2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS) | 978-1-6654-5519-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/FOCS54457.2022.00077

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

Laxman Dhulipala
Department of Computer Science
University of Maryland, College Park
College Park, USA
laxman@umd.edu

Quanquan C. Liu
Department of Computer Science
Northwestern University
Evanston, USA
quanquan@northwestern.edu

Sofya Raskhodnikova
Department of Computer Science
Boston University
Boston, USA
sofya@bu.edu

Jessica Shi
CSAIL
Massachusetts Institute of Technology
Cambridge, USA
jeshi@mit.edu

Julian Shun
CSAIL
Massachusetts Institute of Technology
Cambridge, USA
jshun@mit.edu

Shangdi Yu
CSAIL
Massachusetts Institute of Technology
Cambridge, USA
shangdiy@mit.edu

***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.

I. 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. 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 [18], [46], [49], [52], community detection and computing network centrality measures (where centers tend to have larger core numbers) [22], [35], [54], [71], [78], and network visualization [2], [11], [75], [79]. 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 [51] for a survey).

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 [47], [72], [74] or cryptocurrency payments [55]. 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 [23] is the gold standard for privacy in data analysis. We present results in two differential privacy models: *central* [23] and *local* [43]. 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., [6], [12], [23]).

a) Outline: We provide all definitions and notation for this paper in Section II. We summarize our contributions and provide a technical overview in Section III. In Section IV, we present our locally differentially private k-core decomposition, low out-degree ordering, and densest subgraph algorithms. In Section V, 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 VI, 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.

b) Related Works: DP algorithms have been developed for graph statistics such as subgraph counts [9], [16], [38], [39], [42], [44], [68], [80], degree distribution [20], [34], [60], [81], minimum spanning tree and clustering [37], [41], [57], [58], spectral properties [3], [73], cut problems [3], [25], [30], [45], [62], [64], and parameter estimation [50], [76], [77].

DP Densest Subgraph. The currently best DP algorithms for densest subgraphs are due to Nguyen and Vullikanti [56] and Farhadi et al. [27]. We describe them in detail in Section III.

Non-Private Algorithms for k-Core Decomposition and Related Problems. In the non-DP, fully-dynamic setting, Bhattacharya et al. [8], Henzinger et al. [36], and Sawlani and Wang [61] 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. [67] 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. [13] in the distributed setting. Finally, Liu et al. [48] formulate a parallel, batch-dynamic $(4+\eta)$-approximate k-core decomposition algorithm which we use in our work[1].

II. 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.

A. 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 II.1), an approximate ***densest subgraph*** (Definition II.4), and an approximate ***low out-degree ordering*** (Definition II.3).

Definition II.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*

[1] We only consider one-sided multiplicative error. Thus, we translate the approximation factors of [67] and [48], 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.

*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 $k(v) - \zeta \leq \hat{k}(v) \leq \phi \cdot k(v) + \zeta$.*

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 II.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 II.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 II.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 II.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 $\frac{D^*}{\phi} - \zeta$.*

B. Differential Privacy

We consider two models of differential privacy: central [23] and local [43]. 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 II.6 (Edge-Neighboring [57])**.** *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 II.7 (ε-Edge Differential Privacy [57])**.** *Algorithm $\mathcal{A}(G)$, that takes as input a graph G and outputs some value in range R, is ε-**edge differentially private** (ε-edge DP) if for all $S \subseteq R$ and all edge-neighboring graphs G and G',*

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

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

C. Local Edge Differential Privacy (LEDP)

The LEDP model is an extension of the local differential privacy (LDP) model originally introduced by [43]. Below we use the definitions given in [24] which are based on definitions in [40] for non-graph data. The LEDP was also defined in [38], [39] for the special case of one round, and [39] 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.

Definition II.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,*

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

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 II.9 (LEDP)**.** *A **transcript** π is a vector consisting of 5-tuples $(S^t_U, S^t_R, S^t_\varepsilon, S^t_\delta, S^t_Y)$ – 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". Additional differential privacy tools and definitions can be found in the full version of our paper.

III. 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 [8], [36], [48] 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 [8], [13], [36], [48]. 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.

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 [27], [28].

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 [8], [36], [48]. 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. [13]. 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 IV.

Theorem III.1 (ε-LEDP k-Core Decomposition)**.** *There exists an $O\left(\log^2 n\right)$ 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 [53], matching [19], triangle counting [17], independent set [59], dominating set [1], and many others. Furthermore, many real-world graphs have small degeneracy [7], [21], [26], [63], 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. [30] give an ε-edge DP ordering for the vertex cover problem (where the earliest endpoint of an edge covers that edge).

Theorem III.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 III.3 (ε-LEDP Densest Subgraph)**.** *There exists an $O(\log n)$-round ε-LEDP algorithm that returns a set of nodes that induce a $\left(4+\eta, O\left(\log^3 n/\varepsilon\right)\right)$-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 [8], [36], [48], [67]. 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. [5] together with the modifications made by Su and Vu [66] 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 [56] and the more recent ε-edge DP $\left(2+\eta, O\left(\frac{\log^{2.5}(n)\log(1/\sigma)}{\varepsilon}\right)\right)$-approximation algorithm of Farhadi et al. [27] 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. [15], which obtains a $(1+\eta)$-approximate densest subgraph algorithm. We present our algorithm and its analysis in Section V.

