able to show that the worst-case number of rounds is $O(\log^2 n)$ (and $O(\log n)$ rounds with a modification).

Non-Private Dynamic Algorithms of [BHNT15, HNW20, LSY+22, SCS20] The level data structure of [BHNT15, HNW20] partitions the nodes of the graph into $O(\log^2 n)$ *levels*. The levels are partitioned into *groups* of $O(\log n)$ levels each. Nodes move up and down the levels according to a set of rules on the induced degree of a node i with respect to the number of neighbors in the same or higher level than i (sometimes the set of nodes in the level just below the level of i is also considered). Specifically, if the induced degree is too high, i moves one level up; if the induced degree is too low, i moves one level down. One crucial aspect of this algorithm is that *a node only needs to know the levels of its immediate neighbors.*

In addition to the above structure, we use a crucial aspect of the algorithm of [LSY+22] that allows us to move nodes from the same level *simultaneously* without causing additional nodes below them to move. Since we only have $O(\log^2 n)$ levels, we can process them in a bottom-up fashion while accounting for all vertices in the current level of the iteration simultaneously, achieving the $O(\log^2 n)$ round complexity of our ε-LEDP algorithm. A further modification that accounts for different groups *simultaneously* allows us to decrease the round complexity.

4.2 Detailed Algorithm

Our algorithms use the notation shown in Table 1. For the remainder of this section, $\log n$ means $\log_{(1+\psi)} n$ for a parameter $\psi > 0$ that affects our approximation factor. There are $4\log^2 n$ levels in the structure. As in previous works [BHNT15, HNW20, LSY+22], we call this structure a *level data structure*. Our descriptions

Algorithm 1: ε-LEDP Decomposition and Ordering

1 **Input:** Adjacency lists $(\mathbf{a}_1, \ldots, \mathbf{a}_n)$, constant $\eta \in (0,1)$, and privacy parameter $\varepsilon \in (0,1)$.
2 **Output:** ε-LEDP $\left(2+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate core numbers and low out-degree ordering of each node in G.
3 **Function** `LEDPCoreDecomp`$((\mathbf{a}_1, \ldots, \mathbf{a}_n), \varepsilon, \eta)$
4 Set $\psi = 0.5$ and $\lambda = \frac{2}{9}(2\eta - 5)$.
5 Curator initializes $L_0, \ldots, L_{4\log^2 n - 1}$ with $L_r[i] \leftarrow 0$ for every $i \in [n], r \in [0, \ldots, 4\log^2 n - 1]$.
6 **for** $r = 0$ *to* $4\log^2 n - 2$ **do**
7 **for** $i = 1$ *to* n **do**
8 $L_{r+1}[i] \leftarrow L_r[i]$.
9 **if** $L_r[i] = r$ **then**
10 Let $\mathcal{U}_i$ be the number of neighbors $j \in \mathbf{a}_i$ where $L_r[j] = r$.
11 Sample $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$.
12 Compute $\widehat{\mathcal{U}}_i \leftarrow \mathcal{U}_i + X$.
13 **if** $\widehat{\mathcal{U}}_i > (1+\psi)^{\mathcal{F}(r)}$ **then**
14 i **releases** 1.
15 $L_{r+1}[i] \leftarrow L_r[i] + 1$. ▷ Curator moves i up one level.
16 **else**
17 i **releases** 0.
18 Curator publishes L_{r+1}.
19 Curator calls $C \leftarrow$ `EstimateCoreNumbers`$(L_{4\log^2 n - 1}, \lambda, \psi)$.
20 Curator orders nodes in D by $L_{4\log^2 n - 1}$ (from smaller to larger) breaking ties by node index.
21 **Return** (C, D).

use terminology from these previous works. However, because we are in the static setting, we are able to simplify the algorithm of [LSY+22] in some ways by considering the input graph as a single batch of edge insertions.

Levels in the level data structure are partitioned into $2\log n$ ***groups*** of equal size. Each group g_i contains $2\log n$ *consecutive* levels. We number the levels starting with 0 as the bottommost level and $4\log^2 n - 1$ as the topmost level. Each group g_i has an associated index i, and contains levels in $[i \cdot 2\log n, (i+1) \cdot 2\log n)$. Let the group index that a level r belongs to be $\mathcal{F}(r)$. In other words, $\mathcal{F}(r) = f$ if $r \in g_f$, which can be computed as $\mathcal{F}(r) = \lfloor r/2\log n \rfloor$. We denote the level of a node i as level(i). Finally, a node is ***unsettled*** if it must move to a higher level.

We now describe our algorithm. The pseudocode for our algorithm is given in Algorithm 1. The algorithm is given a sequence of adjacency lists, $(a_1, \ldots, a_n)$, constant parameters $\lambda, \psi > 0$ that will determine the approximation factor, and privacy parameter $\varepsilon \in (0,1)$ as input. The algorithm outputs $\left(2+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate core numbers and a low out-degree ordering with the same guarantee on the out-degree. The approximation parameter η is determined from λ and ψ. All nodes start in level 0 (Line 5). Throughout the algorithm, the curator maintains and publishes the current levels of the nodes in a set of $4\log^2 n$ lists, each of size n where the i-th index of a list contains the level of node i (Line 18).

First, the curator iterates through the levels starting from level 0 to level $4\log^2 n - 1$ (Line 6). Let r be the current level. The curator asks each node, i, in level r to compute its *noisy* number of neighbors in level r, denoted $\widehat{\mathcal{U}}_i$ (Line 9). To compute its $\widehat{\mathcal{U}}_i$, node i first computes $\mathcal{U}_i$ using the most recently published levels of its neighbors (Line 10) where $\mathcal{U}_i$ is the number of its neighbors in level r. If this is the first round of the algorithm, then all neighbors of i are on its level, level 0. Then, i computes $\widehat{\mathcal{U}}_i \leftarrow \mathcal{U}_i + X$, where $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$ denotes a noise drawn i.i.d. from the symmetric geometric distribution with parameter $\varepsilon/(8\log^2 n)$ (Line 12). The curator moves i *up a level* (to level $r+1$) if and only if the released bit is 1 (i.e., when $\widehat{\mathcal{U}}_i > (1+\psi)^{\mathcal{F}(r)}$ in Lines 13 to 15). (The curator performs this step for each node in level r.) Otherwise, i releases 0 (Line 17) and it stays in the same level. Then, the curator publicizes a new list, L_{r+1}, that contains the new levels of each node (Line 18). If the node does not move up, then its old level is included in L_{r+1} (Line 8). This repeats in subsequent rounds until we reach the final level $4\log^2 n - 1$.

Algorithm 2: Estimate Core Number [LSY+22]

Function `EstimateCoreNumbers`(L, λ, ψ)
  for $i = 1$ *to* n **do**
    $\hat{k}(i) \leftarrow (2+\lambda)(1+\psi)^{\max\left(\left\lfloor \frac{L(i)+1}{4\lceil \log_{1+\psi} n \rceil} \right\rfloor - 1, 0\right)}$.
  Return $\{(i, \hat{k}(i)) : i \in [n]\}$.

After processing the final level, $4\log^2 n - 1$, the curator estimates the core numbers of nodes using their levels. This computation is shown in Algorithm 2. We use Definition 3.14 of [LSY+22] to calculate this estimate. Intuitively, the core number estimate for node i is calculated to be $(1+\psi)^g$, where g is the maximum group index, where $L_{4\log^2 n-1}[i]$ is the topmost level in group g or is higher than the topmost level in group g.

We provide some brief intuition for the privacy of this algorithm. Each node moves up at most $4\log^2 n - 1$ levels. Thus, there will be at most $4\log^2 n - 1$ rounds of communication. We show that the *sensitivity* of $\mathcal{U}_i$ for any $0 \le r \le 4\log^2 n - 1$ of any node i is 1; not only that, but the sensitivity of the vector of these values is 2 for edge-neighboring graphs. Hence, we add sufficient noise each round to maintain LEDP using the geometric (mechanism Definition 2.13). We show that by the adaptive composition theorem (Theorem 2.15)over $O(\log^2 n)$ rounds that our algorithm is a ε-LEDP algorithm. This is a simplification of our privacy proofs; full details can be found in Section 4.3.

Core Number Estimation Our algorithms here use the core number estimation algorithm presented in Algorithm 2, which takes r as input and computes the estimate of the core number of all nodes according to r. This estimate is denoted $\hat{k}(i)$ for each node i.

Low Out-Degree Ordering The ordering of the nodes is determined first by sorting its final level (contained in $L_{4\log^2 n-1}$), from smallest level to largest, and then breaking ties using the nodes' indices.

4.3 LEDP Guarantees

In this section, we prove that our algorithm is ε-LEDP (see Definition 2.9). For clarity, we specifically prove the ε-edge LEDP of our k-core decomposition and related algorithms here. Later, we show that the algorithms presented here can be generalized to our locally-adjustable algorithms framework, which may significantly simplify the privacy analysis of these algorithms and similar algorithms.

Theorem 4.1. *Algorithm 1 is ε-LEDP.*

Proof. We show that Algorithm 1 can be implemented so that it accesses its input only via local randomizers. In each round r, each node in level r computes its *induced degree* (the number of its neighbors that are also at level r) with added geometric noise in Line 14. We can implement this step with the local randomizer R which takes as input an adjacency list, $\mathbf{a}$, a level r, and a list of levels of all nodes, L_r and computes $R(\mathbf{a}, r, L_r) = \sum_{j \in \mathbf{a}} [L_r[j] = r] + X$ where $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$. This randomizer is run independently by each party i on its adjacency list $\mathbf{a}_i$ and the public information r and L_r. Algorithm R uses the geometric mechanism to obtain the number of its neighbors on r. For any two edge-neighboring adjacency lists, $\mathbf{a}$ and $\mathbf{a}'$, the corresponding sums $\sum_{j \in \mathbf{a}} [L_r[j] = r]$ and $\sum_{j \in \mathbf{a}'} [L_r[j] = r]$ differ by 1. Thus, this sum function has sensitivity 1 and by the privacy of the geometric mechanism is $\frac{\varepsilon}{8\log^2 n}$-private. Lines 13, 14, 16 and 17 post-processes the output of the geometric mechanism and so, by Theorem 2.17, R is a $\frac{\varepsilon}{8\log^2 n}$-local randomizer by Lemma 2.14. Using this released information, the curator then computes a new public set of levels L_{r+1}.

For each edge $\{i, j\}$, we apply two $\frac{\varepsilon}{8\log^2 n}$-local randomizers to each endpoint over $4\log^2 n$ rounds. By the adaptive composition theorem (Theorem 2.15), group privacy (Lemma 2.16) and because privacy is preserved after post-processing (Theorem 2.17), Algorithm 1 is ε-LEDP because it satisfies Definition 2.9. ∎

4.4 Approximation Guarantees

In this section, we calculate the approximation factors produced by Algorithm 1. We first note that *in expectation*, we return a $(2+\eta, 0)$-approximation (where the additive error is 0) for the core numbers for any

constant $\eta > 0$.

Lemma 4.2. *Algorithm 1 returns a $(2+\eta, 0)$-approximation for the core numbers and low out-degree ordering, in expectation.*

Proof. First, for any random variable $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$, the expectation $\mathbb{E}[X] = 0$ by definition of the symmetric geometric distribution. Thus, $\mathbb{E}[\widehat{\mathcal{U}}_i] = \mathcal{U}_i$ for all i and the lemma follows from a slightly modified version of the proof of Lemma 5.13 of [LSY+22] given in Appendix B. ☐

In the remainder of this section, we show that we also obtain an additive error of $O\left(\frac{\log^3 n}{\varepsilon}\right)$ where the error is bounded by $O\left(\frac{\log^3 n}{\varepsilon}\right)$ whp.

First, we show that Algorithm 1 maintains the following two invariants (which we prove in Lemma 4.5). These invariants are crucial in both our bounds on the approximation factor as well as our message complexity analysis. For the following two invariants, let Z_r be the set of nodes in levels r and above and V_r be the set of nodes in level r. Recall as before that the bottommost level is level 0 and the topmost level is $4\log^2 n - 1$. All node levels are taken from $L_{4\log^2 n - 1}$ in the final computation of approximate core numbers.

Invariant 1 (Degree Upper Bound)**.** *If node $i \in V_r$ and level $r < 4\log^2 n - 1$, then i has at most $(1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$ neighbors in Z_r whp.*

Invariant 2 (Degree Lower Bound)**.** *If node $i \in V_r$ and level $r > 0$, then i has at least $(1+\psi)^{\mathcal{F}(r-1)} - \frac{8c\log^3 n}{\varepsilon} - 1$ neighbors in Z_{r-1} whp.*

Before we prove our algorithm maintains these invariants, we first prove the following lemma for each node i that moves up a level.

Lemma 4.3. *Let $\mathcal{U}_i$ be the number of neighbors of node i in level r where the levels of nodes are taken from L_r. Each node i that moves up one level (from level r to level $r+1$) in Algorithm 1 has $\mathcal{U}_i \geq (1+\psi)^{\mathcal{F}(r)} - \frac{8c\log^3 n}{\varepsilon} - 1$ whp.*

Proof. Suppose for contradiction a node i moves up one level (from level r to level $r+1$) and has $\mathcal{U}_i < (1+\psi)^{\mathcal{F}(r)} - \frac{8c\log^3 n}{\varepsilon} - 1$ with probability $> \frac{1}{n^c}$. This means that it sampled a noise $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$, where $X > \frac{8c\log^3 n}{\varepsilon}$ with probability $> \frac{1}{n^c}$. This is because $\mathcal{U}_i + X$ must exceed the $(1+\psi)^{\mathcal{F}(r)}$ threshold in order to move up. The probability that i sampled noise $X > \frac{8c\log^3 n}{\varepsilon}$ from $\mathsf{Geom}(\varepsilon/(8\log^2 n))$ is $< \frac{1}{n^c}$ by Lemma 2.12. This is a contradiction so $X > \frac{8c\log^3 n}{\varepsilon}$ could *not* have been sampled with probability greater than $\frac{1}{n^c}$. ☐

On the flip side, we can also bound the degree values of nodes that *do not move*.

Lemma 4.4. *Let $\mathcal{U}_i$ be as defined in Lemma 4.3. Each node i that does not move up from a level r in Algorithm 1 after computing $\mathcal{U}_i$ (where the levels of nodes used to compute $\mathcal{U}_i$ are taken from L_r) has $\mathcal{U}_i \leq (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$ whp.*

Proof. As in the proof of Lemma 4.3, we prove this lemma via contradiction. First, suppose for contradiction that i is currently in level r and does not move up, and $\mathcal{U}_i > (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$ with probability $> \frac{1}{n^c}$. Node i does not move up when $\mathcal{U}_i > (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$ if it chooses a noise $X \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$ with value $X < -\frac{8c\log^3 n}{\varepsilon}$. The same calculation as that performed in the proof of Lemma 4.3 shows that with probability $< 1/n^c$, node i does not move up if $\mathcal{U}_i > (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$. This contradicts our assumption that $\mathcal{U}_i > (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1$ and does not move up with probability $> \frac{1}{n^c}$ since we showed that this occurs with probability at most $\frac{1}{n^c}$. ☐

We use Lemmas 4.3 and 4.4 to prove the following lemma, which shows that Invariant 1 and Invariant 2 are maintained by our algorithm.

Lemma 4.5. *Given an input graph $G = ([n], E)$, Invariant 1 and Invariant 2 are satisfied at the end of Algorithm 1 whp.*

Proof. For each node i in the graph that is in the current level r, a noise X_i is independently sampled $X_i \sim \mathsf{Geom}(\varepsilon/(8\log^2 n))$ at the beginning of that round. We first make a few observations for when a node i is in level 0 or $4\log^2 n - 1$. If i did not move up from level 0, then Invariant 2 is trivially satisfied. If i moves up to level $4\log^2 n - 1$, then Invariant 1 is trivially satisfied. By Lemma 4.4, any i which does not move to a higher level than $r \in [0, 4\log^2 n - 1)$ has $\mathcal{U}_i \le (1+\psi)^{\mathcal{F}(L_r[i]))} + \frac{8c\log^3 n}{\varepsilon} + 1$ whp in round r. Furthermore, $|\mathbf{a}_i \cap Z_{L_{4\log^2 n-1}[i]}|$ cannot be greater than $\mathcal{U}_i$ since only nodes in level r can move to a higher level than r in round r. Thus, $|\mathbf{a}_i \cap Z_{L_{4\log^2 n-1}[i]}| \le (1+\psi)^{\mathcal{F}(L_r[i]))} + \frac{8c\log^3 n}{\varepsilon} + 1$ and Invariant 1 is satisfied in this case. To prove that Invariant 2 is also satisfied for any node in level $r \in (0, 4\log^2 n - 1]$ at the end of the algorithm, first observe that because nodes only move up in our algorithm, never down, $|\mathbf{a}_i \cap Z_{r'-1}|$ can also never decrease for *any node that stays in* r' as its neighbors move up for any level r'. Thus, when i moved up from $r-1$ to r, we have $\mathcal{U}_i \ge (1+\psi)^{\mathcal{F}(r-1)} - \frac{8c\log^3 n}{\varepsilon} - 1$ with probability $\ge 1 - \frac{1}{n^c}$ by Lemma 4.3. Since $|\mathbf{a}_i \cap Z_{r-1}|$ cannot decrease if i stays in level r and we know that i is in level r at the end of the algorithm, it is lower bounded by $|\mathbf{a}_i \cap Z_{L_{4\log^2 n-1}[i]}| = |\mathbf{a}_i \cap Z_{r-1}| = \mathcal{U}_i \ge (1+\psi)^{\mathcal{F}(r-1)} - \frac{8c\log^3 n}{\varepsilon} - 1$ and Invariant 2 is also satisfied. This proves that both invariants hold for all nodes in all levels with probability at least $1 - \frac{1}{n^d}$ for any constant $d \ge 1$ by the union bound over $4\log^2 n$ levels and n nodes when $c \ge 1$ is a sufficiently large constant. □

The theorem below is the main result of this section. The proof method is inspired by the proof of the approximation factor given in the algorithm of Liu et al. [LSY+22]. However, our proof is more involved because of the added noise and requires new insights in order to obtain our additive approximation bound. First, our algorithm is randomized. Since we move each node at most $O(\log^2 n)$ times, we can show via the union bound that our guarantees hold with high probability. Second, the additive noise from Invariant 1 and Invariant 2 requires us to re-perform the analysis taking this noise into account.

We are now ready to prove the approximation factor of our LEDP algorithm in this section. The proof of Theorem 4.7 uses the following folklore lemma, the proof of which follows immediately from the traditional peeling method which provides the exact core number of each node in a static graph. Note that although some previous literature use the term *peel* for the below process, we instead use the phrase *prune* to distinguish from the classic peeling algorithm of Matula and Beck [MB83].

Lemma 4.6 (Folklore)**.** *Suppose we perform the following* peeling *procedure. Provided an input graph $G = (V, E)$, we iteratively remove (prune) nodes with degree $\le d^*$ for some $d^* > 0$ until no nodes with degree $\le d^*$ remain. Then, $k(i) \le d^*$ for each removed node v and all remaining nodes w (not pruned) have core number $k(w) > d^*$.*

Theorem 4.7. *Our static LEDP algorithm (Algorithm 1) on input $G = ([n], E)$ gives $(2+\eta, O(\log^3 n/\varepsilon))$-approximate core numbers whp.*

Proof. In this proof, when we refer to the level of a node i, we mean the level of i in $L_{4\log^2 n-1}$. Using notation from previous work, let $\hat{k}(i)$ be the core number estimate of i and $k(i)$ be the core number of i. First, we show that

$$\text{if } \hat{k}(i) \le (2+\lambda)(1+\psi)^{g'}, \text{ then } k(i) \le (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1 \tag{1}$$

with probability at least $1 - \frac{1}{n^{c-1}}$ for any group $g' \ge 0$ and for any constant $c \ge 2$. Let $T(g')$ be the topmost level of group g'. In order for $(2+\lambda)(1+\psi)^{g'}$ to be the estimate of i's core number, the level of i is bounded by $T(g') \le \text{level}(i) \le T(g'+1) - 1$. Let r be i's level (in $L_{4\log^2 n-1}$). By Invariant 1, if $r < T(g'+1)$, then $|\mathbf{a}_i \cap Z_r| \le (1+\psi)^{\mathcal{F}(r)} + \frac{8c\log^3 n}{\varepsilon} + 1 \le (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$ with probability at least $1 - \frac{1}{n^c}$ for any constant $c \ge 1$. Furthermore, each node w at the same or lower level $r' \le r$ has $|\mathbf{a}_w \cap Z_{r'}| \le (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$, also by Invariant 1.

Suppose we perform the following iterative procedure: starting from level $r = 0$, remove all nodes in level r during this turn and set $r \leftarrow r+1$ for the next turn. Using this procedure, the nodes in level 0 are removed in the first turn, the nodes in level 1 are removed in the second turn, and so on until the graph is empty. Let $d_r(i)$ be the induced degree of any node i after the removal in the $(r-1)$-st turn and prior to the

removal in the r-th turn. Since we showed above that $|\mathbf{a}_i \cap Z_r| \leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$ for any node i at level $r < T(g'+1)$, node i on level $r < T(g'+1)$ during the r-th turn has $d_r(v) \leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$. Thus, when i is removed in the r-th turn, it has degree $\leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$. Since all nodes removed before i also had degree $\leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$ when they were removed, by Lemma 4.6, node i has core number $k(i) \leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$ with probability $\geq 1 - \frac{1}{n^c}$. By the union bound over all nodes, this proves that $k(i) \leq (1+\psi)^{g'+1} + \frac{8c\log^3 n}{\varepsilon} + 1$ for all $i \in [n]$ with $\hat{k}(i) \leq (2+\lambda)(1+\psi)^{g'}$ for all constants $c \geq 2$ with probability at least $1 - \frac{1}{n^{c-1}}$.

Now we prove our lower bound on $\hat{k}(i)$. We prove that for any $g' \geq 0$, either the approximation falls within our additive bounds or,

$$\text{if } \hat{k}(i) \geq (1+\psi)^{g'}, \text{ then } k(i) \geq \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)} \tag{2}$$

for all nodes i in the graph with probability at least $1 - \frac{1}{n^{c-1}}$ for any constant $c \geq 2$. We assume for contradiction that with probability $> \frac{1}{n^{c-1}}$ there exists a node i where $\hat{k}(i) \geq (1+\psi)^{g'}$ and $k(i) < \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$. This, in turn, implies that $k(i) \geq \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$ for all $i \in [n]$ and $\hat{k}(i) \geq (1+\psi)^{g'}$ with probability $< 1 - \frac{1}{n^{c-1}}$. To consider this case, we use the *pruning* process defined in Lemma 4.6. In the below proof, let $d_S(i)$ denote the induced degree of node i in the subgraph induced by nodes in S. For a given subgraph S, we *prune* S by repeatedly removing all nodes $i \in S$ whose $d_S(i) < \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$. As in the proof of Lemma 5.12 of [LSY+22], we consider levels from the same group g' since levels in groups lower than g' will also have smaller upper bound cutoffs, leading to an easier proof. Let j be the number of levels below level $T(g')$. We prove via induction that the number of nodes pruned from the subgraph induced by $Z_{T(g')-j}$ must be at least

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} \left(\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right)\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)\right). \tag{3}$$

We first prove the base case when $j = 1$. In this case, we know that for any node i in level $T(g')$, it holds that $d_{Z_{T(g')-1}}(i) \geq (1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1$ with probability $\geq 1 - \frac{1}{n^c}$ by Invariant 2. Taking the union bound over all nodes in level $T(g')$ shows that this bound holds for all nodes $i \in [n]$ with probability at least $1 - \frac{1}{n^{c-1}}$. All below expressions hold with probability at least $1 - \frac{1}{n^{c-1}}$; for simplicity, we omit this phrase when giving these expressions. In order to prune i from the graph, we must prune at least

$$\begin{aligned}
&\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right) - \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)} \\
&= \left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right) \cdot \left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right).
\end{aligned}$$

neighbors of i from $Z_{T(g')-1}$. We must prune at least this many neighbors in order to reduce the degree of i to below the cutoff for pruning a vertex (as we show more formally below). In the case when $(1+\psi)^{g'} \leq 4(\frac{8c\log^3 n}{\varepsilon} - 1)$, then our original approximation statement in the lemma is trivially satisfied because the core number is always non-negative and so even if $k(i) = 0$, this is still within our additive approximation bounds. Hence, we only need to prove the case when $(1+\psi)^{g'} > 4\left(\frac{8c\log^3 n}{\varepsilon} + 1\right)$.