Theorem III.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 VI.

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

IV. 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. All proofs are provided in the full version of our paper.

A. ε-LEDP k-Core Decomposition

We present our algorithm in this section, and provide our analyses in the full version of our paper. Our LEDP algorithm is inspired by the sequential level data structure algorithms of [8], [36] and the parallel level data structure of [48] used to obtain the densest subgraphs and bounded degeneracy orientations of a graph in the dynamic setting. We crucially use the observation made by [48] 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 [48] to $(2+\eta)$ (see Footnote 1) because [48] 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).

a) Non-Private Dynamic Algorithms of [8], [36], [48], [67]: The level data structure of [8], [36] 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 [48] 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.

B. Detailed Algorithm

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 [8], [36], [48], we call this structure a *level data structure*. Our descriptions use terminology from these previous works. However, because we are in the static setting, we are able to simplify the algorithm of [48] 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, $(a_1, \dots, a_n)$, 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

Algorithm 1: LEDP Decomposition and Ordering

```
1  Input: Adjacency lists $(\mathbf{a}_1, \dots, \mathbf{a}_n)$, constant $\eta \in (0,1)$, and
     privacy parameter $\varepsilon \in (0,1)$.
2  Output: $\varepsilon$-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($(\mathbf{a}_1, \dots, \mathbf{a}_n), \varepsilon, \eta$)
4      Set $\psi = 0.5$ and $\lambda = \frac{2}{9}(2\eta - 5)$.
5      Curator initializes $L_0, \dots, L_{4\log^2 n - 1}$ with $L_r[i] \leftarrow 0$
         for every $i \in [n], r \in [0, \dots, 4\log^2 n - 1]$.
6      for $r = 0$ to $4\log^2 n - 1$ 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 $(C, D)$.
```

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 $4\log^2 n - 1$.

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 [48] 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 $0 \leq r \leq 4\log^2 n - 1$ 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. We show

Algorithm 2: Estimate Core Number [48]

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

that by the adaptive composition theorem 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 the full version of our paper.

a) 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.

b) 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.

c) 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 [8] 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. Our algorithm is given in Algorithm 3 and the proof of our approximation bound follows from the proofs of Theorem 2.6 and Corollary 2.7 of Bhattacharya et al. [8] with minor modifications.

First, we describe a few key points of Algorithm 3. 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 $L_{2\log n-1}[i]$ for each $i \in [n]$. 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 using the sum of the released noisy degrees.

Algorithm 3: ε-LEDP Densest Subgraph

```
1 Input: Adjacency lists $(\mathbf{a}_1, \dots, \mathbf{a}_n)$, constants $\psi \in (0,1)$,
    and privacy parameter $\varepsilon \in (0,1)$.
2 Output: A private set of nodes whose induced subgraph is a
    $\left(2+\eta, O\left(\frac{\log^3 n}{\varepsilon}\right)\right)$-approximate densest subgraph in $G$.
3 Function LEDPDensestSubgraph($((\mathbf{a}_1, \dots, \mathbf{a}_n), \varepsilon, \psi)$)
4     Curator initializes
        $L^0_0, L^0_1, \dots, L^0_{2\log n-1}, \dots, L^{2\log n-1}_0, \dots, L^{2\log n-1}_{2\log n-1}$
        with $L^g_r[i] \leftarrow 0$ for every
        $i \in [n], r, g \in [0, \dots, 2\log n - 1]$.
5     for $r = 0$ to $2\log n - 1$ do
6         for $i = 1$ to $n$ do
7             for $g = 0$ to $2\log n - 1$ do
8                 $L^g_{r+1}[i] \leftarrow L^g_r[i]$.
9                 if $L^g_r[i] = r$ then
10                    Let $\mathcal{U}_{i,g}$ be the number of neighbors
                        $j \in \mathbf{a}_i$ where $L^g_r[j] = r$.
11                    Sample $X \leftarrow \mathsf{Geom}(\varepsilon/(8\log^2 n))$.
12                    Compute $\widehat{\mathcal{U}}_{i,g} \leftarrow \mathcal{U}_{i,g} + X$.
13                    $i$ releases $\widehat{\mathcal{U}}_{i,g}$.
14                    if $\widehat{\mathcal{U}}_{i,g} > (1+\psi)^g$ then
15                        $L^g_{r+1}[i] \leftarrow L^g_r[i] + 1$.    ▷ Curator
                            moves $i$ up one level in group $g$.
16        Curator publishes $L^g_{r+1}$ for every
            $g \in [0, \dots, 2\log n - 1]$.
17    Curator uses released noisy degrees of all nodes and
        $L^g_{2\log n-1}$ for all $g \in [0, \dots, 2\log n - 1]$ to peel the
        levels one by one and determine and return the set of
        nodes $S$ whose induced subgraph is an approximate
        densest subgraph.
18    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 $c \geq 1$.
```