Then, if fewer than $\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right)\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ neighbors of i are pruned from the graph, then i is not pruned from the graph. If i is not pruned from the graph, then i is part of a $\left(\frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}\right)$-core (by Lemma 4.6), then by what we showed above using Invariant 2, $k(i) \geq \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$ for all $i \in [n]$ and $\hat{k}(i) \geq (1+\psi)^{g'}$ with probability $\geq 1 - \frac{1}{n^{c-1}}$, a contradiction with our assumption. Thus, it must be

the case that there exists at least one i where at least $\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right)\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ neighbors of i are pruned in $Z_{T(g')-1}$. For our induction hypothesis, we assume that at least the number of nodes as indicated in Eq. (3) is pruned for j and prove this for $j+1$.

Each node w in levels $T(g') - j$ and above has $d_{Z_{T(g')-j-1}}(w) \geq (1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1$ by Invariant 2 (recall that all j levels below $T(g')$ are in group g'). For simplicity of expression, we denote $J \triangleq \left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right)\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$. Then, in order to prune the $\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J$ nodes by our induction hypothesis, we must prune at least

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J \cdot \left(\frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{2}\right) \tag{4}$$

edges where we "charge" the edge to the endpoint that is pruned last. We use the phrase *charge* to mean that if an edge (u, v) is charged to endpoint v, then the edge needs to be pruned in order for the endpoint v to be pruned. (Note that we actually need to prune at least $\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right) \cdot \left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ edges per pruned node as in the base case but $\frac{\left((1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1\right)}{2}$ lower bounds this amount.) Each pruned node prunes less than $\frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$ edges. Thus, using Eq. (4), the number of nodes that must be pruned from $Z_{T(g')-j-1}$ is

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J \cdot \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{2\left(\frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}\right)} = \left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j} J. \tag{5}$$

Eq. (5) proves our induction step. Using Eq. (3), the number of nodes that must be pruned from $Z_{T(g')-2\log_{(2+\lambda)(1+\psi)/2}(n)}$ is greater than n since $J \geq 1$ by our assumption that $(1+\psi)^{g'} > 4\left(\frac{8c\log^3 n}{\varepsilon} + 1\right)$:

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{2\log_{(2+\lambda)(1+\psi)/2}(n)} \cdot J \geq n^2. \tag{6}$$

Thus, at $j = 2\log_{(2+\lambda)(1+\psi)/2}(n)$, we run out of nodes to prune. We have reached a contradiction as we require pruning greater than n nodes with probability at least $\frac{1}{n^{c-1}} \cdot \left(1 - \frac{1}{n^{c-1}}\right) > 0$ via the union bound over all nodes where $\hat{k}(i) \geq (1+\psi)^{g'}$ and using our assumption that with probability $> \frac{1}{n^{c-1}}$ there exists an $i \in [n]$ where $k(i) < \frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$. This contradicts with the fact that more than n nodes can be pruned with 0 probability.

From Eq. (1), we can first obtain the inequality $k(i) \leq \hat{k}(i) + \frac{8c\log^3 n}{\varepsilon} + 1$ since this bounds the case when $\hat{k}(i) = (2+\lambda)(1+\psi)^{g'}$; if $\hat{k}(i) < (2+\lambda)(1+\psi)^{g'}$ then the largest possible value for $\hat{k}(i)$ is $(2+\lambda)(1+\psi)^{g'-1}$ by Algorithm 2 and we can obtain the tighter bound of $k(i) \leq (1+\psi)^{g'} + \frac{8c\log^3 n}{\varepsilon} + 1$. We can substitute $\hat{k}(i) = (2+\lambda)(1+\psi)^{g'}$ since $(1+\psi)^{g'+1} < (2+\lambda)(1+\psi)^{g'}$ for all $\psi \in (0,1)$ and $\lambda > 0$. Second, from Eq. (2), for any estimate $(2+\lambda)(1+\psi)^{g}$, the largest g' for which this estimate has $(1+\psi)^{g'}$ as a lower bound is $g' = g + \lfloor \log_{(1+\psi)}(2+\lambda) \rfloor \geq g + \log_{(1+\psi)}(2+\lambda) - 1$. Substituting this g' into $\frac{(1+\psi)^{g'} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$ results in

$$\frac{(1+\psi)^{g+\log_{(1+\psi)}(2+\lambda)-1} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)} = \frac{\frac{(2+\lambda)(1+\psi)^{g}}{1+\psi} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)} = \frac{\frac{\hat{k}(i)}{1+\psi} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}.$$

Thus, we can solve $k(i) \geq \frac{\frac{\hat{k}(i)}{1+\psi} - \frac{8c\log^3 n}{\varepsilon} - 1}{(2+\lambda)(1+\psi)}$ and $k(i) \leq \hat{k}(i) + \frac{8c\log^3 n}{\varepsilon} + 1$ to obtain $k(i) - \frac{8c\log^3 n}{\varepsilon} - 1 \leq \hat{k}(i) \leq ((2+\lambda)(1+\psi)^2)k(i) + \frac{8c(1+\psi)\log^3 n}{\varepsilon} + (1+\psi)$. Simplifying, we obtain

$$k(i) - O(\log^3(n)/\varepsilon) \leq \hat{k}(i) \leq (2+\lambda)(1+\psi)^2 k(i) + O(\log^3(n)/\varepsilon)$$

which is consistent with the definition of a $(2+\eta, O(\log^3(n)/\varepsilon))$-approximation algorithm for core number for any constant $\eta > 0$ and appropriately chosen constants $\lambda, \psi \in (0,1)$ that depend on η. ☐

Algorithm 3: ε-LEDP k-Core Decomposition (Smaller Rounds)

Input: Adjacency lists $(\mathbf{a}_1, \ldots, \mathbf{a}_n)$, constant $\eta \in (0,1)$, and privacy parameter $\varepsilon \in (0,1)$.

Output: ε-LEDP $\left(2+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate core numbers of each node in G.

Function `LEDPSmallRoundsCoreDecomposition`$((\mathbf{a}_1, \ldots, \mathbf{a}_n), \varepsilon, \eta)$

Set $\psi = 0.5$ and $\lambda = \frac{2}{9}(2\eta - 5)$.

Curator initializes $L_0^0, L_1^0, \ldots, L_{2\log n-1}^0, \ldots, L_0^{2\log n-1}, \ldots, L_{2\log n-1}^{2\log n-1}$ with $L_r^g[i] \leftarrow 0$ for every $i \in [n]; r, g \in [0, \ldots, 2\log n - 1]$.

for $r = 0$ *to* $2\log n - 2$ **do**

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

    Initialize $A_i[g] \leftarrow 0$ for every $g \in [0, \ldots, 2\log n - 1]$.

    for $g = 0$ *to* $2\log n - 1$ **do**

      $L_{r+1}^g[i] \leftarrow L_r^g[i]$.

      if $L_r^g[i] = r$ **then**

        Let $\mathcal{U}_{i,g}$ be the number of neighbors $j \in \mathbf{a}_i$ where $L_r^g[j] = r$.

        Sample $X \leftarrow \mathsf{Geom}(\varepsilon/(8\log^2 n))$.

        Compute $\widehat{\mathcal{U}}_{i,g} \leftarrow \mathcal{U}_{i,g} + X$.

        if $\widehat{\mathcal{U}}_{i,g} > (1+\psi)^g$ **then**

          $A_i[g] \leftarrow 1$.

    i **releases** A_i.

    $L_{r+1}^g[i] \leftarrow L_r^g[i] + 1$ for every i, g where $A_i[g] = 1$. ▷ Curator moves i up in group g.

  Curator publishes L_{r+1}^g for every $g \in [0, \ldots, 2\log n - 1]$.

Curator calls $C \leftarrow$ `EstimateSmallRoundsCoreNumbers`$(L_{2\log n-1}^0, \ldots, L_{2\log n-1}^{2\log n-1}, \lambda, \psi)$ [Algorithm 4].

return C.

Algorithm 4: Small Rounds Estimate Core Number

Function `EstimateSmallRoundsCoreNumbers`$(L_{2\log n-1}^0, \ldots, L_{2\log n-1}^{2\log n-1}, \lambda, \psi)$

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

    Let g' be the largest group number where $L_{2\log n-1}^{g'}[i] = 2\log n - 1$ or 0 if no group satisfies this condition.

    $\hat{k}(i) \leftarrow (2+\lambda)(1+\psi)^{g'}$.

  Return $\{(i, \hat{k}(i)) : i \in [n]\}$.

4.5 Reducing the Number of Rounds

In this section, we describe how to reduce the number of rounds of our procedure given in Algorithm 1 to $O(\log n)$. Recall that the previous algorithm given in Algorithm 1 consists of $O(\log n)$ groups each with a set of $O(\log n)$ levels that are stacked on top of one another. The intuition behind our modified procedure lies in separating out the groups in our level data structure into separate duplicate copies of the structure, each with $O(\log n)$ levels. Then, the procedure moves each node *simultaneously* up the levels in all groups. This increases the node communication complexity by a factor of $O(\log n)$. A similar procedure to the above is given for our densest subgraph algorithm in Algorithm 5. We describe our modified algorithm in Algorithm 3.

In Algorithm 3, the curator first initializes a list for each level and group (Line 5) which contains the levels of each node in each of the groups (Line 11) with cutoffs determined by the group number. Then, we iterate level by level (Line 6) for each of the $O(\log n)$ groups simultaneously. For each level r in the iteration, each node i (Line 7) initializes an array A_i which keeps track of whether i moves up a level in group g (Line 8). As before, the curator first sets the next levels of each node in each of the groups to equal the current level of the node (Line 10). If i is in the current level r of the iteration in group g (Line 11), then i computes its induced degree among its neighbors also in level r using the published levels from the

previous iteration (Line 12). Then, it samples a noise from the symmetric geometric distribution (Line 13). If the induced degree plus the sampled noise exceeds the threshold for the group (Line 15), then i adds a 1 for the corresponding group g into A_i, indicating that i moves up a level in group g (Line 16). After performing this check for all of the groups, i then releases A_i (Line 16). The curator then uses the released A_i to determine if i moves up one level in each group (Line 18) and publishes the list of new levels for the next iteration (Line 19). After all the rounds are completed, the curator uses all of the published levels for the *last iteration* for each of the groups to compute the core number estimates (Line 20) using Algorithm 4. In Algorithm 4, each i computes the largest g' where $L^{g'}_{2\log n-1}[i] = 2\log n - 1$ (Line 3); in other words, each i finds the largest g' where i is on the topmost level of group g' after finishing all the iterations of the algorithm. Using g', node i computes the estimate $\hat{k}(i)$ as before (Line 4). The core estimates for all nodes are then returned by the procedure (Line 5).

The privacy of Algorithm 3 follows using the same local randomizers as in the proof of Theorem 4.1. The proof of the approximation factor follows almost identically from the proof of Theorem 4.7 using the modified invariants below (changes are underline). Since only minor changes to the approximation proofs are necessary, we do not repeat the proofs again here. Let V^g_r be the set of nodes i where $L^g_{2\log n-1}[i] = 2\log n-1$ and $Z^g_r = \bigcup_{r'\geq r} V^g_{r'}$.

Invariant 3 (Degree Upper Bound). *For all groups $g \in \{0,\ldots,2\log n-1\}$, if node $i \in V^g_r$ and level $r < 2\log n - 1$, then i has at most $(1+\psi)^g + \frac{8c\log^3 n}{\varepsilon} + 1$ neighbors in Z^g_r whp.*

Invariant 4 (Degree Lower Bound). *For all groups $g \in \{0,\ldots,2\log n-1\}$, if node $i \in V^g_r$ and level $r > 0$, then i has at least $(1+\psi)^g - \frac{8c\log^3 n}{\varepsilon} - 1$ neighbors in Z^g_{r-1} whp.*

4.6 Low Out-Degree Ordering and Densest Subgraph

In this section, using Algorithm 1, we provide results for low out-degree ordering of the nodes in the graph. The procedure that outputs the low out-degree ordering is provided in Line 20 of Algorithm 1. The ordering that we study in this paper is a private, approximate degeneracy ordering of the nodes. Recall in Algorithm 1, we calculate the approximate low out-degree ordering, D, by ordering the nodes starting from level 0 and going up. For nodes in the same level, we order the nodes according to their index. In other words, node i is earlier in the ordering than j if and only if $L_{4\log^2 n-1}(i) < L_{4\log^2 n-1}(j)$ or $L_{4\log^2 n-1}(i) = L_{4\log^2 n-1}(j)$ and $i < j$.

We now prove that our algorithm is also a $(2+\eta, O(\log^3(n)/\varepsilon))$-approximation algorithm for low out-degree ordering.

Lemma 4.8. *Our static LEDP algorithm (Algorithm 1) gives an $(2+\eta, O(\log^3(n)/\varepsilon))$-approximate low out-degree ordering with probability at least $1-\frac{1}{n^c}$ for any constants $\eta > 0, c \geq 1$.*

Proof. By Invariant 1, if we order the nodes as described in Algorithm 1, then each node i has degree at most $(1+\psi)^{\mathcal{F}(L_{4\log^2 n-1}[i])} + O(\log^3(n)/\varepsilon)$. By Theorem 4.7, the approximation on $d = \max_{i\in[n]}\{k(i)\}$ is upper bounded by $(2+\lambda)(1+\psi)^2 d + O\left(\log^3(n)/\varepsilon\right)$ Then, the highest group that any node j can be in has index

$$g = \log_{(1+\psi)}((2+\lambda)(1+\psi)^2 d + O(\log^3(n)/\varepsilon)) + 1,$$

where the additional 1 at the end comes from the fact that we take the maximum group number g where the topmost level of g is below or at $L_{4\log^2 n-1}[j]$. Within group g, the out-degree of any node in that group is upper bounded by $(1+\psi)^g + O(\log^3(n)/\varepsilon)$ with high probability by Invariant 1.

Simplifying by substituting the value for g,

$$\begin{aligned}
&(1+\psi)^{\log_{(1+\psi)}\left((2+\lambda)(1+\psi)^2 d + O(\log^3(n)/\varepsilon)\right)+1} + O(\log^3(n)/\varepsilon)\\
&= (1+\psi)\left((2+\lambda)(1+\psi)^2 d + O(\log^3(n)/\varepsilon)\right) + O(\log^3(n)/\varepsilon)\\
&= (2+\lambda)(1+\psi)^3 d + O(\log^3(n)/\varepsilon).
\end{aligned}$$

Finally, the last expression gives a $(2+\eta)d + O(\log^3(n)/\varepsilon)$ approximation for any constant $\eta > 0$ for appropriately chosen λ and ψ. This gives a $(2+\eta, O(\log^3 n/\varepsilon))$-approximation for the orientation produced by the low out-degree ordering. ∎

Densest Subgraph A straightforward extension of Algorithm 1 where nodes move up levels in all groups simultaneously also leads to a ε-LEDP approximate densest subgraph algorithm which is derived from the non-private algorithm of [BHNT15] that finds an approximate densest subgraph by peeling the layers of the level data structure one by one for each group g and taking the subgraph with largest density. The privacy for the densest subgraph algorithm holds by Theorem 4.1 and because privacy is maintained after post-processing (Theorem 2.17). Our algorithm is given in Algorithm 5 and the proof of our approximation bound follows from the proofs of Theorem 2.6 and Corollary 2.7 of Bhattacharya et al. [BHNT15] with minor modifications.

First, we describe a few key points of Algorithm 5. The algorithm operates over $O(\log n)$ rounds (Line 5) where in each round, each node (Line 6) computes its noisy degrees for each of the $O(\log n)$ groups (Line 7). For each group, each node i computes a noisy degree (if it is in the current level r) and releases this noisy degree (Line 13); this is in contrast to Algorithm 1, where i releases either 1 or 0. The noisy degree in this setting is used to compute the approximate density. Finally, Line 17 is performed by the curator who has all of the released degrees of every node $i \in [n]$ in each of the $O(\log n)$ levels and $O(\log n)$ groups. Thus, the curator can successively peel levels from the smallest to largest level while computing the sum of the noisy degrees of all nodes that remain after the most recently peeled level. Using this sum of noisy degrees as well as the public set of levels of each node, the curator can produce a subset of nodes whose induced subgraph is an approximate densest subgraph as follows.

The curator finds the group g with the largest index that has a non-empty last level (using $L^g_{2\log n-1}$ for every $g \in \{0,\dots,2\log n-1\}$). Taking the noisy degrees of all nodes released for group g and using $L^g_{2\log n-1}$, the curator peels the levels one by one from the smallest level to the largest level while maintaining the sum of the degrees divided by the number of nodes that remain after each round of peeling. The curator uses the released noisy degrees corresponding with $L^g_{2\log n-1}[i]$ for each $i \in [n]$ and $g \in \{0,\dots,2\log n-1\}$. The curator keeps and returns as S the subset of nodes whose ratio of the sum of the noisy degrees over the number of nodes in the subset is the largest across all iterations of peeling. Line 18 returns the noisy density of the set S (scaled by an appropriate factor) using the sum of the released noisy degrees.

The privacy of this algorithm follows from Theorem 4.1 and the fact that privacy is maintained after post-processing (Theorem 2.17). We now prove the approximation guarantees of this algorithm.

Lemma 4.9. *There exists an ε-edge LEDP algorithm on input $G=(V,E)$ that gives a set of nodes whose induced subgraph is a $(4+\eta, O(\log^3 n/\varepsilon))$-approximate densest subgraph in $O(\log n)$ rounds.*

Proof. First, we show that if $(1+\psi)^g > 2(1+\psi)D^* + \frac{c\log^3 n}{\varepsilon}$, then for all $i \in [n]$, it holds that $L^g_{2\log n-1}[i] < 2\log n-1$ whp for appropriately large constant $c \geq 1$; in other words, the last level of group g is empty whp. As in the proof of Theorem 2.6 in [BHNT15], we can show that for any level $r \in \{0,\dots,2\log n-1\}$ in group g it holds that (let $\delta(G[Z_r])$ be the sum of the degrees of every node in the induced subgraph of Z_r divided by $|Z_r|$):

$$\delta(G[Z_r]) = 2\cdot\rho(G[Z_r]) \leq 2\cdot\max_{S\subseteq V}\rho(G[S]) = 2\cdot D^* < \frac{(1+\psi)^g - \frac{c\log^3 n}{\varepsilon}}{1+\psi}. \tag{7}$$

From this, we know that the number of nodes in Z_r with induced degree in Z_r at least $(1+\psi)^g - \frac{c\log^3 n}{\varepsilon}$ is at most $\frac{|Z_r|}{1+\psi}$ since otherwise, it violates Eq. (7). Let C_r be the set of nodes in Z_r with degree less than $(1+\psi)^g - \frac{c\log^3 n}{\varepsilon}$. Then, it follows that $|Z_r \setminus C_r| \leq \frac{|Z_r|}{1+\psi}$. By Line 13 and Lemma 2.12, we have $Z_{r+1}\cap C_r = \emptyset$ since the noise does not exceed $\frac{c\log^3 n}{\varepsilon}$ whp. Thus, $|Z_{r+1}| \leq |Z_r \setminus C_r| \leq \frac{|Z_r|}{1+\psi}$. Then, for all $r \in \{0,\dots,2\log n-2\}$, we have that $|Z_{r+1}| \leq \frac{|Z_r|}{1+\psi}$. Multiplying these inequalities gives us $|Z_{2\log n-1}| \leq \frac{|Z_1|}{(1+\psi)^{2\log n-1}}$. Since $|Z_1| \leq n$, we get $|Z_{2\log n-1}| \leq \frac{n}{(1+\psi)^{2\log n-1}} = \frac{(1+\psi)n}{n^2} < 1$ which means that $Z_{2\log n-1} = \emptyset$ whp.

Now, suppose that $(1+\psi)^g < \frac{D^*}{1+\psi} - \frac{c\log^3 n}{\varepsilon}$, and let $S^* \subseteq V$ be a subset of nodes with highest density, i.e. $\rho(G[S^*]) = D^*$. We will show that $S^* \in Z_r$ for all $r \in \{0,\dots,2\log n-1\}$. This means that $Z_{2\log n-1} \neq \emptyset$. We prove this via induction. Clearly, in the base case, since $S^* \subseteq V$, then $S^* \subseteq Z_0$. For the induction hypothesis, we assume that $S^* \subseteq Z_r$. Then, we prove that $S^* \subseteq Z_{r+1}$. By Lemma 2.4 of [BHNT15], for every node $i \in S^*$, we have that $\deg_{Z_r}(i) \geq \deg_{S^*}(i) \geq \rho(G[S^*]) = D^* > (1+\psi)\cdot\left((1+\psi)^g + \frac{c\log^3 n}{\varepsilon}\right)$.

Algorithm 5: ε-LEDP Densest Subgraph

Input: Adjacency lists $(\mathbf{a}_1, \ldots, \mathbf{a}_n)$, constants $\psi \in (0,1)$, and privacy parameter $\varepsilon \in (0,1)$.

Output: A private set of nodes whose induced subgraph is a $\left(4+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate densest subgraph in G.

Function `LEDPDensestSubgraph`$((\mathbf{a}_1, \ldots, \mathbf{a}_n), \varepsilon, \psi)$

  Curator initializes $L_0^0, L_1^0, \ldots, L_{2\log n-1}^0, \ldots, L_0^{2\log n-1}, \ldots, L_{2\log n-1}^{2\log n-1}$ with $L_r^g[i] \leftarrow 0$ for every $i \in [n]; r, g \in [0, \ldots, 2\log n - 1]$.

  for $r = 0$ *to* $2\log n - 2$ **do**

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

      for $g = 0$ *to* $2\log n - 1$ **do**

        $L_{r+1}^g[i] \leftarrow L_r^g[i]$.

        if $L_r^g[i] = r$ **then**

          Let $\mathcal{U}_{i,g}$ be the number of neighbors $j \in \mathbf{a}_i$ where $L_r^g[j] = r$.

          Sample $X \leftarrow \mathsf{Geom}(\varepsilon/(8\log^2 n))$.

          Compute $\widehat{\mathcal{U}}_{i,g} \leftarrow \mathcal{U}_{i,g} + X$.

          i **releases** $\widehat{\mathcal{U}}_{i,g}$.

          if $\widehat{\mathcal{U}}_{i,g} > (1+\psi)^g$ **then**

            $L_{r+1}^g[i] \leftarrow L_r^g[i] + 1$. ▷ Curator moves i up one level in group g.

    Curator publishes L_{r+1}^g for every $g \in [0, \ldots, 2\log n - 1]$.

  Curator uses released noisy degrees of all nodes and $L_{2\log n-1}^g$ for all $g \in [0, \ldots, 2\log n - 1]$ to peel the levels one by one and determine and return the set of nodes S whose induced subgraph is an approximate densest subgraph.

  Let $\hat{W}$ be the sum of the noisy degrees of S. **return** $(S, \frac{\hat{W}}{2|S|} - \frac{c\log^3 n}{\varepsilon})$ for sufficiently large $c \geq 1$.

By Line 13 and Lemma 2.12, the minimum value of the noise variable added to the degree of i is $-\frac{c\log^3 n}{\varepsilon}$ whp and $(1+\psi)^{g+1}$ exceeds the cutoff of $(1+\psi)^g$; so i is in level $r+1$. Thus, for all $i \in S^*$, node $i \in Z_{r+1}$. This shows that the topmost level of group g contains S^*.

Now, we show that if $(1+\psi)^g < \frac{D^*}{1+\psi} - \frac{c\log^3 n}{\varepsilon}$, then there is a level $r \in \{0, \ldots, 2\log n - 1\}$ where $\rho(G[Z_r]) \geq \frac{(1+\psi)^g}{2(1+\psi)} - \frac{c\log^3 n}{\varepsilon}$. For the sake of contradiction, suppose this is not the case. Then, we have $\delta(G[Z_r]) = 2 \cdot \rho(G[Z_r]) < \frac{(1+\psi)^g}{1+\psi} - \frac{2c\log^3 n}{\varepsilon}$ for every $r \in \{0, \ldots, 2\log n - 1\}$. Then, applying a similar argument as for the first case (since this equation follows Eq. (7) except for the constant on the additive noise), we conclude that $|Z_{r+1}| \leq \frac{|Z_r|}{1+\psi}$ for every $r \in \{0, \ldots, 2\log n - 1\}$ which implies that $Z_{2\log n-1} = \emptyset$. We have arrived at a contradiction.

The rest of the proof follows from a modification of the proof of Corollary 2.7 of [BHNT15]. We consider all groups $g \in \{0, \ldots, 2\log n - 1\}$. We find the group with the largest index that has a non-empty last level and perform the peeling procedure. Suppose g is the group. By what we showed above, we know that $\frac{D^*}{1+\psi} - \frac{c\log^3 n}{\varepsilon} \leq (1+\psi)^g \leq 2(1+\psi)D^* + \frac{c\log^3 n}{\varepsilon}$. By the above, if the last level is not empty in group g, then, there exists an induced subgraph Z_r where $r \in \{0, \ldots, 2\log n - 1\}$, obtained from peeling the levels, that has density at least $\frac{(1+\psi)^g}{2(1+\psi)} - \frac{c\log^3 n}{\varepsilon}$. Finally, the returned noisy degrees for each node is at least its actual degree minus $\frac{c\log^3 n}{\varepsilon}$ whp. Let $r \in \{0, \ldots, 2\log n - 1\}$ be the level where Z_r has the highest density in g. Let $\hat{W}_r$ be the sum of the noisy degrees of nodes in Z_r. Then, we can bound $\hat{W}_r$ by the following expression in terms of $\delta(G[Z_r])$ and $\rho(G[Z_r])$,

$$\rho(G[Z_r]) - \frac{c\log^3 n}{\varepsilon} = \frac{\delta(G[Z_r])}{2} - \frac{c\log^3 n}{\varepsilon} \leq \hat{W}_r \leq \delta(G[Z_r]) + \frac{c\log^3 n}{\varepsilon} = 2 \cdot \rho(G[Z_r]) + \frac{c\log^3 n}{\varepsilon}.$$

By the above expression, we are guaranteed to select an r' whp where

$$\frac{\rho(G[Z_r])}{2} - \frac{c\log^3 n}{\varepsilon} = \frac{\delta(G[Z_r])}{4} - \frac{c\log^3 n}{\varepsilon} \leq \frac{\hat{W}_{r'}}{2} \leq \frac{\delta(G[Z_{r'}])}{2} + \frac{2c\log^3 n}{\varepsilon} = \rho(G[Z_{r'}]) + \frac{2c\log^3 n}{\varepsilon} \quad (8)$$

Combining all of the above observations, we get that the returned set of nodes in the approximate densest subgraph has density at least

$$\begin{aligned}\rho(G[Z_{r'}]) &\geq \frac{\rho(G[Z_r])}{2} - \frac{3c\log^3 n}{\varepsilon} \\ &\geq \frac{1}{2}\cdot\left(\frac{(1+\psi)^g}{2(1+\psi)} - \frac{c\log^3 n}{\varepsilon}\right) - \frac{3c\log^3 n}{\varepsilon} \\ &\geq \frac{\frac{D^*}{1+\psi} - \frac{c\log^3 n}{\varepsilon}}{4(1+\psi)} - \frac{7c\log^3 n}{2\varepsilon} = \frac{D^*}{4(1+\psi)^2} - O\left(\frac{\log^3 n}{\varepsilon}\right).\end{aligned}$$

□

4.7 Complexity Measures

In this section, we analyze the *round and node communication complexity* of our ε-edge LEDP algorithm; the latter is measured in terms of the size of messages sent by the nodes. Each node in Algorithm 1 releases $O(1)$ bits of information in each round. Then, we show a simple modification to the algorithm that allows us to obtain $O(\log n)$ rounds but with an additional node communication complexity of $O(\log n)$ bits.

Theorem 4.10. *Algorithm 1 has round complexity $O(\log^2 n)$. Released messages contain $O(1)$ bits, with high probability.*

Proof. Our algorithm requires $O(\log^2 n)$ rounds since we move vertices up at most $O(\log^2 n)$ levels. During each round, to obtain the communication complexity of $O(1)$ bits, each vertex outputs $O(1)$ bits indicating whether it is moving up a level or staying in the same level. □

Corollary 4.11. *There exists a $\left(2+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate LEDP algorithm for k-core decomposition that takes $O(\log n)$ rounds and outputs messages with $O(\log n)$ bits, with high probability. There exists a $\left(4+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate LEDP algorithm for densest subgraph that takes $O(\log n)$ rounds and outputs messages with $O(\log^2 n)$ bits, with high probability.*

Proof. Algorithm 3 modifies Algorithm 1 in the following way. Instead of computing the levels sequentially, we compute multiple groups in parallel. Specifically, each nodes outputs in each round, a message of $2\log_{(1+\varepsilon)} n - 1$ length where the i-th bit is 1 if node i moves up a level in group i. Thus, since each group has $O(\log n)$ levels, we can compute the movements of all vertices in all groups simultaneously in $O(\log n)$ rounds.

In Algorithm 5, each node releases $O(\log n)$ bits for each level in each group for their noisy degree. Thus, given $O(\log n)$ groups, the total node communication complexity is $O(\log^2 n)$ per level $r \in \{0, \ldots, 2\log n - 1\}$. There are $O(\log n)$ rounds since Line 6 uses $O(\log n)$ rounds. (Everything inside the for loop can be done simultaneously.) □

5 ε-Edge DP Densest Subgraph

In this section, we present an ε-edge DP densest subgraph algorithm that returns a subset of vertices $V' \subseteq V$ that induces a $\left(1+\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$-approximate densest subgraph for any constant $\eta > 0$ in $O\left((n+m)\log^3 n\right)$ time. Specifically, we prove the following theorem.

Theorem 5.1. *There exists an ε-edge DP densest subgraph algorithm that runs in $O\left((n+m)\log^3 n\right)$ time and returns a subset of vertices $V' \subseteq V$ that induces a $\left(1+\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$-approximate densest subgraph for any constant $\eta > 0$.*

We build upon the multiplicative weight update algorithm from [BGM14] along with modifications made by [SV20] in their distributed algorithm. Although a number of private algorithms [BLR13, GLM+10, GRU12, GZ21, HLM12, HR10, SU17, UV11, Vad17] exist for the multiplicative weight updates (MWU)

(PRIMAL) Maximum Density Subgraph

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{v \in V} y_v = 1 \\ & x_e \leq y_u, x_e \leq y_v \qquad \forall e = \{u, v\} \in E \\ & y_v, x_e \geq 0 \qquad \forall e \in E, v \in V \end{aligned}$$

(DUAL) Lowest Out-Degree Orientation

$$\begin{aligned} \text{minimize} \quad & B \\ \text{subject to} \quad & \alpha_{eu} + \alpha_{ev} \geq 1 \qquad \forall e = \{u, v\} \in E \\ & \sum_{e \ni u} \alpha_{eu} \leq B \qquad \forall u \in V \text{ where } e \in E \\ & \alpha_{eu}, \alpha_{ev} \geq 0 \qquad \forall e = \{u, v\} \in E \end{aligned}$$

Figure 1: Fig. 1 of [SV20]: Linear programs for densest subgraph (primal) and fractional lowest out-degree orientation (dual).

method of Arora et al. [AHK12], such private algorithms focus on private multiplicative weights for linear and non-linear queries into databases and data release. Such techniques in the database query setting do not immediately transfer to our densest subgraph setting; namely, these algorithms often use the exponential mechanism to select queries, which is unnecessary in our setting, since all nodes are queried during each update step and every node provides a value to update the state of the algorithm. Conversely, our ε-edge DP densest subgraph algorithm also does not have implications for private multiplicative weight update algorithms for database queries.

In this section, to be consistent with the previous non-private works [BGM14, CQT22, SV20], we give our multiplicative approximation factors as $(1 - a \cdot \eta')$ for fixed constant a and $\eta' \in (0, 1/a)$. Specifically, they define a $(1 - \eta')$-approximate densest subgraph to be one with density at least $(1 - \eta') \cdot D^*$, where D^* is the density of the densest subgraph. In Section 2, we define our multiplicative approximation factor as $(1 + \eta)$ to be consistent with the other private densest subgraph works [FHS22, NV21]. It is easy to convert between the two approximation factors since a $(1 - a \cdot \eta')$ guarantee for any constant $\eta' \in (0, 1/a)$ implies a $(1 + \eta)$ multiplicative guarantee for any $\eta > 0$ since $(1 - a \cdot \eta') \leq \frac{1}{1+\eta}$ when $0 < \eta \leq \frac{a\eta'}{1-a\eta'}$, and we can choose a sufficiently small constant η. For simplicity, from now on we fix a constant $\eta \in (0, 1/12)$ to be the input parameter for our algorithms.

We present our algorithm in two parts. The main part, given in Algorithm 6, calls the subroutine given in Algorithm 7 on various values of z. Algorithm 6 iterates through powers of $(1 + \eta)^i$ for every $i \in [\lfloor \log_{(1+\eta)} n \rfloor]$. For each $(1+\eta)^i$, the algorithm passes the value as the input parameter z into Algorithm 7. Algorithm 7 goes through $O\left(\frac{\log n}{\eta^3}\right)$ phases where loads are added to the edges in each phase. Then, the algorithm returns a set of nodes whose induced subgraph has density at least $(1-12\eta)z - \frac{c \log^4 n}{\varepsilon}$ for sufficiently large constant $c \geq 1$ whp. Algorithm 6 then returns the set of nodes returned for the largest power of $(1+\eta)^i$ or all of the nodes in the input graph if Algorithm 7 did not return a subset of the nodes for any of the inputs for the parameter z.

The crux of our approximate densest subgraph algorithm lies in Algorithm 7. In order to ensure ε-edge DP, our algorithm creates ***dummy*** edges that take a portion of the load. This technique also naturally leads to a general privacy framework for a certain class of *locally-adjustable algorithms*, which we present in Section 6. Such dummy edges are responsible for both accumulating load and for determining whether the stopping conditions are satisfied. We describe our algorithm in more detail and prove its privacy, approximation, and runtime guarantees in the following sections.

The densest subgraph algorithm performs multiplicative weight update on the dual of the densest subgraph LP [Cha00] (shown in Fig. 1). Intuitively, this algorithm works by distributing a given load z on a node (corresponding to the loads given by edges to the nodes in the dual LP) to its adjacent edges. Nodes with a large number of adjacent edges will be able to spread out its load among its many adjacent edges. Edges with small cumulative load will be adjacent to two high-degree nodes. Hence, they should be included in the densest subgraph. We can find such a subgraph by iterating from small to large load and keeping nodes that are adjacent to many edges with small loads. Bahmani et al. [BGM14] were the first to apply the MWU framework to densest subgraphs and Su and Vu [SV20] give an explicit analysis for this algorithm.

Algorithm 6: ε-Edge DP Densest Subgraph

1 **Input:** Graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, a constant $\eta \in (0, 1/12)$, and privacy parameter $\epsilon \in (0, 1)$.
2 **Output:** A subset of nodes whose induced subgraph is a $\left(1 - 12\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$-approximate densest subgraph.
3 **Function** `EdgeDPDensestSubgraph`$(G = (V, E), \eta, \varepsilon)$
4 $V_{\max} \leftarrow V$.
5 **for** $i \in [\lfloor \log_{(1+\eta)} n \rfloor]$ **do**
6 Set $z = (1 + \eta)^i$.
7 $(S, x) \leftarrow$ `EdgeDPDensestSubgraphZ`$(G = (V, E), z, \eta, \epsilon)$ (Algorithm 7).
8 **if** $x \neq 0$ **then**
9 $V_{\max} \leftarrow S$.
10 **return** $V_{\max}$.

Although the non-private algorithms of [BGM14, SV20] adapt the MWU framework, the analyses of [SV20] are self-contained. Thus, we present our private algorithm in its entirety without the need to define the MWU framework. We modify the analysis of [SV20] to show the approximation factor of our ε-DP algorithm.

5.1 Detailed Algorithm

The pseudocode for our algorithm is given in Algorithm 6. Recall from Definition 2.11 that $\mathsf{Geom}(b)$ denotes the symmetric geometric distribution. Specifically, $\mathsf{Geom}(b)$ gives an integer output $X \sim \mathsf{Geom}(b)$ with probability $\frac{e^b-1}{e^b+1} \cdot e^{-|X| \cdot b}$. In Algorithm 6, we iterate through $i \in [\lfloor \log_{(1+\eta)}(n) \rfloor]$ (Line 5) to obtain $z = (1+\eta)^i$ (Line 6). After obtaining z, we pass the value into Algorithm 7 as the parameter z (Line 7). Algorithm 7 returns a subset of nodes with density at least $z - \frac{c \log^4 n}{\varepsilon}$ (for sufficiently large constant $c \geq 1$) whp or all the nodes in the graph. We then check whether a set of nodes for the current parameter z is returned (Line 8); if so, then the subgraph $S \subseteq V$ obtained for the largest z parameter that is passed into the algorithm (Lines 4 and 9) is returned. If no set of nodes is returned for any z, then all of the nodes in the graph are returned. We now explain the crux of our algorithm: Algorithm 6 calls Algorithm 7 in Line 7 with input parameter z and Algorithm 7 returns a subset of nodes with density bounded by z minus additive $\text{poly}(\log n)$ error.

Algorithm 7 operates over $T \leftarrow \frac{c_0 \log n}{\eta^3}$ phases for some sufficiently large constant $c_0 > 0$ (Line 4). Each phase updates the *load* of each edge e, where the initial loads $\ell(e)$ of all edges are set to 0 (Line 5). Then, for each of the T phases, in parallel, each node v sorts its adjacent edges in non-decreasing order according to their load (Line 10). The loads of the edges essentially correspond to variable values in the well-known dual of the densest subgraph LP (shown in Fig. 1). Then, we sample from the symmetric geometric distribution to create the dummy edges (Line 11). Let X_v be the number of these dummy edges created for a node v. There are two ways that these dummy edges affect the loads distributed on the real edges. First, if X_v is negative, then we create dummy edges that we assume have the *smallest* loads among all adjacent edges to v. In this case, these dummy edges then take up part of the load given by v and hence, the real edges adjacent to v altogether receive less total load than z. However, if X_v is positive, then we imagine we create more open slots for additional real edges to take on load from v. In this case, the sum of the loads received by all real edges adjacent to v may be greater than z. This intuition translates to the procedure for assigning loads to edges in Line 13: each of the $[\lceil z/2 \rceil - 1 + X_v]_0^{\deg(v)}$ edges with the smallest loads is assigned a load of 2. After assigning new loads to the edges, the algorithm attempts to find an approximate densest subgraph with sufficient density.

We iterate through each possible integer load in $[0, 4T]$ (Line 14), which are the minimum and maximum possible load on any edge due to the fact that each endpoint of an edge adds at most a load of 2 during each of the T phases. For each possible integer load in $[0, 4T]$, we check whether there exists a densest subgraph of sufficient density, specifically, density at least $z + Y - \frac{c_2 \log^4 n}{\varepsilon}$ (Line 21). If so, then the algorithm returns a set of nodes V'_ℓ, where the density of the induced subgraph consisting of the nodes in V'_ℓ approximates the density of the densest subgraph in G (Line 22). To check for whether a sufficiently dense subgraph exists, first, each node $v \in V$ samples a noise Z from the symmetric geometric distribution (Line 17). Then, node v checks the number of incident edges e to v that have load $\ell(e) \leq \ell$ that is at most the load of the current

Algorithm 7: Edge DP Densest Subgraph using Density Parameter z

Input: Graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, density $z \geq 0$, constant $\eta \in (0, 1/12)$, privacy parameter $\epsilon \in (0, 1)$, and sufficiently large constants $c_0, c_1, c_2, c > 0$.

Output: A pair (V', z') where V' is a set of nodes $V' \subseteq V$ where $G[V']$ has density at least $z' - O\left(\frac{\log^4 n}{\varepsilon}\right)$.

Function `EdgeDPDensestSubgraphZ`$(G = (V, E), z, \eta, \varepsilon)$

Let $T \leftarrow \frac{c_0 \log n}{\eta^3}$.

Initialize load $\ell(e) \leftarrow 0$ for all $e \in E$.

if $z = 0$ **then**

  return $(V, 0)$.

for *phase* $t = 1$ *to* T **do**

  for *each node* $v \in V$ **do**

    Let $[e_1, e_2, \ldots, e_{\deg(v)}]$ be an ordered list of edges adjacent to v sorted in non-decreasing order by $\ell(e_1) \leq \ell(e_2) \leq \cdots \leq \ell(e_{\deg(v)})$ (breaking ties by the ID of the other endpoint).

    Sample $X_v \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T \log_{(1+\eta)} n}\right)$.

    Initialize each $\hat{\alpha}^t_{e_i v} \leftarrow 0$.

    Set $\hat{\alpha}^t_{e_i v} \leftarrow 2$ for $i = 1, \ldots, [\lceil z/2 \rceil - 1 + X_v]_0^{\deg(v)}$.

  for *each integer* $\ell \in [0, 4T]$ **do**

    Set $V'_\ell \leftarrow \emptyset$.

    for *each node* $v \in V$ **do**

      Sample $Z \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T(4T+1) \log_{(1+\eta)} n}\right)$.

      if *number of incident edges* e *to* v *with* $\ell(e) \leq \ell$ *is at least* $\lceil z/2 \rceil + Z - \frac{c_1 \log^4 n}{\varepsilon}$ **then**

        $V'_\ell \leftarrow V'_\ell \cup \{v\}$.

    Sample $Y \sim \mathsf{Geom}\left(\frac{\varepsilon}{3T(4T+1) \log_{(1+\eta)} n}\right)$.

    if $G[V'_\ell]$ *is a non-empty graph with density* $\geq z + Y - \frac{c_2 \log^4 n}{\varepsilon}$ **then**

      return (V'_ℓ, z)

  for *each edge* $e = (u, v) \in E$ **do**

    Set $\ell(e) \leftarrow \ell(e) + \hat{\alpha}^t_{eu} + \hat{\alpha}^t_{ev}$.

return $(V, 0)$.

iteration, ℓ. If the number of such incident edges is at least $\lceil z/2 \rceil + Z - \frac{c_1 \log^4 n}{\varepsilon}$ (Line 18), we add the node to the set V'_ℓ. We then check the density of the subgraph induced by all nodes that pass the check (Lines 19 and 21). If the density of this subgraph is sufficiently large, we return the nodes in V'_ℓ as the approximate densest subgraph (Line 22). Otherwise, if the density is not sufficiently large, we update the loads of each of the edges and proceed with the next iteration of the algorithm (Line 24). We show in Section 5.3 that we return a subgraph of sufficiently large density whp if z is sufficiently small (but not too small) compared to the optimum density D^*.

We first prove that Algorithm 6 is ε-edge DP in Section 5.2. As the main intermediate step, we prove that Algorithm 7 is $\left(\frac{\varepsilon}{\log_{(1+\eta)} n}\right)$-edge DP in Lemma 5.2. Then, we prove that we return a set of nodes where the induced subgraph on this set gives our desired approximation factor in Section 5.3. Finally, we analyze the runtime of our algorithm in Section 5.4.

5.2 ε-Edge DP Privacy Guarantees

We prove Algorithm 6 is ε-edge DP in this section. We first prove our main lemma that Algorithm 7 is $\frac{\varepsilon}{\log_{(1+\eta)} n}$-edge DP in Lemma 5.2. Then, we prove Lemma 5.6 via composition (Theorem 2.15) over $\log_{(1+\eta)} n$ calls to the algorithm. The key challenge in our analysis is that although we only use symmetric geometric noise in our algorithm, we sometimes cannot use the proof of the privacy of the geometric mechanism directly. and additional analysis is necessary. Nevertheless, we show that our analysis is quite intuitive and can also

be applied to our framework in Section 6. In the following proofs, we denote edge-neighboring graphs by $G = ([n], E)$ and $G' = ([n], E \cup \{e'\})$. We also use the terminology **ε-indistinguishable distributions** to denote two distributions where the probabilities of any event differs by at most a factor of e^{ε} between the distributions. It follows by definition that two symmetric geometric distributions with the same parameter and whose means differ by at most 1 are ε-indistinguishable. Furthermore, suppose that we have the pairs of distributions (A_1, B_1) and (A_2, B_2) where each pair is ε-indistinguishable. If all four distributions are independent, then, the joint distributions for (A_1, B_1) and (A_2, B_2) are 2ε-indistinguishable.

Lemma 5.2. *Algorithm 7 is $\frac{\varepsilon}{\log_{(1+\eta)} n}$-edge DP.*

Proof. Algorithm 7 iterates through T phases (Line 8) where in each phase, it first assigns loads to edges (Lines 10 to 13). The assigned loads are added to a cumulative load at the end of the procedure (Line 24). After assigning loads to edges, it iterates through $4T$ integer loads (Line 14). For each load, the algorithm iterates through each node (Line 16) and determines whether to add the node to V'_ℓ (Lines 17 to 19). After constructing V'_ℓ, it determines whether to return V'_ℓ if the subgraph induced on V'_ℓ has sufficiently large density (Lines 21 and 22). Finally, if no node is returned at the end of all phases and all iterations, all nodes in the graph are returned (Line 25).

We introduce notation for each of the following random variables: the loads assigned to each edge during each phase, indicator variables for whether each node is added to V'_ℓ during each of the $4T$ iterations of each phase, and indicator variables for whether V'_ℓ is returned. We prove the privacy of our algorithm by conditioning on the states of random variables from previous iterations and phases and showing that with approximately equal probabilities the random variables in the current iteration take the same values in edge-neighboring graphs.

We now define each of the random variables that we use in our proof. Given a graph $G = ([n], E)$, we first define the random variables representing the loads assigned to each edge during each phase. In other words, Let L^t be a set of triples $L^t = \{(e, \hat{\alpha}^t_{ei}, \hat{\alpha}^t_{ej}) \mid e \in E\}$ where each triple $(e, \hat{\alpha}^t_{ei}, \hat{\alpha}^t_{ej})$ contains an edge $e = (i, j) \in E$ and the loads $\hat{\alpha}^t_{ei}, \hat{\alpha}^t_{ej}$ assigned to edge $e = (i, j)$ in phase t. Then, let $L = \left(L^1, \ldots, L^T\right)$ be a sequence of load assignments to edges for each of the T phases. The interesting case is when $z > 0$ as otherwise all nodes of the graph are returned when $z = 0$. Let $F(L^t, E^*)$ be a function that, given L^t and a set of edges E^*, returns a set of loads J^t where $J^t = \{(e, \hat{\alpha}^t_{ei}, \hat{\alpha}^t_{ej}) \in L^t \mid e \in E^*\}$ is a set of tuples $(e, \hat{\alpha}^t_{ei}, \hat{\alpha}^t_{ej})$ from L^t for each edge $e \in E^*$.

Then, let $B^\ell_{v,t}(G)$ be the indicator random variable for node v, phase t, and load cutoff ℓ that is 1 if node i has at least $\lceil z/2 \rceil + Z^\ell_{v,t} - \frac{c_1 \log^4 n}{\varepsilon}$ neighbors in G with load at most ℓ (where $Z^\ell_{v,t} \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T(4T+1)\log_{(1+\eta)} n}\right)$) and 0 otherwise. Finally, let $C^\ell_t(G)$ be the indicator random variable for phase t and load cutoff ℓ in G that is 1 if the density of V'_ℓ satisfies the cutoff in Line 21 and 0 otherwise. If $C^\ell_t(G) = 1$ for some phase t and load ℓ, then we set the values of *all random variables* to 0 for all iterations $\ell' > \ell$ in phase t and for all phases $t' > t$.

We introduce several claims (Claims 5.3 to 5.5) and their proofs before proving this lemma. We first show the following claim about the loads assigned to edges in phase t conditioned on the values of all random variables for phases less than t. Let $\mathcal{J}$ be the set of all possible sets of triples of load assignments to edges in E and let $S^t \in \mathcal{J}$. Let $(b_1, \ldots, b_n)_{\ell,t} \in \{0,1\}^n$ and $d^\ell_t \in \{0,1\}$ for all $t \in [T]$ and $\ell \in \{0, \ldots, 4T\}$. To set up Claim 5.3, let R^t be the sequence of events where for all phases less than a fixed $t \in [T]$,

$$
\begin{aligned}
&R^t(G) = (C^{4T}_{t-1}(G) = d^{4T}_{t-1}, \ldots, C^0_1(G) = d^0_1,\\
&\left(B^{4T}_{1,t-1}(G), \ldots, B^{4T}_{n,t-1}(G)\right) = (b_1, \ldots, b_n)_{4T,t-1}, \ldots, \left(B^0_{1,1}(G), \ldots, B^0_{n,1}(G)\right) = (b_1, \ldots, b_n)_{0,1},\\
&J^{t-1}(G) = S^{t-1}, \ldots, J^1(G) = S^1).
\end{aligned}
$$

Define $\hat{R}^t$ the same way as R^t, substituting G' for G.

Note that since we are in the discrete setting, computing the probability expression in Claim 5.3 is sufficient for privacy as opposed to the common $\frac{1}{e^{\varepsilon/(4T)}} \leq \frac{\mathsf{Pr}\left[J^t(G') \in S \mid \hat{R}^t(G')\right]}{\mathsf{Pr}\left[J^t(G) \in S \mid R^t(G)\right]} \leq e^{\varepsilon/(4T)}$ for any $S \subseteq \mathcal{J}$, which is necessary for continuous settings.

Claim 5.3. *For all edge-neighboring graphs $G = ([n], E)$ and $G' = ([n], E \cup \{e'\})$ and all $S^t \in \mathcal{J}$,*

$$
\frac{1}{e^{\varepsilon/(3T\log_{(1+\eta)} n)}} \leq \frac{\mathsf{Pr}\left[J^t(G') = S^t \mid R^t(G')\right]}{\mathsf{Pr}\left[J^t(G) = S^t \mid R^t(G)\right]} \leq e^{\varepsilon/(3T\log_{(1+\eta)} n)}.
$$

Proof. First, note that only Lines 9 to 13 and 24 assign loads to edges in Algorithm 7. Let $J^t(G)$ be the random variable representing the set of loads returned by $F(L^t, E)$ for phase t. Let $S = (S^1, \ldots, S^{t-1})$ be a sequence of sets of load values on edges in E. We prove, for all $t \in [T]$, that

$$\frac{1}{e^{\varepsilon/(3T\log_{(1+\eta)} n)}} \leq \frac{\mathsf{Pr}\left[J^t(G') = S^t \mid R^t(G')\right]}{\mathsf{Pr}\left[J^t(G) = S^t \mid R^t(G)\right]} \leq e^{\varepsilon/(3T\log_{(1+\eta)} n)}. \tag{9}$$

First, conditioned on the previous sets of loads being the same for edges in $J^t(G)$ and $J^t(G')$ for phases in $[t-1]$, the load at the beginning of phase t on $e \in E$ is given by $\ell_t(e) = \ell'_t(e)$ (where $\ell_t(e)$ is the load on e in phase t in G and $\ell'_t(e)$ is the load on e in phase t in G') since each $\ell_t(e) = \sum_{r=1}^{t-1}\left(\hat{\alpha}^r_{ei} + \hat{\alpha}^r_{ej}\right)$ for all $e \in E$. Then, the distribution on the loads assigned to $\hat{\alpha}^t_{ei}$and $\hat{\alpha}^t_{ej}$ are the same in G and G' for edges $e = (i,j) \in E$ where *neither endpoint* is u or v. It remains to consider loads assigned to edges in E where exactly one of the endpoints is u or v.

Let $(e_1, \ldots, e_{|N(v)|})$ denote the order of the edges determined by loads in $\{L^1(G), \ldots, L^{t-1}(G)\}$ and let $(\hat{e}_1 \ldots, \hat{e}_{|N(v)|+1})$ denote the order of the edges $N(v) \cup \{e'\}$ determined by loads in $\{L^1(G'), \ldots, L^{t-1}(G')\}$ (Line 10). Let edge $e' = (u, v)$ be the k-th edge in $(\hat{e}_1 \ldots, \hat{e}_{|N(v)|+1})$. Since we conditioned on $J^{t-1}(G') = J^{t-1}(G) = S^{t-1}, \ldots, J^1(G') = J^1(G) = S^1$, it follows that $e_p = \hat{e}_p$ for all $p \in [k-1]$ and $e_p = \hat{e}_{p+1}$ for all $p \in \{k, \ldots, |N(v)|\}$. Since G' has an additional edge, $\deg(v) = |N(v)|$ and $\deg'(v) = |N(v)| + 1$ for all nodes v. For the purposes of this proof, we pad each of the two sequences $OrderSeq(G) = (e_1 \ldots, e_{|N(v)|})$ and $OrderSeq(G') = (\hat{e}_1 \ldots, \hat{e}_{|N(v)|+1})$ with extra *filler* edges on the right ends to produce $(e_1, \ldots, e_{|N(v)|}, f_1, f_2, f_3, \ldots, f_{\lceil z/2\rceil - 1 - |N(v)|+1})$ and $(\hat{e}_1 \ldots, \hat{e}_{|N(v)|+1}, f_1, f_2, f_3, \ldots, f_{\lceil z/2\rceil - 1 - |N(v)|-1})$. These filler edges are unique and given in the same order in both sequences. This means that the position $\lceil z/2\rceil - 1$ falls on an edge in both sequences. We now define the *ideal cutoff* for the sequence $OrderSeq(G)$ to be $\lceil z/2\rceil - 1$ and for the sequence $OrderSeq(G')$ to be the position of the $(\lceil z/2\rceil - 1)$-st edge in $OrderSeq(G)$. To state in another way, if we let $\hat{e}$ be the $(\lceil z/2\rceil - 1)$-st edge in $OrderSeq(G')$, we find the position of $\hat{e}$ in $OrderSeq(G)$ (if $\hat{e} = e'$, then we take the edge to its right). The position of this edge is either at position $\lceil z/2\rceil - 1$ in $OrderSeq(G)$ or at position $\lceil z/2\rceil - 2$ in $OrderSeq(G)$. Trivially, the ideal cutoff for the $(\lceil z/2\rceil - 1)$-st edge in $OrderSeq(G)$ is $\lceil z/2\rceil - 1$. We now define the following variables with respect to these ideal cutoffs which we denote by $IDEAL_v(G')$ and $IDEAL_v(G)$. Conditioned on $R^t(G)$ and $R^t(G')$, these ideal cutoffs are fixed.

Let $CUT^t_v(G) = IDEAL_v(G) + X_v$ be the symmetric geometric variable with mean $IDEAL_v(G)$ for G in phase t and $CUT^t_v(G') = IDEAL_v(G') + \hat{X}_v$ be the variable for G'. Then, the conditional distributions (conditioned on $R^t(G)$ and $R^t(G')$, respectively) for $CUT^t_v(G)$ and $CUT^t_v(G')$ are symmetric geometric distributions with parameter given in Line 11 and with means $IDEAL_v(G)$ and $IDEAL_v(G')$, respectively. These random variables are drawn from symmetric geometric distributions where the mean differs by at most 1. Hence, these distributions are $\frac{\varepsilon}{6T\log_{(1+\eta)} n}$-indistinguishable by the parameter given in Line 11. The same argument above holds for u.

Now, we consider the joint distributions for the variables $CUT^t_v(G)$ and $CUT^t_u(G)$ (and respectively, $CUT^t_v(G')$ and $CUT^t_u(G')$). Conditioned on $R^t(G)$ and $R^t(G')$, the orderings are fixed for u and v which means that $IDEAL_u(G)$ and $IDEAL_v(G)$ are also fixed; thus, the variables $CUT^t_u(G)$ and $CUT^t_v(G)$ are independently drawn, the events for each endpoint are independent and so the joint conditional probability distributions are $\frac{\varepsilon}{3T\log_{(1+\eta)} n}$-indistinguishable, proving our claim. ☐

Intuitively, Claim 5.3 shows that for the set of edges E in both G and G', our algorithm assigns the same loads to edges in E in G and G' with roughly equal probability, conditioned on the values of the random variables from previous iterations and phases.

We show the following claim regarding the values of $B^\ell_{v,t}(G)$. Let W be the sequences of events for iterations less than ℓ in phase t and all phases less than t where

$$\begin{aligned}
&W = (C^{\ell-1}_t(G) = d^{\ell-1}_t, \ldots, C^0_1(G) = d^0_1,\\
&\left(B^{\ell-1}_{1,t}(G), \ldots, B^{\ell-1}_{n,t}(G)\right) = (b_1, \ldots, b_n)_{\ell-1,t}, \ldots, \left(B^0_{1,1}(G), \ldots, B^0_{n,1}(G)\right) = (b_1, \ldots, b_n)_{0,1},\\
&J^t(G) = S^t, \ldots, J^1(G) = S^1).
\end{aligned}$$

Define W' the same way as W, substituting G' instead of G.

Claim 5.4. *Given two edge-neighboring graphs G and G', let $(B^\ell_{1,t}(G),\dots,B^\ell_{n,t}(G))$ be the sequence of random variables and $(b_1,\dots,b_n)_{\ell,t}\in\{0,1\}^n$ for all nodes $i\in[n]$. Then, for any phase $t\in[T]$ and load $\ell\in\{0,\dots,4T\}$,*

$$\frac{1}{e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)}n)}}\le\frac{\mathsf{Pr}[(B^\ell_{1,t}(G'),\dots,B^\ell_{n,t}(G'))=(b_1,\dots,b_n)_{\ell,t}\mid W']}{\mathsf{Pr}[(B^\ell_{1,t}(G),\dots,B^\ell_{n,t}(G))=(b_1,\dots,b_n)_{\ell,t}\mid W]}\le e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)}n)}.$$

Proof. Let $Z^\ell_{v,t}$ and $\hat{Z}^\ell_{v,t}$ be the noise drawn in phase t and iteration ℓ for i in Line 17 of Algorithm 7 for G and G', respectively. Conditioned on $J^t(G)=S^t$ and $J^t(G')=S^t$, edge $e'=(u,v)\in E'\setminus E$ has some arbitrary load from $L^t(G')$ and all other edges have the same load in phase t in G and G'. For all nodes $i\neq u,v\in[n]$, the conditional distributions of the random variables $B^\ell_{i,t}(G)$ and $B^\ell_{i,t}(G')$ are the same given W' and W. Now we consider the conditional distributions of $B^\ell_{v,t}(G)$ and $B^\ell_{v,t}(G')$. Let $load^\ell_{v,t}(G)$ and $load^\ell_{v,t}(G')$ be the number of adjacent edges to v in G and G', respectively, with load at most ℓ in phase t; conditioned on W and W', respectively, $load^\ell_{v,t}(G)$ and $load^\ell_{v,t}(G')$ are fixed and $load^\ell_{v,t}(G)\le load^\ell_{v,t}(G')\le load^\ell_{v,t}(G)+1$ since v is adjacent to one additional edge in G' which can have load at most ℓ. Then, $B^\ell_{v,t}(G)=1$ when $LOAD^\ell_{v,t}(G)=Z^\ell_{v,t}-load^\ell_{v,t}(G)+\lceil z/2\rceil-\frac{c_1\log^4 n}{\varepsilon}\le 0$ (and symmetrically for $B^\ell_{v,t}(G)=0$). We define the random variable $LOAD^\ell_{v,t}(G')$ the same way except substituting $\hat{Z}^\ell_{v,t}$ and $load^\ell_{v,t}(G')$. The random variables $LOAD^\ell_{v,t}(G)$ and $LOAD^\ell_{v,t}(G')$ are symmetric geometric variables drawn from distributions with the same parameter and where the mean differs by at most 1. Thus, the conditional distributions for these two random variables are $\left(\frac{\varepsilon}{6T(4T+1)\log_{(1+\eta)}n}\right)$-indistinguishable (using the parameter from Line 17). Finally, the conditional distributions for $LOAD^\ell_{v,t}(G)$ and $LOAD^\ell_{u,t}(G)$ are independent since $load^\ell_{v,t}(G)+\lceil z/2\rceil-\frac{c_1\log^4 n}{\varepsilon}$ are fixed and $Z^\ell_{v,t}$ and $\hat{Z}^\ell_{v,t}$ are drawn independently. (The same holds for the distributions of the corresponding random variables for G'.) Hence, the conditional joint distributions for the pairs in G and the pairs in G' are $\left(\frac{\varepsilon}{3(4T+1)T\log_{(1+\eta)}n}\right)$-indistinguishable. ☐

Finally, we return the set of nodes in V'_ℓ in Lines 21 and 25 if the density of the induced subgraph $G[V'_\ell]$ is at least $z+Y-\frac{c_2\log^4 n}{\varepsilon}$ and if no induced density satisfies the cutoff, respectively. Similarly to the previous claims, we can prove the following claim for $C^\ell_t(G)$ and $C^\ell_t(G')$ conditioned on the sequence of indicator random variables $(B^\ell_{1,t}(G),\dots,B^\ell_{n,t}(G))$ and $(B^\ell_{1,t}(G'),\dots,B^\ell_{n,t}(G'))$.

Let Q be the sequence of events where

$$\begin{aligned}&Q=(C^{\ell-1}_t(G)=d^{\ell-1}_t,\dots,C^0_1(G)=d^0_1,\\&\left(B^\ell_{1,t}(G),\dots,B^\ell_{n,t}(G)\right)=(b_1,\dots,b_n)_{\ell,t},\dots,\left(B^0_{1,1}(G),\dots,B^0_{n,1}(G)\right)=(b_1,\dots,b_n)_{0,1},\\&J^t(G)=S^t,\dots,J^1(G)=S^1).\end{aligned}$$

We define Q' the same way as Q, substituting G' instead of G.

Claim 5.5. *Given two edge-neighboring graphs G and G', let Q and Q' be the sequences of events as defined above. Let $C^\ell_t(G)$ and $C^\ell_t(G')$ be the indicator random variables for G and G', respectively, as defined above, and let $d^\ell_t\in\{0,1\}$. Then, for any phase $t\in[T]$ and integer load $\ell\in\{0,\dots,4T\}$,*

$$\frac{1}{e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)}n)}}\le\frac{\mathsf{Pr}[C^\ell_t(G')=d^\ell_t\mid Q']}{\mathsf{Pr}[C^\ell_t(G)=d^\ell_t\mid Q]}\le e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)}n)}.$$

Proof. This proof follows the proof of Claim 5.4 almost identically except for the use of different random variables. Given the conditioning on events Q and Q', the set of nodes in V'_ℓ are fixed and the same in phase t in both G and G'. Then, the sensitivity of a function that returns the density $\rho(G[V'])$ of any subset $V'\subseteq V$ of nodes is 1. It follows that $\rho(G[V'_\ell])\le\rho(G'[V'_\ell])\le\rho(G[V'_\ell])+1$. This means that we can again argue that the random variables $DENSE^\ell_t(G)=Y^\ell_t+z-\rho(G[V'_\ell])-\frac{c_2\log^4 n}{\varepsilon}$ and $DENSE^\ell_t(G')$ (defined the same way except substituting $\hat{Y}^\ell_t$ and $\rho(G'[V'_\ell])$) are symmetric geometric variables drawn from distributions with the same parameter and where the mean of the distributions differ by at most 1. Thus, these two distributions are $\left(\frac{\varepsilon}{3T(4T+1)\log_{(1+\eta)}n}\right)$-indistinguishable and the claim follows. ☐

We now apply the chain rule on Claims 5.3 to 5.5 to prove our lemma. During each phase $t \in [T]$, we first compute the new loads assigned to $\hat{\alpha}_{eu}$ for all $e \in E$ and $u \in e$. Then, we iterate through all loads $\ell \in \{0, \ldots, 4T\}$ to obtain the random variables $B^{\ell}_{v,t}(G)$ and C^{ℓ}_{t}. We apply the chain rule as follows:

$$
\begin{aligned}
A &= \frac{\mathsf{Pr}[C_T^{4T}(G') = d_T^{4T} \mid Q']}{\mathsf{Pr}[C_T^{4T}(G) = d_T^{4T} \mid Q]} \times \frac{\mathsf{Pr}[(B_{1,T}^{4T}(G'), \ldots, B_{n,T}^{4T}(G')) = (b_1, \ldots, b_n)_{4T,T} \mid W']}{\mathsf{Pr}[(B_{1,T}^{4T}(G), \ldots, B_{n,T}^{4T}(G)) = (b_1, \ldots, b_n)_{4T,T} \mid W]} \\
&\times \frac{\mathsf{Pr}\left[J^T(G') = S^T \mid R'\right]}{\mathsf{Pr}\left[J^T(G) = S^T \mid R\right]} \times \cdots \times \frac{\mathsf{Pr}[C_1^0(G') = d_1^0 \mid (B_{1,1}^0(G'), \ldots, B_{n,1}^0(G')) = (b_1, \ldots, b_n)_{0,1}, J^1(G') = S^1]}{\mathsf{Pr}[C_1^0(G) = d_1^0 \mid (B_{1,1}^0(G), \ldots, B_{n,1}^0(G)) = (b_1, \ldots, b_n)_{0,1}, J^1(G) = S^1]} \\
&\times \frac{\mathsf{Pr}[(B_{1,1}^0(G'), \ldots, B_{n,1}^0(G')) = (b_1, \ldots, b_n)_{0,1} \mid J^1(G') = S^1]}{\mathsf{Pr}[(B_{1,1}^0(G), \ldots, B_{n,1}^0(G)) = (b_1, \ldots, b_n)_{0,1} \mid J^1(G) = S^1]} \times \frac{\mathsf{Pr}\left[J^1(G') = S^1\right]}{\mathsf{Pr}\left[J^1(G) = S^1\right]} \\
&= \frac{\mathsf{Pr}[C_T^{4T}(G') = d_T^{4T} \cap (B_{1,T}^{4T}(G'), \ldots, B_{n,T}^{4T}(G')) = (b_1, \ldots, b_n)_{4T,T} \cap C_1^0(G') = d_1^0 \cap J^T(G') = S^T \cap \cdots]}{\mathsf{Pr}[C_T^{4T}(G) = d_T^{4T} \cap (B_{1,T}^{4T}(G), \ldots, B_{n,T}^{4T}(G)) = (b_1, \ldots, b_n)_{4T,T} \cap C_1^0(G) = d_1^0 \cap J^T(G) = S^T \cap \cdots]}
\end{aligned}
$$

By Claims 5.3 to 5.5, we can upper and lower bound A by $\left(\frac{1}{e^{\varepsilon/(3T\log_{(1+\eta)} n)}}\right)^T \cdot \left(\frac{1}{e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)} n)}}\right)^{T(4T+1)} \cdot \left(\frac{1}{e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)} n)}}\right)^{T(4T+1)} \leq \frac{1}{e^{\varepsilon/\log_{(1+\eta)} n}} \leq A \leq e^{\varepsilon/\log_{(1+\eta)} n} \leq \left(e^{\varepsilon/(3T\log_{(1+\eta)} n)}\right)^T \cdot \left(e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)} n)}\right)^{T(4T+1)} \cdot \left(e^{\varepsilon/(3T(4T+1)\log_{(1+\eta)} n)}\right)^{(4T+1)T}$, proving Lemma 5.2. ☐

Lemma 5.6. *Algorithm 6 is ε-edge DP.*

Proof. Algorithm 6 calls Algorithm 7 a total of $\log_{(1+\eta)} n$ times. By Lemma 5.2 and by the composition theorem (Theorem 2.15, Algorithm 7 is $\frac{\varepsilon}{\log_{(1+\eta)} n}$-edge DP and by applying composition $\log_{(1+\eta)} n$ times, Algorithm 7 is ε-edge DP. ☐

5.3 Approximation Analysis

The proof of our approximation guarantee relies on the LP formulation of the densest subgraph problem and its dual introduced by Charikar [Cha00]. We modify the proofs of Su and Vu [SV20] to prove the approximation factor of our algorithm in the ε-edge DP setting; the application of the multiplicative weights framework to densest subgraph was first given in [BGM14] and the analysis was made explicit for the densest subgraph problem by [SV20]. Due to the added noise, all of the proofs for the original algorithm must be modified to account for the additional error. The primal LP solves the densest subgraph problem while its dual is traditionally used for the (fractional) lowest out-degree orientation problem. The primal (left) and its dual (right) are shown in Fig. 1. We use the formulation of the LP and its dual that is given in [SV20] and is also present in [BGM14]. In our analysis, we let D^* be the density of the densest subgraph in the input graph.

Charikar [Cha00] showed, via a rounding algorithm, that the optimal value of the LP is exactly the density of the densest subgraph in the graph. We use the notation from [SV20] when describing solutions to the primal and dual. Specifically, we say that a solution to the primal satisfies $\textsc{Primal}(A)$ if it is a feasible solution whose objective value is at least A. A dual solution satisfies $\textsc{Dual}(B)$ if it is a feasible solution with objective value at most B. Let D^* be the density of the densest subgraph in the input graph. We reiterate that the primal is feasible if $A \leq D^*$ and the dual is feasible if $B \geq D^*$. There are two main aspects of the analysis. As in Su and Vu [SV20], for each value of z and appropriately large constant $c \geq 1$, we show that at least one of the following conditions is satisfied:

- Algorithm 7 returns a subgraph which satisfies $\textsc{Primal}\left((1-12\eta)z - \frac{c_2 \log^4 n}{\varepsilon}\right)$ with high probability.
- The loads on the edges given by Algorithm 7 form a solution to $\textsc{Dual}\left((1+2\eta)z + \frac{c\log^3 n}{\varepsilon\eta^3}\right)$ with high probability.

We use the above to show that if $z < \frac{D^* - \frac{c\log^3 n}{\varepsilon\eta^3}}{1+10\eta}$, then our algorithm returns a subgraph with density at least $(1-12\eta)z - \frac{c_2\log^4 n}{\varepsilon}$ whp for sufficiently large constant $c \geq 1$ (that is also dependent on the constant

parameter $\eta \in (0, 1/12)$). To show this, we show that if $z < \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta}$, then no feasible solution to $\textsc{Dual}((1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3})$ exists, whp. This means that our algorithm will obtain a $(1-12\eta)z - \frac{c_2 \log^4 n}{\varepsilon}$ approximate densest subgraph since in order for at least one of the two conditions above to be satisfied, the first condition must be satisfied.

We use the following notation in our proofs. We define $w_e^t = (1-\eta)^{\ell^t(e)}$ to be the *weight* of edge e in phase t where $\ell^t(e)$ is the load on edge e in the *beginning* of phase t. Specifically, $\ell^t(e)$ is the load on edge e in phase t *before* the update in Line 24. Note that $\ell(e)$ is updated with a value in $\{0, 2\}$ for all phases $1 \le t \le T$, unlike the original algorithm given by [SV20] which updates the load with a potentially non-integer value. We, in fact, only need to modify the proofs of Lemma 3.1 and Lemma 3.2 (and not the proof of Lemma 3.3) from Su and Vu [SV20] since the loads on our edges are guaranteed to be integers throughout all phases of Algorithm 7. Our additive error allows us to update the loads with only integer values. Thus, our modification to only update loads using integers actually allows us to simplify our accuracy proof in some aspects.

We use the bound on the symmetric geometric noise (Lemma 2.12) in our main lemmas. Now, we present the lemmas that we will use to prove our approximation factors. The structure of this section is as follows. First, in Lemma 5.7, we show that if $z < \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta}$, then there must exist a phase t where $\sum_{e \in E} (w_e^t (\alpha_{eu}^t + \alpha_{ev}^t)) < \sum_{e \in E} w_e^t$. Then, in Lemma 5.8, we show that if $\sum_{e \in E} (\hat{w}_e^t (\alpha_{eu}^t + \alpha_{ev}^t)) < \sum_{e \in E} \hat{w}_e^t$ for a given set of weights $\hat{w}_e^t$, then there exists a λ where taking all nodes which are incident to at least $\lceil z/2 \rceil + Z$ edges with weight at least λ gives a subgraph with density at least $z - \frac{c_2 \log^4 n}{\varepsilon}$, whp, for a large enough constant $c \ge 1$. We show in Lemma 5.9 that because the loads on all edges are integers, the set of weights w'_e obtained from Algorithm 7 gives a value for $\lambda = (1-\eta)^{\ell'}$ where ℓ' is an integer in $[0, 4T]$ that satisfies the conditions of Lemma 5.8; this directly leads to our final approximation factor given in Lemma 5.10. Unlike Lemma 3.2 in [SV20], we do not need to prove Lemma 5.8 in terms of a factor $F \in (1/2, 1)$ since our additive error allows us to assign *integer* loads to edges upfront. The reason that Lemma 3.2 in [SV20] requires the factor of $F \in (1/2, 1)$ is that they allow no additive error; thus, each node must assign all of its z load to its adjacent edges (if it has degree at least $\lceil z/2 \rceil$) and the load assigned to the $\lceil z/2 \rceil$-th edge may not be an integer. In our case, because we have an additional additive error of $O\left(\frac{\log^4 n}{\varepsilon}\right)$, each node can afford to assign $z \pm \frac{c \log^4 n}{\varepsilon}$ load for constant $c \ge 1$.

Lemma 5.7. *If $z < \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta}$ for a large enough constant $c \ge 1$ (depending on constant parameter $\eta > 0$), then whp there exists a phase $1 \le t \le T$ where $\sum_{e \in E} (w_e^t (\alpha_{eu}^t + \alpha_{ev}^t)) < \sum_{e \in E} w_e^t$.*

Proof. We modify the proof of [SV20, Lemma 3.1], which is a standard analysis of the multiplicative weights update method of [AHK12, You95]. As in [SV20, Lemma 3.1], we prove this theorem via the following statements. This proof makes use of a feasible solution to the dual (Fig. 1). We make a small modification to our algorithm to obtain a potential set of variable values α_{eu} for every $e \in E, u \in e$ for the dual. Namely, we obtain a value for each α_{eu} by computing $\alpha_{eu} = (1+10\eta) \cdot \frac{\sum_{t=1}^{T} \alpha_{eu}^t}{T}$. Note that we do not return the solution to the dual in any part of our algorithm but only make use of it in the proof of this lemma. First, as noted before, $\textsc{Dual}(B)$ is feasible if and only if $B \ge D^*$. This means that if $B = (1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3} < D^*$, then $\textsc{Dual}\left((1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3}\right)$ is not feasible. The quantity satisfies $(1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3} < D^*$ when $z < \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+2\eta}$. If $\textsc{Dual}\left((1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3}\right)$ is not feasible, then naturally our algorithm cannot return a feasible solution. The final statement that we need to show is then: if our algorithm does not return a feasible solution to $\textsc{Dual}\left((1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3}\right)$, then there must exist a phase $1 \le t \le T$ where $\sum_{e \in E} w_e^t (\alpha_{eu}^t + \alpha_{ev}^t) < \sum_{e \in E} w_e^t$. This set of statements directly proves our desired lemma statement. The first two statements are trivial to prove. The crux of this proof is then focused on proving the last statement. To prove the last statement, it suffices to prove the contrapositive of the statement. Specifically, we spend the remaining part of this proof proving the statement: if $\sum_{e \in E} w_e^t (\alpha_{eu}^t + \alpha_{ev}^t) \ge \sum_{e \in E} w_e^t$ for all phases $1 \le t \le T$, then our algorithm returns a feasible solution to $\textsc{Dual}\left((1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3}\right)$. Note that returning a feasible solution given parameter z necessarily implies that $(1+2\eta)z + \frac{c \log^3 n}{\varepsilon \eta^3} \ge D^*$ and $z \ge \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+2\eta}$.

In order for the returned solution to be a feasible solution to the dual, it must satisfy all constraints in the dual. We first prove that if $\sum_{e\in E} w_e^t\,(\alpha_{eu}^t + \alpha_{ev}^t) \geq \sum_{e\in E} w_e^t$ for all $1 \leq t \leq T$, then $\sum_{e\ni u} \alpha_{eu}$ is upper bounded by $(1+2\eta)z + \frac{c\log^3 n}{\varepsilon\eta^3}$. This satisfies one of the constraints in the dual. Let X_u^t be the noise chosen by node u in phase t. We must upper bound the sum of the loads so cases when $X_u^t < 0$ are upper bounded by the cases when $X_u^t \geq 0$. If $X_u^t \geq 0$, then we potentially give weights of 2 to X_u^t additional adjacent edges to u. This means that the following holds:

$$\sum_{e\ni u} \frac{\sum_{t=1}^T \alpha_{eu}^t}{T}\cdot(1+10\eta) = \frac{(1+10\eta)}{T}\cdot\sum_{t=1}^T\sum_{e\ni u}\alpha_{eu}^t \leq \frac{(1+10\eta)}{T}\cdot\sum_{t=1}^T\left(z + \frac{2d\log^3 n}{\varepsilon\eta^3}\right) \tag{10}$$

$$= (1+10\eta)\cdot\left(z + \frac{2d\log^3 n}{\varepsilon\eta^3}\right) \leq (1+10\eta)z + \frac{c\log^3 n}{\varepsilon\eta^3} \tag{11}$$

$$\sum_{e\ni u}\alpha_{eu} \leq (1+10\eta)z + \frac{c\log^3 n}{\varepsilon\eta^3} \tag{12}$$

for appropriately large constant values $c, d \geq 1$. The last expression in Eq. (10) holds because X_u^t is upper bounded by $O\left(\frac{\log^3 n}{\varepsilon}\right)$ by Lemma 2.12. The first part of Eq. (11) follows by canceling T. Then, the second expression follows by substituting a large enough constant $c \geq 1$ for $2(1+10\eta)d$. Finally, Eq. (12) follows from substituting $\alpha_{eu} = (1+10\eta)\cdot\sum_{t=1}^T \frac{\alpha_{eu}^t}{T}$.

To prove that the returned solution satisfies the other constraints, we use a similar potential function as used in the proof of Lemma 3.1 in [SV20].[5] The potential function is defined as follows

$$\Phi(t) = \sum_{e\in E}\frac{(1-\eta)^{\ell^t(e)}\cdot\left(1-\eta+8\eta^2\right)^{T-t}}{\left(1-\eta+8\eta^2\right)^{(1-\eta)\cdot T}}.$$

We show that $\Phi(T) = \sum_{e\in E}\frac{(1-\eta)^{\ell^T(e)}}{(1-\eta+8\eta^2)^{(1-\eta)\cdot T}} < 1$ which implies for every edge $e = \{u, v\}$,

$$(1-\eta)^{\ell^T(e)} < \left(1-\eta+8\eta^2\right)^{(1-\eta)\cdot T} \tag{13}$$

$$\ell^T(e)\log_{(1-\eta+8\eta^2)}(1-\eta) > (1-\eta)T \tag{14}$$

$$\sum_{t=1}^T\left(\alpha_{eu}^t + \alpha_{ev}^t\right) > \frac{(1-\eta)T}{\log_{(1-\eta+8\eta^2)}(1-\eta)} \tag{15}$$

$$\frac{\sum_{t=1}^T\left(\alpha_{eu}^t + \alpha_{ev}^t\right)}{T} > \frac{(1-\eta)}{\log_{(1-\eta+8\eta^2)}(1-\eta)} \tag{16}$$

$$\frac{\sum_{t=1}^T\left(\alpha_{eu}^t + \alpha_{ev}^t\right)}{T}\cdot(1+10\eta) > \frac{(1-\eta)(1+10\eta)}{\log_{(1-\eta+8\eta^2)}(1-\eta)} \tag{17}$$

$$\alpha_{eu} + \alpha_{ev} > 1 \text{ when } \eta \in (0, 1/12). \tag{18}$$

The first inequality (Eq. (13)) follows since if $\sum_{e\in E}\frac{(1-\eta)^{\ell^T(e)}}{(1-\eta+8\eta^2)^{(1-\eta)\cdot T}} < 1$, this means that $\frac{(1-\eta)^{\ell^T(e)}}{(1-\eta+8\eta^2)^{(1-\eta)\cdot T}} < 1$ for each $e \in E$ since $\frac{(1-\eta)^{\ell^T(e)}}{(1-\eta+8\eta^2)^{(1-\eta)\cdot T}} \geq 0$. This in turn implies that $(1-\eta)^{\ell^T(e)} < (1-\eta+8\eta^2)^{(1-\eta)\cdot T}$. Eq. (14) follows by taking $\log_{(1-\eta+8\eta^2)}$ of both sides (where $(1-\eta+8\eta^2) \in (0,1)$). Then, the load on edge e is defined to be the sum of the loads assigned to it from each phase t. Thus, the total load on edge $\ell^T(e) = \sum_{t=1}^T(\alpha_{eu}^t + \alpha_{ev}^t)$. Substituting this expression into Eq. (14) results in Eq. (15). We divide both sides by T in Eq. (16). Then, we multiply both sides by $(1+10\eta)$ in Eq. (17). Finally, by definition,

[5] We use the version of the proof of Lemma 3.1 in v4 of [SV19] where Appendix B had a typo in the last steps which necessitated this change in multiplicative factor in the lower bound on the weights and the change in the potential function. Namely in the last steps, they wrote $\Phi(0) = \sum_{e\in E}\frac{(1-\eta/2)^T}{(1-\eta/2)^{(1-\eta)T}}$ which is a typo and the denominator should instead be $(1-\eta)^{(1-\eta)T}$. Thus, our analysis differs somewhat from the original analysis of Lemma 3.1 to correct for this typo.

$\alpha_{eu} = (1+10\eta)\cdot\sum_{t=1}^{T}\frac{\alpha_{eu}^t}{T}$. So, because $\frac{(1-\eta)(1+10\eta)}{\log_{(1-\eta+8\eta^2)}(1-\eta)} > 1$ for all $\eta \in (0, 1/12)$, we obtain Eq. (18), and the solutions satisfy the constraint.

We show that Φ is non-increasing as t increases; then, if initially, $\Phi(0) < 1$, then we directly prove our desired statement. The rest of the proof follows almost identically (except for the fix for the typo) from the proof of Lemma 3.1 in [SV20] but we repeat the proof for completeness. For any $0 \le t < T$, we have the following:

$$\Phi(t+1) = \sum_{e\in E}\frac{(1-\eta)^{\ell^{t+1}(e)}\cdot(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}} \tag{19}$$

$$= \sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta)^{\alpha_{eu}^t+\alpha_{ev}^t}(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}} \tag{20}$$

$$\le \sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta(\alpha_{eu}^t+\alpha_{ev}^t)+\eta^2(\alpha_{eu}^t+\alpha_{ev}^t)^2/2)(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}} \tag{21}$$

$$\le \left(\sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}}\right) - (\eta-8\eta^2)\left(\sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}}\right) \tag{22}$$

$$= (1-\eta+8\eta^2)\cdot\left(\sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta+8\eta^2)^{T-t-1}}{(1-\eta+8\eta^2)^{(1-\eta)T}}\right) \tag{23}$$

$$= \left(\sum_{e\in E}\frac{(1-\eta)^{\ell^{t}(e)}(1-\eta+8\eta^2)^{T-t}}{(1-\eta+8\eta^2)^{(1-\eta)T}}\right) = \Phi(t). \tag{24}$$

Eq. (19) is obtained by definition of the potential function $\Phi(t)$. Eq. (20) follows because $\ell^{t+1}(e) = \ell^t(e) + \alpha_{eu}^t + \alpha_{ev}^t$. Using the well-known fact that $(1-a)^b \le e^{-ab} \le 1-ab+(ab)^2/2$ for $0 \le ab \le 1$, we let $a = \eta$ and $b = \alpha_{eu}^t + \alpha_{ev}^t$; assuming $\eta \le 1/4$ ensures $0 \le \eta\,(\alpha_{eu}^t+\alpha_{ev}^t) \le 1$ since $0 \le (\alpha_{eu}^t+\alpha_{ev}^t) \le 4$. We obtain Eq. (21) by using the aforementioned inequalities. Assuming $\sum_{e\in E} w_e^t\,(\alpha_{eu}^t+\alpha_{ev}^t) \ge \sum_{e\in E} w_e^t$, we obtain Eq. (22) by lower bounding $\sum_{e\in E}(1-\eta)^{\ell^t(e)}(\alpha_{eu}^t+\alpha_{ev}^t) = \sum_{e\in E} w_e^t\,(\alpha_{eu}^t+\alpha_{ev}^t) \ge \sum_{e\in E} w_e^t = \sum_{e\in E}(1-\eta)^{\ell^t(e)}$ since by definition $w_e^t = (1-\eta)^{\ell^t(e)}$ and upper bounding $(\alpha_{eu}^t+\alpha_{ev}^t)^2 \le 16$. Eq. (23) simplifies the previous expression. Eq. (24) shows that $\Phi(t+1) \le \Phi(t)$ for all $0 \le t < T$. Hence,

$$\begin{aligned}\Phi(T) \le \Phi(T-1) \le \ldots \le \Phi(0) &= \sum_{e\in E}\frac{(1-\eta+8\eta^2)^T}{(1-\eta+8\eta^2)^{(1-\eta)T}}\\ &\le \sum_{e\in E}\exp\left(-\left(\eta^2-8\eta^3\right)T\right)\\ &\le m\cdot\exp(-\left(\eta^2-8\eta^3\right)\cdot(c_0\ln n)/\left(\eta^2-8\eta^3\right)) = O\left(\frac{1}{\mathrm{poly}(n)}\right).\end{aligned}$$

We obtain the last expression since $T = \frac{c_0\ln n}{\eta^3} \ge \frac{c_0\ln n}{\eta^2-8\eta^3}$ for large enough constant $c_0 > 0$ and $\eta \in (0, 1/12)$. The above shows that $\Phi(T) < 1$ and we conclude our proof of the lemma. ☐

Knowing the condition that whp at least one phase t exists where $\sum_{e\in E} w_e^t\,(\alpha_{eu}^t+\alpha_{ev}^t) < \sum_{e\in E} w_e^t$ allows us to prove Lemma 5.8.

Let $S(z)$ be the set of all possible *integer* assignments of weights to α_{eu}^t for all $e = \{u, v\}$ and endpoints u, v.

Lemma 5.8. *Suppose there exists a set of (non-negative) weights $\{\hat{w}_e^t\}$ obtained by our algorithm in phase t such that*

$$\sum_{e\in E}\hat{w}_e^t > \sum_{e\in E}\hat{w}_e^t\left(\hat{\alpha}_{eu}^t+\hat{\alpha}_{ev}^t\right), \tag{25}$$

where $\hat{\alpha}^t_{eu}, \hat{\alpha}^t_{ev} \in S(z)$ *is determined by our algorithm during phase* t*. Let* V''_λ *denote the set of nodes, determined by our algorithm, that are incident to at least* $\lceil z/2 \rceil + Z$ *real edges* e *with* $\hat{w}^t_e \geq \lambda$ *where* $Z \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T(4T+1)\log_{(1+\eta)} n}\right)$*. Then, there exists* λ *such that* $\delta\left(G[V''_\lambda]\right) > z - \frac{c_2 \log^4 n}{\varepsilon}$ *whp for sufficiently large constant* $c \geq 1$ *(which depends on constant parameter* $\eta \in (0, 1/12)$*).*

Proof. This proof closely follows the proof of Lemma 3.2 in [SV20] (and, hence, also the proof in [BGM14]). The main difference in our proof is to show that the statement still holds when we lose some part of the loads due to the dummy edges and the random noise Z; specifically, we care more about the instances when $Z < 0$, which is the more difficult setting to prove. First, note that $\hat{\alpha}^t_{eu} \in \{0, 2\}$ by definition of $S(z)$ and how we assign load to edges within a phase of the algorithm. Then, let $E_\lambda := \{e \in E : \hat{w}^t_e \geq \lambda\}$ be the set of edges where $\hat{w}^t_e \geq \lambda$ and let $\deg_\lambda(v)$ be the number of edges in E_λ that are incident to v. To provide some intuition, $\hat{w}^t_e$ *decreases* when the load on the edges *increases*. Thus, edges with higher weights correspond with edges with lower loads and new loads within a phase are assigned to edges in order from largest weights (lowest load) to smallest weights (highest load).

Now, we define a variable $d_\lambda(v)$ for each node v which lower bounds the the sum of current loads α^t_{eu} assigned to all *real* edges adjacent to v from the current phase t. Recall from our algorithm that out of the real edges with smallest accumulated load a noise X_v is picked that can cause more or less of these real edges to obtain new loads. Edges with weight larger than λ (equivalent to number of edges with load smaller than $\log_{(1-\eta)}(\lambda)$) receive loads starting from the largest weight to smallest weight; however, not all of the $\min(\lceil z/2 \rceil - 1, \deg(v))$ edges with the largest weight may receive loads due to the presence of dummy edges. This difference due to dummy edges is what we capture below. We *lower bound* the sum of the loads given to the highest weight edges adjacent to v whp in terms of $\min(\lceil z/2 \rceil - 1, \deg(v))$. We thus define $d_\lambda(v)$ to be

$$d_\lambda(v) = \begin{cases} 2\deg_\lambda(v), \text{ if } \deg_\lambda(v) < \lceil z/2 \rceil - \frac{cT\log^2 n}{\varepsilon} \\ \max(0, z - \frac{2cT\log^2 n}{\varepsilon}), \text{ otherwise.} \end{cases}$$

This is the key difference between our procedure and the proof of Lemma 3.3 in [SV20]. We require $\frac{2cT\log n}{\varepsilon}$ to be subtracted from the sum of the loads of adjacent edges since whp, at most $\frac{cT\log n}{\varepsilon}$ dummy adjacent edges resulting from X_v for each v are added to each node from Line 11. This means that part of the total load of z could be spent on these dummy edges, decreasing the total load on the real edges. Thus, the subtracted value of $\frac{2cT\log n}{\varepsilon}$ from $\deg_\lambda(v)$ reflects the load that is lost to these dummy edges. We first note that $d_\lambda(v) \leq \sum_{e \ni v : \hat{w}_e \geq \lambda} \hat{\alpha}^t_{ev}$ with high probability. If $\deg_\lambda(v) + \frac{cT\log n}{\varepsilon} < \lceil z/2 \rceil$, then a number of real edges plus at most $\frac{cT\log n}{\varepsilon}$ dummy edges are each given 2 load. If $\deg_\lambda(v) \geq \lceil z/2 \rceil - \frac{cT\log n}{\varepsilon}$, then $2\deg_\lambda(v) \geq z - \frac{2cT\log n}{\varepsilon}$. When $X_v > 0$, additional edges receive loads, maintaining our lower bound. Thus,

$$\sum_{v \in V} d_\lambda(v) \leq \sum_{v \in V} \sum_{e \ni v : \hat{w}^t_e \geq \lambda} \hat{\alpha}^t_{ev} = \sum_{e \in E : \hat{w}^t_e \geq \lambda} (\hat{\alpha}^t_{ev} + \hat{\alpha}^t_{eu}). \tag{26}$$

Suppose the weights are upper bounded by b: $a \leq \hat{w}^t_e \leq b$ for all $e \in E$. Then,

$$\begin{aligned} \int_a^b |E_\lambda| d\lambda &= \sum_{e \in E} \hat{w}^t_e \\ &> \sum_{e \in E} \hat{w}^t_e \left(\hat{\alpha}^t_{eu} + \hat{\alpha}^t_{ev}\right) && \text{by Eq. (25)} \\ &= \int_a^b \sum_{e \in E : \hat{w}^t_e \geq \lambda} (\hat{\alpha}^t_{eu} + \hat{\alpha}^t_{ev}) d\lambda \\ &\geq \int_a^b \sum_{v \in V} d_\lambda(v) d\lambda && \text{by Eq. (26)} \end{aligned}$$

Thus, there exists a $a \le \lambda \le b$ where $|E_\lambda| > \sum_{v \in V} d_\lambda(v)$. Recall V''_λ is the set of nodes with $\deg_\lambda(v) \ge \lceil z/2 \rceil + Z - \frac{c_1 \log^4 n}{\varepsilon}$ where $Z \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T(4T+1)\log_{(1+\eta)} n}\right)$. Let $c_2 \ge 1$ be an appropriately large constant such that $\frac{c_1 \log^4 n}{\varepsilon} \le \frac{c_2 T^2 \log^2 n}{\varepsilon}$. Then, whp,

$$|E(V''_\lambda)| \ge |E_\lambda| - \sum_{v \in V - V''_\lambda} \deg_\lambda(v) \tag{27}$$

$$> \sum_{v \in V} d_\lambda(v) - \sum_{v \in V - V''_\lambda} \deg_\lambda(v) \tag{28}$$

$$\ge \left(|V''_\lambda| \cdot \max\left(0, z - \frac{2(c+c_2)T^2 \log^2 n}{\varepsilon}\right) + \sum_{v \in V - V''_\lambda} (2 \deg_\lambda(v)) \right) - \sum_{v \in V - V''_\lambda} \deg_\lambda(v) \tag{29}$$

$$\ge |V''_\lambda| \cdot \max\left(0, z - \frac{2(c+c_2)T^2 \log^2 n}{\varepsilon}\right). \tag{30}$$

Eq. (27) follows since the number of edges in the induced subgraph of $E(V''_\lambda)$ is lower bounded by the number of edges in $|E_\lambda|$ minus the set of those edges that are adjacent to nodes not in V''_λ; this set of edges can be each be counted at most twice. Eq. (28) follows from substituting $|E_\lambda| > \sum_{v \in V} d_\lambda(v)$. Eq. (29) follows from the definition of $d_\lambda(v)$ and the fact that $Z \le \frac{cT^2 \log^2 n}{\varepsilon}$ whp for constant $c \ge 1$; thus, every $v \in V''_\lambda$ has $\deg_\lambda(v) \ge \lceil z/2 \rceil - \frac{(c+c_2)T^2 \log^2 n}{\varepsilon}$ whp which means that the load for each of these nodes is at least $2\deg_\lambda(v) \ge z - \frac{2(c+c_2)T^2 \log^2 n}{\varepsilon}$. Furthermore, we can guarantee that each node in $v \in V - V''_\lambda$ has $\deg_\lambda(v) \le \lceil z/2 \rceil - \frac{2c_2 T^2 \log^2 n}{\varepsilon} + \frac{cT^2 \log^2 n}{\varepsilon} < \lceil z/2 \rceil - \frac{cT \log^2 n}{\varepsilon}$ for appropriately large constants $c_1, c_2 \ge 1$ which means that the load given to them is $2\deg_\lambda(v)$. Finally, Eq. (30) follows because the last two terms is lower bounded by 0.

The last step is simple; since the right hand side of Eq. (30) is lower bounded by 0, then $|E(V''_\lambda)| > 0, |V''_\lambda| > 0$ and so $\frac{|E(V''_\lambda)|}{|V''_\lambda|} = \rho\left(G[V''_\lambda]\right) > z - \frac{2(c+c_2)T^2 \log^2 n}{\varepsilon}$. ☐

Lemma 5.9. *Given a* $z < \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta}$ *for appropriately large constant* $c > 0$*, then, for* $\eta \in (0, 1/12)$*, Algorithm 7 will output a subgraph of density at least* $z - O\left(\frac{\log^4 n}{\varepsilon}\right)$*, whp.*

Proof. Lemma 5.8 proves there exists a λ such that $\rho(G[V''_\lambda]) > z - O\left(\frac{\log^4 n}{\varepsilon}\right)$. Since all loads given by Algorithm 7 are integers, the load on any edge is an integer in $[0, 4T]$. This means that the weights on the edges $(1-\eta)^{\ell^t(e)}$ in the proof of Lemma 5.8 are also obtained from these integer loads. Thus, it is sufficient to look at all possible integer loads $\ell \in [0, 4T]$ which give the appropriate weight cutoffs $\lambda = (1-\eta)^\ell$ and we prove the lemma by Lemmas 5.7 and 5.8 which ensure such a λ exists.

Finally, to pass the last check just requires setting the constant $c_3 \ge 1$ to a large enough value since $|Y| = O\left(\frac{\log^4 n}{\varepsilon}\right)$ whp. By the proof above, for small enough z, we are guaranteed a set of nodes of sufficiently large density. ☐

Lemma 5.10. *Algorithm 6 returns a* $\left(1 - 12\eta, O\left(\frac{\log^4 n}{\varepsilon}\right)\right)$*-approximate densest subgraph for any constant* $\eta \in (0, 1/12)$ *with high probability.*

Proof. First, as long as $z \le \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta}$, we obtain our desired approximation. Algorithm 6 searches for the largest value of z that satisfies this constraint by searching all $(1+\eta)^i$ for all $i \in [\lfloor \log_{(1+\eta)} n \rfloor]$. In the worst case, $(1+\eta)^i = \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{1+10\eta} + y$ by some very small $y \in (0,1)$. This means that the closest $(1+\eta)^{i-1}$ is the largest value of z that is smaller than the cutoff; in other words, $z = (1+\eta)^{i-1} \ge \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{(1+\eta)(1+10\eta)}$. Thus, by Lemma 5.9, we obtain a set of nodes whose induced subgraph has density at least $z - \frac{c \log^4 n}{\varepsilon}$ for an

appropriately large constant $c \geq 1$. Together, the returned set of nodes has density at least $z - \frac{c \log^4 n}{\varepsilon} \geq \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{(1+\eta)(1+10\eta)} - \frac{c \log^4 n}{\varepsilon} \geq (1 - 12\eta)D^* - O\left(\frac{\log^4 n}{\varepsilon}\right)$. In the case when no value of z returns a set of nodes, all the nodes in the graph are returned. This case occurs when all values of z passed into Algorithm 7 are too large which means that $1 \geq \frac{D^* - \frac{c \log^3 n}{\varepsilon \eta^3}}{(1+10\eta)}$ and $D^* \leq \frac{c \log^3 n}{\varepsilon \eta^3} + 1 + 10\eta$ and returning all the nodes in the graph will result in an induced subgraph with density $\rho(G) \leq D^* \leq \frac{c \log^3 n}{\varepsilon \eta^3} + 1 + 10\eta = O\left(\frac{\log^4 n}{\varepsilon}\right)$ which falls within our additive error bound. □

5.4 Efficiency Analysis

We analyze the runtime of our ε-DP densest subgraph algorithm in this section. Specifically, we show that we obtain the same runtime as the original non-private algorithm. As before, we assume that $\eta \in (0, 1/12)$ is constant. Throughout our runtime analysis, we use the standard assumption that obtaining a sample from the symmetric geometric distribution requires $O(1)$ time. Such an assumption is reasonable for the following reason. One can generate a random variable from the geometric distribution in $O(1)$ time by generating a uniformly random real number in binary (up to a certain precision) and outputting the position of the first 1. A recent paper of Balcer and Vadhan [BV18] shows how to generate such random variables on finite memory machines. Once such a random variable has been generated, we can generate a *symmetric geometric* random variable by first deciding whether the random variable is 0, positive, or negative with appropriate probability determined by the parameter b. If the chosen symmetric random variable is either positive or negative, then we use the random variable generated by the geometric distribution with the appropriate sign as the output.

Theorem 5.11. *The runtime of Algorithm 6 is* $O\left((n+m)\log^3 n\right)$.

Proof. This analysis assumes the input parameter $\eta \in (0, 1/12)$ is constant. First, the search in Line 5 of Algorithm 6 requires $O(\log n)$ iterations of Algorithm 7, which means that the runtime of Algorithm 7 is multiplied by $O(\log n)$. Next, we determine the runtime of Algorithm 7. For a given z, our algorithm runs through $T = O(\log n)$ phases. Each phase requires $O(m)$ time to sort the loads (using radix sort) and to determine the new loads to apply to each edge. Then, we run $O(\log n)$ trials of finding a subgraph of sufficient density. Each of these trials requires $O(m)$ time. Altogether, the runtime for a fixed parameter z of this algorithm is $O\left((m+n)\log^2 n\right)$. Thus, in total, our running for Algorithm 6 is dominated by $O(\log n)$ iterations of Algorithm 7 which take a total of $O\left((n+m)\log^3 n\right)$ time. □

6 Differential Privacy from Locally Adjustable Algorithms

The graph algorithms studied in this paper all have a generalizable structure that is conducive to privacy. Our framework applies to a number of parallel, distributed, and dynamic algorithms [LSY+22, BHNT15, SV20, HNW20, BGM14, CSS21], described in detail in Sections 4 and 5. In this section, we describe a general class of algorithms and show that we can transform them to be ε-edge DP (Section 6.2). Some even become ε-LEDP (Section 6.3). We call the algorithms that have these characteristics *locally adjustable* algorithms. Beyond the various algorithms we study in this paper, we believe that our generalization and the privacy framework can be applied to a broader set of non-private graph algorithms, most naturally, in the parallel and distributed settings. Furthermore, interestingly, the techniques that we use to obtain our privacy guarantees result in only a small *additive* polylogarithmic error while maintaining the *same multiplicative* approximation factor of each of the original non-private algorithms given in Sections 4 and 5. However, we do not have a general statement bounding the utility (or error) of private algorithms obtained via our framework and leave this as an interesting open question.

6.1 Locally Adjustable Graph Algorithms

We call a graph algorithm $\mathcal{A}$ on input graph $G = (V, E)$ ***locally adjustable*** if it has the following characteristics. The algorithm proceeds in at most K total phases. Two or more phases may occur in parallel if the pairs of phases $k_1, k_2 \leq K$ do not depend on each other. Each node v maintains an internal state $I_{v,p}$

parameterized by the phase number p; similarly, each edge e maintains an internal state $I_{e,p}$ also parameterized by p. Initially, in phase 0, all states are set to default identical values. Since the phases may be processed in parallel, for each phase p, we refer to the previous phase that phase p depends on as $\tilde{p}$, where $\tilde{p} < p$, but $\tilde{p}$ is not necessarily equal to $p - 1$.

During each of the at most K phases, each node $v \in V$ computes a function using *only* the previous states $I_{w,\tilde{p}}, I_{e,\tilde{p}}$ of its immediate one-hop neighborhood, where $w \in N(v)$ and $e = \{v, a\}$ for any $a \in N(v)$. Notably, these functions determine for a node v whether its neighbors and/or its incident edges satisfy a condition. Then, v uses another function with the *number* of neighbors or incident edges that satisfy the condition as input to compute a new state. Each edge $e \in E$ also computes a function using *only* information *received* from its two endpoints. Formally, these functions are defined in the following paragraphs. Importantly, all of the functions satisfy a "local" property where any edge insertion or deletion, $e' = (u, v)$, in the graph affects the count of the number of neighbors that satisfy the condition of only u or v and no other nodes. The output of the function is not changed for any other node $w \notin \{u, v\}$ or any other edge $e = (i, j)$ where neither i or j is u or v. This is the *locally adjustable* specification of the type of algorithms that we are considering.

Node functions Let B be a predicate that can be satisfied (or not) by a neighbor $w \in N(v)$ of v. Node v has a deterministic function that is evaluated in each phase p:

$$\texttt{adj-neighb}_v(w) = \begin{cases} 1, \text{ if } w \in N(v) \text{ and } I_{w,\tilde{p}} \\ \quad \text{satisfies condition } B \\ 0, \text{ otherwise} \end{cases}$$

that takes as input a neighbor, $w \in N(v)$, of v and outputs a 0 or 1 bit for the neighbor indicating whether the neighbor satisfies B using the previous state of the neighbor $I_{w,\tilde{p}}$. Whether w satisfies B is determined by the state $I_{w,\tilde{p}}$ of node w, parameterized by the last phase $\tilde{p}$ that p depends on.

For clarity, we emphasize a few crucial observations regarding B. Suppose, without loss of generality, that an edge $e = \{u, v\}$ is inserted in the beginning of phase p. This means that only the count of the number of neighbors of u or v that satisfy B is affected compared to the case when e is not inserted. Specifically, this count can increase by at most 1. Since the previous states $I_{v,p'}$ for all $p' < p$ are fixed prior to the insertion, the insertion cannot affect the output $\texttt{adj-neighb}_v(w)$ for any other $w \neq u \in N(v)$. Hence, the count for the number of neighbors of v that satisfy B increases by 1 (compared to the case when e is not inserted) when $\texttt{adj-neighb}_v(u) = 1$. Symmetrically, for an edge deletion, the number of neighbors that satisfy B can decrease by at most 1 compared to the case when e is not deleted. If the update is not incident to a node w, then whether B is satisfied or not *does not change* for any neighbor of w. This means that $\texttt{adj-neighb}_v$ is a "local" function where edge updates in the graph can only affect their incident endpoints.

Similarly, let C be a condition that can be satisfied (or not) by an incident edge to v. Again, we specify the trivial "local" requirement for C that the addition or deletion of any edge $e = \{u, v\}$ (with an arbitrary state $I_{e,\tilde{p}}$) at the beginning of phase p changes whether C is satisfied only for edge e; this is a trivial requirement since edge e did not exist prior to the insertion of e (and the edge e no longer exists after the deletion of e). Then, v has another deterministic function that is also evaluated in phase p,

$$\texttt{adj-edge}_v(\{v, w\}) = \begin{cases} 1, \text{ if } \{v, w\} \in E \text{ and } I_{\{v,w\},\tilde{p}} \\ \quad \text{satisfies condition } C \\ 0, \text{ otherwise} \end{cases}$$

that takes as input an incident edge to v and outputs a 0 or 1 bit depending on whether the edge satisfies C. As before, whether $\{v, w\}$ satisfies C depends on the previous state $I_{\{v,w\},\tilde{p}}$ of the edge $\{v, w\}$.

Let $\texttt{update-node-state}_v$ be a deterministic function that updates the state of v using only the *number* of neighbors, $n_{v,p}$ and/or edges, $e_{v,p}$, that satisfy the conditions B and C, respectively. Notably, the function does not require knowledge about the state of the neighbors or edges that satisfy the condition. Specifically, let $n_{v,p} = |\{w \in N(v) : \texttt{adj-neighb}_v(w) = 1\}|$ and $e_{v,p} = |\{\{v, w\} \in E : \texttt{adj-edge}_v(\{v, w\}) = 1\}|$. In phase p, the function $\texttt{update-node-state}_v(I_{v,\tilde{p}}, n_{v,p}, e_{v,p}) \to I_{v,p}$ outputs the next state of v provided the state $I_{v,\tilde{p}}$ of v from the most recent phase $\tilde{p}$ that p depends on and the computed values $n_{v,p}$ and $e_{v,p}$.

Finally, on the set of incident edges to v that satisfy condition C, $\{\{v, w\} \in E : \texttt{adj-edge}_v(\{v, w\}) = 1\}$, $\mathcal{A}$ uses each edge's function, $\texttt{update-edge-state}_{\{v,w\}}$, to update the state of the edge, the details of which are given next.

Edge function Each edge $e = \{u, v\}$ has a function $\texttt{update-edge-state}_e$ that takes its previous edge state and real-valued inputs from its adjacent nodes and outputs its current state. Namely, in phase p, edge e computes $\texttt{update-edge-state}_e(I_{e,\tilde{p}}, i_u, i_v) \to I_{e,p}$ to determine its next state $I_{e,p}$ using its previous state $I_{e,\tilde{p}}$ and inputs from its endpoints, i_u and i_v.

The algorithm proceeds with the next phases until a *stopping condition* is satisfied for the entire graph. The stopping function of the algorithm is based on a *threshold function*, which takes as input the states of the nodes and edges computed in the current phase p. If the number of nodes/edges that satisfy the condition exceeds a threshold, then the algorithm terminates. Specifically, these stopping functions are defined as follows.

Stopping functions The stopping functions determine whether the algorithm stops running or continues running with the next phase. There are stopping functions for each individual node and also a global stopping function that determines whether a certain number of nodes and edges that satisfy a condition F is at least some threshold T. The individual stopping function prevents a particular node from participating in the next phases while a global stopping function stops the algorithm. The individual stopping function for each node, $\texttt{stop}_v$, relies on how many neighbors' states or neighboring adjacent edges' states satisfy a condition F. We denote the number of neighbors and adjacent edges that satisfy F by $s_{v,p} = |\{w \in N(v) : I_{w,p} \text{ satisfies } F\}|$ and $t_{v,p} = \left|\left\{w \in N(v) : I_{\{v,w\},p} \text{ satisfies } F\right\}\right|$, respectively. Then, we define $\texttt{stop}_v$ as follows, for some fixed constants $c_1, c_2 \geq 0$ and fixed threshold $T \geq 0$:

$$\texttt{stop}_v\left(s_{v,p}, t_{v,p}\right) = \begin{cases} 1, \text{ if } c_1 \cdot s_{v,p} + c_2 \cdot t_{v,p} \geq T; \\ 0, \text{ otherwise.} \end{cases}$$

We define the global stopping function, in each phase p, using $s_p = |\{v \in V : I_{v,p} \text{ satisfies } F\}|$ and $t_p = |\{e \in E : I_{e,p} \text{ satisfies } F\}|$, for some fixed constants $c_3, c_4 \geq 0$ and fixed threshold $T \geq 0$:

$$\texttt{global-stop}\left(s_p, t_p\right) = \begin{cases} 1, \text{ if } c_3 \cdot s_p + c_4 \cdot t_p \geq T \ ; \\ 0, \text{ otherwise.} \end{cases}$$

Output function Once the algorithm terminates, each node outputs an answer to the problem using output functions $\texttt{out}_v(I_{v,p}) \to \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. There may exist a global function $\texttt{global-out}(\{I_{c,p} : c \in V \cup E\}) \to \mathbb{Z}$, which takes the internal states of the nodes and edges and outputs an integer answer.

A non-trivial number of parallel and distributed graph algorithms are locally adjustable, including the non-private algorithms from prior sections for k-core decomposition, densest subgraphs, and low out-degree orderings. In the next section, we discuss how to obtain ε-edge DP graph algorithms from locally adjustable graph algorithms.

6.2 Edge Differential Privacy from Local Adjustability

We first show how to obtain ε-edge DP locally adjustable algorithms. Then, a slight modification to the locally adjustable conditions also allows us to obtain ε-*LEDP* algorithms. We leave as an interesting open question proving general utility bounds for our framework.

The main idea here is to show that, on edge-neighboring graphs $G = (V, E)$ and $G' = (V, E')$, the probability that the *same* states are maintained over the at most K phases satisfy Definition 2.7. We prove this by conditioning on the states from previous phases. Then, we show via the chain rule that this implies that our algorithm is ε-edge DP. To do this, we make several modifications to the node, edge, stopping, and output functions. Our privacy framework is given as follows:

Privacy Framework Suppose we are provided a locally adjustable algorithm $\mathcal{A}$. Then we formulate the following mechanism $\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, \mathcal{A}, p) \to (\mathcal{I}_p, \mathcal{O})$ performed by the curator which takes as input the graph

G, the node and edge states of G from phase $\tilde{p}$, the locally adjustable algorithm $\mathcal{A}$, and the phase number p. Mechanism $\mathcal{M}$ outputs the next set of states $\mathcal{I}_p$ for phase p if the algorithm is still running or $\emptyset$ if the algorithm has stopped. It also outputs the set of outputs $\mathcal{O}$ if the algorithm has stopped or $\emptyset$ if the algorithm is still running. The mechanism modifies $\mathcal{A}$ in the following ways:

- For each node v, the node computes $\hat{n}_{v,p}$ and $\hat{e}_{v,p}$ by first sampling $X_{v,p}, Y_{v,p} \sim \mathsf{Geom}(\varepsilon/20K)$ and calculates $\hat{n}_{v,p} = n_{v,p} + X_{v,p}$ and $\hat{e}_{v,p} = e_{v,p} + Y_{v,p}$. Node v uses $\hat{n}_{v,p}$ and $\hat{e}_{v,p}$ instead of $n_{v,p}$ and $e_{v,p}$ in $\texttt{update-node-state}_v(I_{v,\tilde{p}}, \hat{n}_{v,p}, \hat{e}_{v,p})$.
- When determining the stopping condition, node v samples $S_{v,p}, T_{v,p} \sim \mathsf{Geom}(\varepsilon/(20K))$ and computes $\hat{s}_{v,p} = s_{v,p} + S_{v,p}$ and $\hat{t}_{v,p} = t_{v,p} + T_{v,p}$. Node v uses $\hat{s}_{v,p}$ and $\hat{t}_{v,p}$ instead of $s_{v,p}$ and $t_{v,p}$ in $\texttt{stop}_v(\hat{s}_{v,p}, \hat{t}_{v,p})$. The curator also samples $S_p, T_p \sim \mathsf{Geom}(\varepsilon/(5K))$ and computes $\hat{s}_p = s_p + S_p$ and $\hat{t}_p = t_p + T_p$. The curator uses $\hat{s}_p$ and $\hat{t}_p$ as input into $\texttt{global-stop}(\hat{s}_p, \hat{t}_p)$.
- Let $GS_{\texttt{global-out}}$ and $GS_{\texttt{out}_v}$ be the global sensitivities of $\texttt{global-out}$ and $\texttt{out}_v$, respectively. To compute the output after satisfying the stopping condition, each node v samples $Q_{v,p} \sim \mathsf{Geom}(\varepsilon/(10 \cdot GS_{\texttt{out}_v} \cdot K))$ and outputs $\texttt{out}_v(I_{v,p}) + Q_{v,p}$. Then, the curator samples $W_p \sim \mathsf{Geom}(\varepsilon/(5 \cdot GS_{\texttt{global-out}} \cdot K))$ and outputs $\texttt{global-out}(\mathcal{I}_p) + W_p$.

The main intuition behind our privacy framework is derived from our ε-edge DP densest subgraph algorithm (Section 5) regarding the creation of ***dummy*** edges that satisfy the conditions of the various functions. These dummy edges account for the case when the extra edge $e' \in E' \setminus E$ where $e' \in E'$ satisfies any of the functions. The number of these dummy edges that are added for each node is drawn from a symmetric geometric distribution using the sensitivities of the appropriate functions in Corollaries 6.2 and 6.4 and Lemmas 6.1 and 6.3.

We now prove some useful lemmas that will help us in proving the privacy of our framework.

Sensitivity We first make a few intuitive observations before proving the privacy of our DP framework. First, instead of using the global sensitivity of function $\texttt{update-node-state}_v$, for node v, we instead use the sensitivity of a function F which takes a graph $G = (V, E)$, a phase p, and a set of valid states, $\mathcal{I}_V$, for all nodes $v \in V$ and a set of valid states, $\mathcal{I}_{V \times V}$, for every pair of two nodes. Function $F(G, p, \mathcal{I}_V, \mathcal{I}_{V \times V})$ then outputs $\sum_{v \in V} n_{v,p} + \sum_{e \in E} e_{v,p}$ for phase p using the states given in I_V and $I_{V \times V}$ as the previous states (assuming p, I_V and $I_{V \times V}$ are public). By definition of $n_{v,p}$ and $e_{v,p}$, we can in fact bound the sensitivity of F (on edge-neighboring inputs $G = (V, E)$ and $G' = (V, E \cup \{u, v\})$).

Lemma 6.1. *The sensitivity of function $F(\mathbf{a}_v, p, \mathcal{I}_V, \mathcal{I}_{V \times V})$ that computes the sum $\sum_{v \in V} n_{v,p} + \sum_{e \in E} e_{v,p}$ for a given phase p using the states of neighbors in I_V and adjacent edges in $\mathcal{I}_{V \times V}$ is 4.*

Proof. The count $n_{v,p}$ for each node v is the *number* of neighbors w of v for which $\texttt{adj-neighb}_v(w)$ outputs 1. Suppose $\{u, v\} \in E' \setminus E$. Then, $\texttt{adj-neighb}_v(u)$ may equal 1 (and similarly $\texttt{adj-neighb}_u(v)$ may equal 1). The edge would not affect $n_{w,p}$ of any $w \notin \{u, v\}$. Thus, the additional edge in E' can make at most two endpoints increase their $n_{v,p}$, each by 1, and $\sum_{v \in V} n_{v,p}$ increases by at most 2. By the same argument, $\sum_{e \in E} e_{v,p}$ increases by at most 2. ☐

Suppose, we have another function $f_n(\mathbf{a}_v, p, \mathcal{I}_V, \mathcal{I}_{V \times V})$ that takes as input the adjacency list of a node $\mathbf{a}_v$ and outputs $n_{v,p}$ using the states of neighbors and adjacent edges in $\mathcal{I}_V$ and $\mathcal{I}_{V \times V}$. (Define the function f_e similarly except it outputs $e_{v,p}$.) An immediate corollary of the above proof bounds the sensitivity of $n_{v,p}$ and $e_{v,p}$.

Corollary 6.2. *The sensitivity of f_n for phase p is 1. The sensitivity of f_e is also 1.*

We define similar functions to the above for $s_{v,p}, t_{v,p}, s_p, t_p$. Namely, we pass in the states I_V and $I_{V \times V}$ into the functions (so they are public information) and compute the sensitivity for edge-neighboring inputs whose states come from I_V and $I_{V \times V}$. For simplicity, we do not define these functions explicitly and instead give the sensitivity for $s_{v,p}, t_{v,p}, s_p, t_p$ in lieu of these functions. Our framework also relies on the sensitivity of $s_{v,p}$ and $t_{v,p}$, whose proof is identical to Lemma 6.1.

Lemma 6.3. *The sensitivity of $s_{v,p}$ for phase p is 1. The same holds for $t_{v,p}$.*

We bound the sensitivity of $s_p + t_p = |\{c \in (V \cup E) : I_{c,p} \text{ satisfies } F\}|$.

Corollary 6.4. *The sensitivity of $s_p = |\{v \in V : I_{v,p} \text{ satisfies } F\}|$ for any phase p is 0. The sensitivity of t_p is 1.*

Proof. At most one additional edge, $I_{e',p}$ in G' can satisfy F. Thus, t_p increases by at most 1. All nodes remain the same and hence, s_p does not change. Thus, the sensitivity is 1. □

We first show the following for $\mathcal{M}$.

Lemma 6.5. *For any pair of edge-neighboring graphs $G = (V, E)$ and $G' = (V, E \cup \{e'\})$, let e' be the edge that is present in E' but not in E. Let R and R' be the events where given valid state values $state_{v,\tilde{p}}$ and $state_{e,\tilde{p}}$ (for all $v \in V, e \in E$), for all $I_{v,\tilde{p}}, I_{e,\tilde{p}} \in \mathcal{I}_{\tilde{p}}$ and $I'_{v,\tilde{p}}, I'_{e,\tilde{p}} \in \mathcal{I}'_{\tilde{p}}$, it holds that $I_{v,\tilde{p}} = I'_{v,\tilde{p}} = state_{v,\tilde{p}}$ for all $v \in V$ and $I_{e,\tilde{p}} = I'_{e,\tilde{p}} = state_{e,\tilde{p}}$ for all $e \in E$. Conditioned on R and R', the following hold for all $S \in Range(\mathcal{M})$ on inputs A, and p:*[6]

$$\frac{1}{\exp(\varepsilon/K)} \leq \frac{\mathsf{Pr}\left[\mathcal{M}(G', \mathcal{I}'_{\tilde{p}}, A, p) \setminus \{I_{e,p}\} = S \mid R'\right]}{\mathsf{Pr}\left[\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, A, p) = S \mid R\right]} \leq \exp(\varepsilon/K).$$

Proof. In each phase p, mechanism $\mathcal{M}$ uses the states of the edges and nodes from the most recent phase $\tilde{p}$ it depends on and evaluates the node, edge, stopping and output functions. Each function $\texttt{adj-neighb}_v$ is deterministic. By Corollary 6.2, $n_{v,p}$ increases by at most 1 in G' compared to G conditioned on the events R and R'. In the below proofs, we remove the conditioning on the previous states for simplicity in notation, but all probabilities are conditioned on the events R and R'. Similarly, $\texttt{adj-edge}_v$ is also deterministic and $e_{v,p}$ increases by at most 1, by Corollary 6.2, in G' compared to G. Since $\texttt{update-node-state}_v$ is a deterministic function on $I_{v,\tilde{p}}$, $\hat{n}_{v,p}$, and $\hat{e}_{v,p}$, the outputs of the functions are equal in G and G' if $\hat{n}_{v,p}$ and $\hat{e}_{v,p}$ are equal in G and G'. What remains here is to bound the probabilities that $\hat{n}_{v,p}$ and $\hat{e}_{v,p}$ equal particular values in $\mathbb{Z}$ for both G and G'. Let $\hat{n}_{v,p}$ and $\hat{e}_{v,p}$ represent the values in G and $\hat{n}'_{v,p}$ and $\hat{e}'_{v,p}$ represent the values in G'.

Conditioned on R and R', $\hat{n}_{v,p}$ is drawn from a conditional symmetric geometric distribution (SGD) with mean $n_{v,p}$ and $\hat{e}_{v,p}$ is drawn from a conditional SGD with mean $e_{v,p}$. By Corollary 6.2, $\hat{n}'_{v,p}$ is drawn from a conditional SGD with mean $n_{v,p} \leq n'_{v,p} \leq n_{v,p} + 1$ (and similarly for $\hat{e}'_{v,p}$). All the distributions have the same parameter, $\varepsilon/20K$, by definition of our mechanism. Thus, the conditional distributions from which $\hat{n}_{v,p}$ and $\hat{n}'_{v,p}$ are drawn are ε-indistinguishable (see the beginning of Section 5.2 for the definition). The same holds for all other pairs of random variables for nodes v and u in G and G'. There are four pairs of $\varepsilon/(20K)$-indistinguishable conditional distributions. We now show that they are independent. Conditioned on R and R', the values $n_{v,p}, e_{v,p}, n_{u,p}$ and $e_{u,p}$ are fixed and the noises drawn for each of these values are independent. Thus, the distributions are independent and the joint distributions for $(\hat{n}_{v,p}, \hat{e}_{v,p}, \hat{n}_{u,p}, \hat{e}_{u,p})$ and $(\hat{n}'_{v,p}, \hat{e}'_{v,p}, \hat{n}'_{u,p}, \hat{e}'_{u,p})$ are $4 \cdot \varepsilon/(20K) = \varepsilon/(5K)$-indistinguishable.

We just showed that the new states of the nodes are the same for G and G' with similar probabilities. Now, we condition on the outputs of $\texttt{update-node-state}_v$ being identical in G and G' for every $v \in V$. This means that the inputs sent to the adjacent edges will be identical. Since the function $\texttt{update-edge-state}_e$ for each $e \in E \cap E'$ is deterministic, all $I_{e,p} = I'_{e,p}$ for $e \in E \cap E'$.

Finally, we must determine the outputs of the stopping functions and the final outputs of the mechanism are identical in G and G'. Since each $\texttt{stop}_v$ is a deterministic function, it will output the same output in G and G' as long as $\hat{s}_{v,p} = \hat{s}'_{v,p}$ and $\hat{t}_{v,p} = \hat{t}'_{v,p}$. By Lemma 6.3, the sensitivity of $s_{v,p}$ is 1 (same for $t_{v,p}$). We now condition on the events W and W' where $I_{v,p} = I'_{v,p} = state_{v,p}, I_{e,p} = I'_{e,p} = state_{e,p}$. Hence, by the same argument as above using Lemma 6.3, the fact that $s_{v,p}, t_{v,p}, s'_{v,p}, t'_{v,p}$ are fixed conditioned on W and W', and the independence of the noises, $\hat{s}_{v,p}$ and $\hat{s}'_{v,p}$ (resp. $\hat{t}_{v,p}$ and $\hat{t}'_{v,p}$) are drawn from $\varepsilon/(20K)$-indistinguishable conditional distributions.

The same argument holds for the global stopping function. Given that the stopping functions are met, the $(\varepsilon/5K)$-edge DP of the outputs hold by Lemma 6.5 provided the sensitivities of $\texttt{out}_v$ and $\texttt{global-out}$ are $GS_{\texttt{out}_v}$ and $GS_{\texttt{global-out}}$, respectively.

Altogether, by the chain rule over the probabilities that the outputs of all functions, $\texttt{update-node-state}_v$, $\texttt{update-edge-state}_e$, $\texttt{stop}_v$, $\texttt{global-stop}$, $\texttt{out}_v$, and $\texttt{global-out}$ are identical, we obtain the conditions of the lemma. □

[6] We abuse notation slightly to indicate $(\mathcal{I}_{\tilde{p}} \setminus \{I_{e,p}\}, \mathcal{O})$ by $\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, \mathcal{A}, p) \setminus \{I_{e,p}\}$.

Let $\hat{\mathcal{A}}$ be the algorithm obtained from locally adjustable algorithm A that runs $\mathcal{M}$ during each phase of the algorithm. $\hat{\mathcal{A}}$ outputs $\mathcal{O}$ obtained from $\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, \mathcal{A}, p)$ We now show our main theorem.

Theorem 6.6. *$\hat{\mathcal{A}}$ is ε-edge DP provided locally adjustable algorithm A.*

Proof. Given K as the upper bound on the number of phases used by the algorithm, we show this theorem via induction on the phase number $1 \leq p \leq K$ using Lemma 6.5. We show that in the p-th phase, the ratio of the probabilities that the outputs in G and G' for phase p equals $Z_p \in \mathit{Range}(\mathcal{M})$ is upper bounded by $\exp(\varepsilon \cdot p/K)$, conditioned on the outputs of the previous phases being equal to $Z_1, \ldots, Z_{p-1}$. Our base case consists of the first phase of the algorithm when the states of nodes and edges are identical as they are set to default identical values. By Lemma 6.5, since $\mathcal{I}_0 = \mathcal{I}'_0$, the ratio of the probabilities that the outputs are both equal to Z_1 is upper bounded by $\exp(\varepsilon/K)$.

We assume for our induction step that in the p-th phase, the ratio of the probabilities is upper bounded by $\exp(\varepsilon \cdot p/K)$, conditioned on the outputs of the previous phases, and prove this for phase $p+1$. Let the previous phase that phase $p+1$ depends on be $\tilde{p}$. By our induction hypothesis, we have that the probability ratio is upper bounded by $\exp(\varepsilon \cdot p/K)$ and that $\mathcal{I}_p = \mathcal{I}'_p \setminus \{I'_{e,p}\}$ where $e \in E' \setminus E$. Let W be the event that the outputs of the previous phases equal $Z_1, \ldots, Z_{p-1}$.

Then, since G, A, and $p+1$ are fixed, by Lemma 6.5, $\frac{\mathsf{Pr}[\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, A, p+1) = Z_{p+1} | W]}{\mathsf{Pr}\left[\mathcal{M}(G', \mathcal{I}'_{\tilde{p}}, A, p+1) \setminus \{I_{e,p+1}\} = Z_{p+1} | W\right]} \leq \exp(\varepsilon/K)$ and $\frac{\mathsf{Pr}\left[\mathcal{M}(G', \mathcal{I}'_{\tilde{p}}, A, p+1) \setminus \{I_{e,p+1}\} = Z_{p+1} | W\right]}{\mathsf{Pr}[\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, A, p+1) = Z_{p+1} | W]} \leq \exp(\varepsilon/K)$. Then, by the chain rule over all the conditional probabilities, the ratio of the probabilities that the outputs equal $Z_1, \ldots, Z_{p+1}$ is upper bounded by $\exp(\varepsilon \cdot p/K) \cdot \exp(\varepsilon/K) = \exp(\varepsilon \cdot (p+1)/K)$.

Since K is an upper bound on the total number of phases, the ratio of the probabilities for all phases up to when the algorithm terminates is upper bounded by $\exp(\varepsilon \cdot K/K) = \exp(\varepsilon)$. After the algorithm $\hat{\mathcal{A}}$ terminates, the internal states $\mathcal{I}_K = \mathcal{I}'_K = \emptyset$ (i.e., the curator outputs empty sets for the states of the nodes and edges).

Finally, it is sufficient to show that $\frac{\mathsf{Pr}[\hat{A}(G) = Z]}{\mathsf{Pr}[\hat{A}(G') = Z]} \leq \exp(\varepsilon)$ and $\frac{\mathsf{Pr}[\hat{A}(G') = Z]}{\mathsf{Pr}[\hat{A}(G) = Z]} \leq \exp(\varepsilon)$ for all $Z \in \mathit{Range}(\hat{A})$ for *discrete* probability distributions because for any $S \subseteq \mathit{Range}(\hat{A})$, $\frac{\mathsf{Pr}[\hat{A}(G) \in S]}{\mathsf{Pr}[\hat{A}(G') \in S]}$ is given by $\frac{\mathsf{Pr}[\hat{A}(G) = S_1] + \cdots + \mathsf{Pr}[\hat{A}(G) = S_j]}{\mathsf{Pr}[\hat{A}(G') = S_1] + \cdots + \mathsf{Pr}[\hat{A}(G') = S_j]}$ (for all $S_1, \ldots, S_j \in S$). This expression can be simplified to

$$\frac{\mathsf{Pr}[\hat{A}(G) = S_1] + \cdots + \mathsf{Pr}[\hat{A}(G) = S_j]}{\mathsf{Pr}[\hat{A}(G') = S_1] + \cdots + \mathsf{Pr}[\hat{A}(G') = S_j]} \leq \frac{\exp(\varepsilon) \cdot \mathsf{Pr}[\hat{A}(G') = S_1] + \cdots + \exp(\varepsilon) \cdot \mathsf{Pr}[\hat{A}(G') = S_j]}{\mathsf{Pr}[\hat{A}(G') = S_1] + \cdots + \mathsf{Pr}[\hat{A}(G') = S_j]} = \exp(\varepsilon).$$

Hence, $\hat{\mathcal{A}}$ is ε-edge DP since the algorithm outputs identical outputs when it terminates, with probability ratio bounded by $\exp(\varepsilon)$. □

6.3 ε-LEDP from Locally Adjustable Algorithms

Our framework can also be extended to the ε-LEDP setting. In fact, our ε-LEDP k-core decomposition result is based on this framework. Our modified LEDP framework modifies our DP framework in the following ways. We remove the $\texttt{adj-edge}_v(\{v, w\})$ and $\texttt{update-edge-state}_{\{v,w\}}$ functions for edges. We also remove all global functions `global-stop` and `global-out` since the curator no longer has access to the private graph. We modify all $\texttt{update-node-state}_v$ so they no longer have $e_{v,p}$ as input. Then, we modify all $\texttt{stop}_v(s_{v,p})$ functions so they only take as input $s_{v,p}$ for each $v \in V$ and no longer take the states of adjacent edges. We prove that the modified functions give a ε-LEDP algorithm for any locally adjustable algorithm $\mathcal{A}$ using the modified framework we give below.

Modified Privacy Framework Suppose we are provided a locally adjustable algorithm $\mathcal{A}$ by our modified definition above. Then, we formulate the following distributed mechanism $\mathcal{M}(\mathbf{a}_i, \mathcal{I}_{\tilde{p}}, \mathcal{A}, p) \to (I_{i,p}, \mathcal{O}_i)$ performed by the nodes in the input the graph G; the mechanism takes as input adjacency list $\mathbf{a}_i$ for node i, a set of public (previous) node states $\mathcal{I}_{\tilde{p}}$ of G, the locally adjustable algorithm $\mathcal{A}$, and the phase number p. Mechanism $\mathcal{M}$ outputs the next state $I_{i,p}$ of node i for phase p if the algorithm is still running or $\emptyset$ if the algorithm has stopped. It also outputs the set of outputs $\mathcal{O}_i$ for node i if the algorithm has stopped or $\emptyset$ if the algorithm is still running. The mechanism modifies $\mathcal{A}$ in the following ways:

- For each node i, the node computes $\hat{n}_{i,p}$ by using the public node states, sampling $X_{i,p} \sim \mathsf{Geom}(\varepsilon/6K)$ and calculating $\hat{n}_{i,p} = n_{i,p} + X_{i,p}$. Node i uses $\hat{n}_{i,p}$ instead of $n_{i,p}$ in $\texttt{update-node-state}_v(I_{i,\tilde{p}}, \hat{n}_{i,p})$ where $I_{i,\tilde{p}}$ is from $\mathcal{I}_{\tilde{p}}$.
- When determining the stopping condition, node i samples $S_{i,p} \sim \mathsf{Geom}(\varepsilon/(6K))$ and computes $\hat{s}_{i,p} = s_{i,p} + S_{i,p}$. Node i uses $\hat{s}_{i,p}$ instead of $s_{i,p}$ in $\texttt{stop}_i(\hat{s}_{i,p})$.
- Let $GS_{\texttt{out}_i}$ be the global sensitivity of $\texttt{out}_i$. To compute the output after satisfying the stopping condition, each node i samples $Q_{i,p} \sim \mathsf{Geom}(\varepsilon/(6 \cdot GS_{\texttt{out}_i} \cdot K))$ and outputs $\texttt{out}_i(I_{i,p}) + Q_{i,p}$.

Theorem 6.7. *$\hat{\mathcal{A}}$ is ε-LEDP provided locally adjustable algorithm A.*

Proof. We implement our LEDP algorithm using the following local randomizers. First, by a modified version of Lemma 6.1 and Lemma 2.14, our mechanism implements an $(\varepsilon/6)$-LR for computing the output of $\texttt{update-node-state}_v$. Similarly, by Lemma 6.3 and Lemma 2.14, our mechanism implements a $(\varepsilon/6)$-LR for obtaining the output of $\texttt{stop}_i$. Finally, by Lemma 2.14 and because the global sensitivity of $\texttt{out}_i$ is $GS_{\texttt{out}_i}$, our mechanism implements a $(\varepsilon/3)$-LR for producing the output of each node. Finally, by Theorem 2.15 and Lemma 2.16 over all calls to LRs over the K phases, our algorithm is ε-LEDP. □

Acknowledgements

We thank Talya Eden for helpful discussions. This research was supported by DOE Early Career Award #DE-SC0018947, NSF CAREER Award #CCF-1845763, Google Faculty Research Award, Google Research Scholar Award, DARPA SDH Award #HR0011-18-3-0007, and Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

A Proof of the Adaptive Composition Theorem

Here we present the proof of Theorem 2.15 ([DMNS06, DL09, DNPR10]) for completeness.

Proof of Theorem 2.15. Let $\mathcal{M}$ be an ε-LDP mechanism and $\boldsymbol{y} = (y_1, \ldots, y_{p-1})$ be a set of p outputs for adaptively chosen adjacent inputs $\boldsymbol{x} = (x_0, x_1, \ldots, x_{p-1})$ and $\boldsymbol{x'} = (x'_0, x'_1, \ldots, x'_{p-1})$ to the mechanism where $y_i \in \mathit{Range}(\mathcal{M})$ for every $i \in [p-1]$. Let the randomness over $\mathcal{M}$ be a discrete probability distribution. Below, we abuse notation and let $\mathcal{M}(\boldsymbol{x})$ denote the set of outputs obtained by running $\mathcal{M}$ on adaptive inputs $\boldsymbol{x}$. We then have:

$$\begin{aligned}\frac{\mathsf{Pr}[\mathcal{M}(\boldsymbol{x}) = \boldsymbol{y}]}{\mathsf{Pr}[\mathcal{M}(\boldsymbol{x'}) = \boldsymbol{y}]} &= \left(\frac{\mathsf{Pr}[\mathcal{M}(x_0) = y_1]}{\mathsf{Pr}[\mathcal{M}(x'_0) = y_1]}\right) \cdot \prod_{i=1}^{p-1} \frac{\mathsf{Pr}[\mathcal{M}(x_i) = y_i | y_1, \ldots, y_{i-1}]}{\mathsf{Pr}[\mathcal{M}(x'_i) = y_i | y_1, \ldots, y_{i-1}]} \\ &\leq \prod_{i=1}^{p} \exp(\varepsilon) = \exp(p\varepsilon).\end{aligned}$$

The second inequality follows since $\mathcal{M}$ is ε-LDP, each pair of inputs are adjacent, and previous outputs are public information. □

B k-Core Decomposition Approximation Proofs

Proof of Lemma 4.2. This proof is a simple modification of the proof of Lemma 5.12 of [LSY+22]. In this proof, when we refer to the level of a node i, we mean the level of i in $L_{4\log^2 n - 1}$. Furthermore, all expressions are given *in expectation*. For simplicity, we omit the phrase *in expectation* from now on in this proof. Using notation from previous work, let $\hat{k}(i)$ be the core number estimate of i and $k(i)$ be the core number of i. First, we show that

$$\text{if } \hat{k}(i) \leq (2+\lambda)(1+\psi)^{g'}, \text{ then } k(i) \leq (1+\psi)^{g'+1} \tag{31}$$

for any group g'. Let $T(g')$ be the topmost level of group g' and let Z_r be the set of nodes at level r and above. In order for $(2+\lambda)(1+\psi)^{g'}$ to be the estimate of node i's core number, the level of i is bounded by $T(g') \le \text{level}(i) \le T(g'+1) - 1$. Let r be i's level. By a modified Invariant 1 for the expectation setting, if $r < T(g'+1)$, then $|\mathbf{a}_i \cap Z_r| \le (1+\psi)^{\mathcal{F}(r)} \le (1+\psi)^{g'+1}$. Furthermore, each node w at the same or lower level $r' \le r$ has $|\mathbf{a}_w \cap Z_{r'}| \le (1+\psi)^{g'+1}$.

Suppose we perform the following iterative procedure: starting from level $r = 0$, remove all nodes in level r during this turn and set $r \leftarrow r+1$ for the next turn and perform the removal again; we perform this iterative procedure until all nodes are removed. Using this procedure, the nodes in level 0 are removed in the first turn, the nodes in level 1 are removed in the second turn, and so on. Let $d_r(i)$ be the induced degree of any node i that is in the graph before the removal in the r-th turn. Since we showed above that $|\mathbf{a}_i \cap Z_r| \le (1+\psi)^{g'+1}$ for any node i at level $r \le T(g'+1)$, node i on level $r \le T(g'+1)$ after the r-th turn has $d_r(v) \le (1+\psi)^{g'+1}$. Thus, when i is removed on the $(r+1)$-st turn, it has degree $\le (1+\psi)^{g'+1}$. Since all nodes removed before i also had degree $\le (1+\psi)^{g'+1}$ when it was removed, by Lemma 4.6, node i has core number $k(i) \le (1+\psi)^{g'+1}$.

Now we prove our lower bound on $\hat{k}(i)$. For the below proof, let $d_S(i)$ be the induced degree of node i in the induced subgraph consisting of nodes in S. We prove that for any $g' \ge 0$,

$$\text{if } \hat{k}(i) \ge (1+\psi)^{g'}, \text{ then } k(i) \ge \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)} \tag{32}$$

for all nodes $i \in [n]$ in the graph. We assume for contradiction that there exists a node i where $\hat{k}(i) \ge (1+\psi)^{g'}$ and $k(i) < \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$. To consider this case, we use the *pruning* process defined in Lemma 4.6. For a given subgraph S, we *prune* S by repeatedly removing all nodes $i \in S$ whose $d_S(i) < \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$. As in the proof of Lemma 5.12 of [LSY+22], we only consider levels from the same group g'. Let j be the number of levels below level $T(g')$. We prove via induction that the number of nodes pruned from the subgraph induced by $Z_{T(g')-j}$ must be at least

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1}\left((1+\psi)^{g'}\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)\right). \tag{33}$$

We first prove the base case when $j = 1$. In this case, we know that $d_{Z_{T(g')-1}}(i) \ge (1+\psi)^{g'}$ by a modified version of Invariant 2 that holds in expectation. In order to prune i from the graph, we must prune at least

$$(1+\psi)^{g'} - \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)} = (1+\psi)^{g'} \cdot \left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$$

neighbors of i from $Z_{T(g')-1}$. We must prune at least this many neighbors in order to reduce the degree of i to below the cutoff for pruning a node (as we show more formally below).

Then, if fewer than $(1+\psi)^{g'}\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ neighbors of i are pruned from the graph, then i is not pruned from the graph. If i is not pruned from the graph, then i is part of a $\left(\frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}\right)$-core (by Lemma 4.6) and $k(i) \ge \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$, a contradiction. Thus, it must be the case that at least $(1+\psi)^{g'}\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ neighbors of v are pruned in $Z_{T(g')-1}$. For our induction hypothesis, we assume that at least the number of nodes as indicated in Eq. (33) is pruned for j and prove this for $j+1$.

Each node w in levels $T(g') - j$ and above has $d_{Z_{T(g')-j-1}}(w) \ge (1+\psi)^{g'}$ by Invariant 2 (recall that all j levels below $T(g')$ are in group g'). For simplicity of expression, we denote $J \triangleq (1+\psi)^{g'}\left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$. Then, in order to prune the $\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J$ nodes by our induction hypothesis, we must prune at least

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J \cdot \left(\frac{(1+\psi)^{g'}}{2}\right) \tag{34}$$

edges where each such edge is "charged" to the endpoint that gets pruned last. (Note that we actually need to prune at least $(1+\psi)^{g'} \cdot \left(1 - \frac{1}{(2+\lambda)(1+\psi)}\right)$ edges per pruned node as in the base case but $\frac{(1+\psi)^{g'}}{2}$ lower bounds this amount.) Each pruned node prunes less than $\frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$ edges. Thus, using Eq. (34), the number of nodes that must be pruned from $Z_{T(g')-j-1}$ is at least

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j-1} J \cdot \frac{(1+\psi)^{g'}}{2\left(\frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}\right)} = \left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{j} J. \tag{35}$$

Eq. (35) proves our induction step. Using Eq. (33), the number of nodes that must be pruned from $Z_{T(g')-2\log_{(2+\lambda)(1+\psi)/2}(n)}$ is greater than n (when $n > 2$) since $J \geq 1/2$:

$$\left(\frac{(2+\lambda)(1+\psi)}{2}\right)^{2\log_{(2+\lambda)(1+\psi)/2}(n)} \cdot J \geq \frac{n^2}{2}. \tag{36}$$

Thus, at $j = 2\log_{(2+\lambda)(1+\psi)/2}(n)$, we run out of nodes to prune. We have reached a contradiction as we require pruning greater than n nodes in expectation assuming $k(i) < \frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$. This contradicts the fact that more than n nodes is pruned with 0 probability.

From the above, we can first obtain the inequality $k(i) \leq \hat{k}(i)$ from Eq. (31) since this bounds the case when $\hat{k}(i) = (2+\lambda)(1+\psi)^{g'}$; if $\hat{k}(i) < (2+\lambda)(1+\psi)^{g'}$ then the largest possible value for $\hat{k}(i)$ is $(2+\lambda)(1+\psi)^{g'-1}$ by Algorithm 2 and we can obtain the tighter bound of $k(i) \leq (1+\psi)^{g'}$. We can substitute $\hat{k}(i) = (2+\lambda)(1+\psi)^{g'}$ since $(1+\psi)^{g'+1} < (2+\lambda)(1+\psi)^{g'}$ for all $\psi \in (0,1)$ and $\lambda > 0$.

Second, by Eq. (32), for any estimate $(2+\lambda)(1+\psi)^{g}$, the largest g' for which this estimate has $(1+\psi)^{g'}$ as a lower bound is $g' = g + \lfloor \log_{(1+\psi)}(2+\lambda) \rfloor \geq g + \log_{(1+\psi)}(2+\lambda) - 1$. Substituting this g' into $\frac{(1+\psi)^{g'}}{(2+\lambda)(1+\psi)}$ results in

$$\frac{(1+\psi)^{g+\log_{(1+\psi)}(2+\lambda)-1}}{(2+\lambda)(1+\psi)} = \frac{\frac{(2+\lambda)(1+\psi)^{g}}{1+\psi}}{(2+\lambda)(1+\psi)} = \frac{\frac{\hat{k}(v)}{1+\psi}}{(2+\lambda)(1+\psi)}.$$

Thus, we can solve $k(v) \geq \frac{\frac{\hat{k}(v)}{1+\psi}}{(2+\lambda)(1+\psi)}$ and $k(v) \leq \hat{k}(v)$ to obtain

$$k(v) \leq \hat{k}(v) \leq (2+\lambda)(1+\psi)^2 k(v)$$

which is consistent with the definition of a $(2+\eta, 0)$-factor approximation algorithm (in expectation) for core number for any constant $\eta > 0$ and appropriately chosen constants $\lambda, \psi \in (0,1)$ that depend on η. □

C Challenges with Using Previous Techniques

Let us consider a simple algorithm for obtaining the k-core decomposition. The classic algorithm of Matula and Beck [MB83] for static, centralized, sequential k-core decomposition repeatedly *peels* the node with smallest degree; once a node is peeled, all adjacent edges to the node are removed and the process repeats until the graph is empty. The core number of each node is the degree of the node when it is removed from the graph. In fact, many current non-DP algorithms for k-core decompositions use some variant of peeling.

Suppose we attempt to turn this algorithm into a DP algorithm. Let $f(v_i)$ be a function that takes a node and returns the core number of v_i. On edge-adjacent graphs G and G', the sensitivity of f is one, $GS_f = 1$. Furthermore, suppose $F(V) = \sum_{v \in V} f(v)$ is the function that returns the sum of all outputs of f on input $v \in V$. Then, $GS_F = n = |V|$ since *every* node's core number can increase or decrease by 1 as a result of the deletion or addition of a single edge. (Consider a cycle; all nodes in the cycle have core number 2 but removing one edge reduces the core numbers to 1.)

From the above sensitivity analysis, in order to transform the Matula and Beck algorithm into an ε-edge DP algorithm, we must add $\mathsf{Geom}(\varepsilon/GS_1) = \mathsf{Geom}(\varepsilon)$ noise to the core numbers of every node. By the composition theorem, this results in a $n\varepsilon$-edge DP algorithm. Thus, instead of adding $\mathsf{Geom}(\varepsilon)$ noise, we

must instead add $\mathsf{Geom}(\varepsilon/GS_F) = \mathsf{Geom}(\varepsilon/n)$ noise. However, adding this much noise results in an additive error of $\widetilde{O}(n/\varepsilon)$ (hiding $\mathrm{poly}(\log n)$ factors). This is no better than simply guessing the core number! Thus, this simple algorithm fails to provide us with good approximation factors.

One can instead consider other approximate non-DP algorithms (instead of exact algorithms) but it is difficult to bound the GS_F of these algorithms directly. Thus, we must find a function that simultaneously has small sensitivity and for which it is easy to bound the GS of the function. Then, we need to show that the function is not queried too many times. (We do precisely this in our result.)

D Dual of the Densest Subgraph LP

The classic densest subgraph ILP introduced by Charikar [Cha00] associates each node v with a variable $x_v \in \{0,1\}$ where $x_v = 1$ indicates that node v is in the densest subgraph (and $x_v = 0$ otherwise). Each edge is associated with a variable $y_e \in \{0,1\}$ that indicates whether it is part of the densest subgraph. Relaxing the variables to take real values in $0 \le x_v \le 1$ and $0 \le y_e \le 1$ allows us to obtain an LP whose optimal has been shown to equal the density of the densest subgraph. This LP is shown below in Eq. (37).

$$\begin{aligned} \max \sum_{e \in E} y_e & \\ \text{such that} \quad & y_e \le x_u, x_v, \quad \forall e = \{u,v\} \in E \\ & \textstyle\sum_{v \in V} x_v \le 1, \\ & y_e \ge 0, x_v \ge 0 \quad \forall e \in E, \forall v \in V \end{aligned} \tag{37}$$

One can show that the dual of the densest subgraph LP naturally corresponds to the minimum outdegree orientation problem where edges are provided an orientation and the goal is to minimize the maximum outdegree of any node. The LP itself corresponds to this problem in the following way. Each edge $e = \{u,v\}$ has a load of 1 which it wants to assign to its endpoints. The variables $f_e(u)$ and $f_e(v)$ represent these loads. The objective is to minimize the maximum load assigned to any node for any feasible load assignment. The dual of the densest subgraph LP is given below in Eq. (38).

$$\begin{aligned} \min D & \\ \text{such that} \quad & f_e(u) + f_e(v) \ge 1, \quad \forall e = \{u,v\} \in E \\ & \textstyle\sum_{e \ni v} f_e(v) \le D, \\ & f_e(u), f_e(v) \ge 0 \quad \forall e = \{u,v\} \in E \end{aligned} \tag{38}$$

The dual is used in the proof of Lemma 5.7.

References

[AG09] Noga Alon and Shai Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544–556, 2009.

[AHDBV05] J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, 2005.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[ASM+06] Altaf Amin, Yoko Shinbo, Kenji Mihara, Ken Kurokawa, and Shigehiko Kanaya. Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC bioinformatics*, 7:207, 02 2006.

[AU19] Raman Arora and Jalaj Upadhyay. On differentially private graph sparsification and applications. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[BBDS13] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, pages 87–96, 2013.

[BGKV14] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1316–1325, 2014.

[BGM14] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for MapReduce. In *International Workshop on Algorithms and Models for the Web Graph (WAW)*, volume 8882, pages 59–78, 2014.

[BH03] Gary D. Bader and Christopher WV Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, January 2003.

[BHNT15] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *ACM Symposium on Theory of Computing (STOC)*, pages 173–182, 2015.

[BLR13] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, 60(2):1–25, 2013.

[BNO08] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In *Annual International Cryptology Conference*, pages 451–468, 2008.

[BS16] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *International Conference on Theory of Cryptography*, pages 635–658, 2016.

[BS20] Suman K Bera and C Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 457–467, 2020.

[BV18] Victor Balcer and Salil P. Vadhan. Differential privacy on finite computers. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 43:1–43:21, 2018.

[CGB+20] Martino Ciaperoni, Edoardo Galimberti, Francesco Bonchi, Ciro Cattuto, Francesco Gullo, and Alain Barrat. Relevance of temporal cores for epidemic spread in temporal networks. *Scientific Reports*, 10(1):12529, July 2020.

[Cha00] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, 2000.

[CHK+07] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, and Eran Shir. A model of Internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences*, 104(27):11150–11154, 2007.

[CHS09] Andrzej Czygrinow, Michał Hańćkowiak, and Edyta Szymańska. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *Algorithms and Computation*, 2009.

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.

[CQT22] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555, 2022.

[CSS11] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), November 2011.

[CSS12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Optimal lower bound for differentially private multi-party aggregation. In *European Symposium on Algorithms*, pages 277–288, 2012.

[CSS21] T.-H. Hubert Chan, Mauro Sozio, and Bintao Sun. Distributed approximate k-core decomposition and min-max edge orientation: Breaking the diameter barrier. *J. Parallel Distributed Comput.*, 147:87–99, 2021.

[CZ13] Shixi Chen and Shuigeng Zhou. Recursive mechanism: Towards node differential privacy and unrestricted joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 653–664, 2013.

[CZL+20] Deming Chu, Fan Zhang, Xuemin Lin, Wenjie Zhang, Ying Zhang, Yinglong Xia, and Chenyi Zhang. Finding the best k in core decomposition: A time and space optimal solution. In *IEEE 36th International Conference on Data Engineering (ICDE)*, pages 685–696, 2020.

[DBS17] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 293–304, 2017.

[DBS18] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

[DGP09] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2), April 2009.

[DL09] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, page 371–380, 2009.

[DLL16] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, page 265–284, 2006.

[DNPR10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, pages 715–724, 2010.

[DRV10] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*, page 51–60, 2010.

[EKKL20] Marek Eliáš, Michael Kapralov, Janardhan Kulkarni, and Yin Tat Lee. Differentially private release of synthetic graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–578, 2020.

[ELM18] Hossein Esfandiari, Silvio Lattanzi, and Vahab Mirrokni. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1397–1406, 2018.

[ELRS22] Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam Smith. Triangle counting with edge local differential privacy, 2022. Manuscript submitted for publication.

[ELS13] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)*, 18:3–1, 2013.

[ESTW19] Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Kui Wu. Efficient computation of probabilistic core decomposition at web-scale. In *International Conference on Extending Database Technology*, pages 325–336, 2019.

[FHO21] Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost. Differentially private algorithms for graphs under continual observation. In *29th Annual European Symposium on Algorithms*, 2021.

[FHS22] Alireza Farhadi, MohammadTaghi Hajiaghayi, and Elaine Shi. Differentially private densest subgraph. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 151, pages 11581–11597, 2022.

[GBGL20] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. Core decomposition in multilayer networks: Theory, algorithms, and applications. *ACM Trans. Knowl. Discov. Data*, 14(1), January 2020.

[GLM+10] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1106–1125, 2010.

[GLM19] Mohsen Ghaffari, Silvio Lattanzi, and Slobodan Mitrović. Improved parallel algorithms for density-based network clustering. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2201–2210, 2019.

[GMTV14] Christos Giatsidis, Fragkiskos D. Malliaros, Dimitrios M. Thilikos, and Michalis Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 44–50, 2014.

[GPC21] Kasimir Gabert, Ali Pinar, and Ümit V. Catalyürek. A unifying framework to identify dense subgraphs on streams: Graph nuclei to hypergraph cores. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, page 689–697, 2021.

[GRU12] Anupam Gupta, Aaron Roth, and Jonathan Ullman. Iterative constructions and private data release. In *Theory of Cryptography Conference*, pages 339–356, 2012.

[GZ21] Arun Ganesh and Jiazheng Zhao. Privately answering counting queries with generalized gaussian mechanisms. In *2nd Symposium on Foundations of Responsible Computing*, volume 192, pages 1:1–1:18, 2021.

[GZE+21] Yang Guo, Xuekui Zhang, Fatemeh Esfahani, Venkatesh Srinivasan, Alex Thomo, and Li Xing. Multi-stage graph peeling algorithm for probabilistic core decomposition. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, page 532–539, 2021.

[HJMA06] John Healy, Jeannette Janssen, Evangelos Milios, and William Aiello. Characterization of graphs using degree cores. In *International Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 137–148, 2006.

[HL18] Zhiyi Huang and Jinyan Liu. Optimal differentially private algorithms for k-means clustering. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, page 395–408, 2018.

[HLM12] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. *Advances in Neural Information Processing Systems*, 25, 2012.

[HLMJ09] Michael Hay, Chao Li, Gerome Miklau, and David Jensen. Accurate estimation of the degree distribution of private networks. In *Ninth IEEE International Conference on Data Mining*, pages 169–178, 2009.

[HNW20] Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020.

[HR10] Moritz Hardt and Guy N Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70, 2010.

[IMC21] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Locally differentially private analysis of graph statistics. In *30th USENIX Security Symposium*, pages 983–1000, 2021.

[IMC22] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Communication-Efficient triangle counting under local differential privacy. In *31st USENIX Security Symposium*, pages 537–554, 2022.

[JMNR19] Matthew Joseph, Jieming Mao, Seth Neel, and Aaron Roth. The role of interactivity in local differential privacy. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 94–105. IEEE, 2019.

[KBST15] Wissam Khaouid, Marina Barsky, Venkatesh Srinivasan, and Alex Thomo. K-core decomposition of large networks on a single PC. *Proc. VLDB Endow.*, 9(1):13–23, September 2015.

[KGH+10] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernán A. Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 6(11):888–893, November 2010.

[KL10] Daniel Kifer and Bing-Rong Lin. Towards an axiomatization of statistical privacy and utility. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, page 147–158, 2010.

[KLN+11] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[KM17] H. Kabir and K. Madduri. Parallel k-core decomposition on multicore platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1482–1491, 2017.

[KNRS13] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476, 2013.

[KRSY14] Vishesh Karwa, Sofya Raskhodnikova, Adam D. Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *ACM Trans. Database Syst.*, 39(3):22:1–22:33, 2014.

[KS18] Haim Kaplan and Uri Stemmer. Differentially private k-means with constant multiplicative error. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 5436–5446, 2018.

[LM14] Wentian Lu and Gerome Miklau. Exponential random graph estimation under differential privacy. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 921–930, 2014.

[LPA+14] Susan Little, Sergei Pond, Christy Anderson, Jason Young, Joel Wertheim, Sanjay Mehta, Susanne May, and David Smith. Using hiv networks to inform real time prevention interventions. *PloS one*, 9:e98443, 06 2014.

[LRJA10] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. A Survey of Algorithms for Dense Subgraph Discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer US, 2010.

[LSY+22] Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, and Julian Shun. Parallel batch-dynamic algorithms for k-core decomposition and related graph problems. In *34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 191–204, 2022.

[LTZY15] Ying Liu, Ming Tang, Tao Zhou, and Younghae Do. Core-like groups result in invalidation of identifying super-spreader by k-shell decomposition. *Scientific Reports*, 5:9602–9602, May 2015. Publisher: Nature Publishing Group.

[LYC+21] Qi Luo, Dongxiao Yu, Zhipeng Cai, Xuemin Lin, and Xiuzhen Cheng. Hypercore maintenance in dynamic hypergraphs. In *IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2051–2056, 2021.

[LYL+19] Qi Luo, Dongxiao Yu, Feng Li, Zhenhao Dou, Zhipeng Cai, Jiguo Yu, and Xiuzhen Cheng. Distributed core decomposition in probabilistic graphs. In *Computational Data and Social Networks*, pages 16–32, 2019.

[LZHX21] Qing Liu, Xuliang Zhu, Xin Huang, and Jianliang Xu. Local algorithms for distance-generalized core decomposition over large dynamic graphs. *Proc. VLDB Endow.*, 14(9):1531–1543, may 2021.

[LZL+21a] Zhe Lin, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Zhihong Tian. Hierarchical core maintenance on large dynamic graphs. *Proc. VLDB Endow.*, 14(5):757–770, jan 2021.

[LZL+21b] Qingyuan Linghu, Fan Zhang, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Anchored coreness: efficient reinforcement of social networks. *The VLDB Journal*, May 2021.

[LZZ+19] Conggai Li, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. Efficient progressive minimum k-core search. *Proc. VLDB Endow.*, 13(3):362–375, November 2019.

[MB83] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983.

[MB19] Amir Pasha Motamed and Behnam Bahrak. Quantitative analysis of cryptocurrencies transaction graph. *Applied Network Science*, 4, 12 2019.

[MGPV20] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: theory, algorithms and applications. *VLDB J.*, 29(1):61–92, 2020.

[MMSS20] Sourav Medya, Tianyi Ma, Arlei Silva, and Ambuj Singh. A game theoretic approach for k-core minimization. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1922–1924, 2020.

[MPM13] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on Parallel and Distributed Systems*, 24(2):288–300, 2013.

[MPP+15] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 815–824, 2015.

[MRV16] Fragkiskos D. Malliaros, Maria-Evgenia G. Rossi, and Michalis Vazirgiannis. Locating influential nodes in complex networks. *Scientific Reports*, 6(1):19307, January 2016.

[NRS07] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, page 75–84, 2007.

[NSV16] Kobbi Nissim, Uri Stemmer, and Salil Vadhan. Locating a small cluster privately. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, page 413–427, 2016.

[NV21] Dung Nguyen and Anil Vullikanti. Differentially private densest subgraph detection. In *Proceedings of the 38th International Conference on Machine Learning*, pages 8140–8151, 2021.

[OSSW20] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic mis in uniformly sparse graphs. *ACM Transactions on Algorithms (TALG)*, 16(2):1–19, 2020.

[RS16] Sofya Raskhodnikova and Adam D. Smith. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *IEEE 57th Annual Symposium on Foundations of Computer Science*, pages 495–504, 2016.

[SCR+11] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[SCS20] Bintao Sun, T.-H. Hubert Chan, and Mauro Sozio. Fully dynamic approximate k-core decomposition in hypergraphs. *ACM Trans. Knowl. Discov. Data*, 14(4), May 2020.

[Sea16] Adam Sealfon. Shortest paths and distances with differential privacy. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, page 29–41, 2016.

[SGJS+13] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. Streaming algorithms for k-core decomposition. *Proc. VLDB Endow.*, 6(6):433–444, April 2013.

[ST15] Julian Shun and Kanat Tangwongsan. Multicore triangle computations without tuning. In *2015 IEEE 31st International Conference on Data Engineering*, pages 149–160. IEEE, 2015.

[Sta21] Nina Mesing Stausholm. Improved differentially private euclidean distance approximation. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, page 42–56, 2021.

[SU17] Thomas Steinke and Jonathan Ullman. Between pure and approximate differential privacy. *Journal of Privacy and Confidentiality*, 2017.

[SV19] Hsin-Hao Su and Hoa T. Vu. Distributed dense subgraph detection and low outdegree orientation. *CoRR*, abs/1907.12443, 2019.

[SV20] Hsin-Hao Su and Hoa T. Vu. Distributed Dense Subgraph Detection and Low Outdegree Orientation. In *34th International Symposium on Distributed Computing*, pages 15:1–15:18, 2020.

[SW20] Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 181–193, 2020.

[SXK+19] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Hui (Wendy) Wang, and Ting Yu. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, page 703–717, 2019.

[UV11] Jonathan Ullman and Salil Vadhan. Pcps and the hardness of generating private synthetic data. In *Theory of Cryptography Conference*, pages 400–416. Springer, 2011.

[Vad17] Salil Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.

[VAGK21] Friedhelm Victor, Cuneyt G. Akcora, Yulia R. Gel, and Murat Kantarcioglu. Alphacore: Data depth based core decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 1625–1633, 2021.

[VKBK21] Katherine Van Koevering, Austin Benson, and Jon Kleinberg. Random graphs with prescribed k-core sequences: A new null model for network analysis. In *Proceedings of the Web Conference*, page 367–378, 2021.

[WCL+18] Kai Wang, Xin Cao, Xuemin Lin, Wenjie Zhang, and Lu Qin. Efficient computing of radius-bounded k-cores. In *IEEE 34th International Conference on Data Engineering (ICDE)*, pages 233–244, 2018.

[WPF+17] Joel Wertheim, Sergei Pond, Lisa Forgione, Sanjay Mehta, Ben Murrell, Sharmila Shah, David Smith, Konrad Scheffler, and Lucia Torian. Social and genetic networks of hiv-1 transmission in new york city. *PLOS Pathogens*, 13:e1006000, 01 2017.

[WWM+15] Xicheng Wang, Yasong Wu, Lin Mao, Wei Xia, Weiwei Zhang, Lili Dai, Sanjay R. Mehta, Joel O. Wertheim, Xingqi Dong, Tong Zhang, Hao Wu, and Davey M. Smith. Targeting HIV Prevention Based on Molecular Epidemiology Among Deeply Sampled Subnetworks of Men Who Have Sex With Men. *Clinical Infectious Diseases*, 61(9):1462–1468, 06 2015.

[WWW13] Yue Wang, Xintao Wu, and Leting Wu. Differential privacy preserving spectral graph analysis. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 329–340, 2013.

[WZL+22] Tongfeng Weng, Xu Zhou, Kenli Li, Peng Peng, and Keqin Li. Efficient distributed approaches to core maintenance on large dynamic graphs. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):129–143, 2022.

[YHA+20a] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[YHA+20b] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. Towards locally differentially private generic graph metric estimation. In *IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1922–1925, 2020.

[YL15] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, January 2015.

[You95] Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 170–178, 1995.

[ZCP+15] Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Private release of graph statistics using ladder functions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 731–745, 2015.

[ZHH+21] Wei Zhou, Hong Huang, Qiang-Sheng Hua, Dongxiao Yu, Hai Jin, and Xiaoming Fu. Core decomposition and maintenance in weighted graph. *World Wide Web*, 24(2):541–561, 2021.

[ZNF21] Sen Zhang, Weiwei Ni, and Nan Fu. Differentially private graph publishing with degree distribution preservation. *Computers and Security*, 106:102285, 2021.

[ZZC+10] Haohua Zhang, Hai Zhao, Wei Cai, Jie Liu, and Wanlei Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *J. Supercomput.*, 53(2):352–369, 2010.

[ZZQ+17] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. When engagement meets similarity: Efficient (k,r)-core computation on social networks. *Proc. VLDB Endow.*, 10(10):998–1009, June 2017.