V. 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 V.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 [5] along with modifications made by [66] in their distributed algorithm. Although a number of private algorithms [10], [29]–[33], [65], [69], [70] exist for the multiplicative weight updates (MWU) method of Arora et al. [4], 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 [5], [15], [66], 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 II, we define our multiplicative approximation factor as $(1+\eta)$ to be consistent with the other private densest subgraph works [27], [56]. 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 4, calls the subroutine given in Algorithm 5 on various values of z. Algorithm 4 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 5. Algorithm 5 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 4 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 5 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 5. In order to ensure ε-edge DP, our algorithm creates ***dummy*** edges that take a portion of the load. 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 [14]. 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. [5] were the first to apply the MWU framework to densest subgraphs and Su and Vu [66] give an explicit analysis for this algorithm. Although the non-private algorithms of [5], [66] adapt the MWU framework, the analyses of [66] 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 [66] to show the approximation factor of our ε-DP algorithm.

Algorithm 4: 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 5).
8          if $x \neq 0$ then
9              $V_{\max} \leftarrow S$.
10     return $V_{\max}$.
```

VI. 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 [5], [8], [13], [36], [48], [66], described in detail in Sections IV and V. In this section, we describe a general class of algorithms and show that we can transform them to be ε-edge DP (Section VI-B). 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 IV and V. 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.

A. 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

Algorithm 5: Edge DP Densest Subgraph using Density Parameter z

```
1  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 $c_0, c_1, c_2, c > 0$.
2  Output: A pair $(V', z')$ where $V'$ is a set of nodes $V' \subseteq V$ where $G[V']$ has density at least $z' - O\left(\frac{\log^4 n}{\varepsilon}\right)$.
3  Function EdgeDPDensestSubgraphZ($G = (V, E), z, \eta, \varepsilon$)
4      Let $T \leftarrow \frac{c_0 \log n}{\eta^3}$.
5      Initialize load $\ell(e) \leftarrow 0$ for all $e \in E$.
6      if $z = 0$ then
7          return $(V, 0)$.
8      for phase $t = 1$ to $T$ do
9          for each node $v \in V$ do
10             Let $[e_1, e_2, \dots, e_{\deg(v)}]$ 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).
11             Sample $X_v \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T \log_{(1+\eta)} n}\right)$.
12             Initialize each $\hat{\alpha}^t_{e_i v} \leftarrow 0$.
13             Set $\hat{\alpha}^t_{e_i v} \leftarrow 2$ for $i = 1, \dots, [\lceil z/2 \rceil - 1 + X_v]_0^{\deg(v)}$.
14         for each integer $\ell \in [0, 4T]$ do
15             Set $V'_\ell \leftarrow \emptyset$.
16             for each node $v \in V$ do
17                 Sample $Z \sim \mathsf{Geom}\left(\frac{\varepsilon}{6T(4T+1) \log_{(1+\eta)} n}\right)$.
18                 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
19                     $V'_\ell \leftarrow V'_\ell \cup \{v\}$.
20             Sample $Y \sim \mathsf{Geom}\left(\frac{\varepsilon}{3T(4T+1) \log_{(1+\eta)} n}\right)$.
21             if $G[V'_\ell]$ is a non-empty graph with density $\geq z + Y - \frac{c_2 \log^4 n}{\varepsilon}$ then
22                 return $(V'_\ell, z)$
23         for each edge $e = (u, v) \in E$ do
24             Set $\ell(e) \leftarrow \ell(e) + \hat{\alpha}^t_{eu} + \hat{\alpha}^t_{ev}$.
25     return $(V, 0)$.
```

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.

a) 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 $\texttt{update-node-state}_v(I_{v,\tilde{p}}, n_{v,p}, e_{v,p}) \rightarrow I_{v,p}$ 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 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.

b) 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 $\texttt{update-edge-state}_e(I_{e,\tilde{p}}, i_u, i_v) \rightarrow I_{e,p}$ 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.

c) 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 $t_{v,p} = \left|\{w \in N(v) : I_{\{v,w\},p} \text{ satisfies } F\}\right|$, respectively. Then, we define $\texttt{stop}_v$ as follows, for some fixed constants $c_1, c_2 \geq 0$ and fixed threshold $T \geq 0$:

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

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 $T \geq 0$:

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

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

B. 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 II.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:

a) Privacy Framework: Suppose we are provided a locally adjustable algorithm $\mathcal{A}$. Then we formulate the following mechanism $\mathcal{M}(G, \mathcal{I}_{\tilde{p}}, \mathcal{A}, p) \rightarrow (\mathcal{I}_p, \mathcal{O})$ 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 V) 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 that we analyze in the full version of our paper.

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.

References

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

[2] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. 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.

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

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

[5] B. Bahmani, A. Goel, and K. 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.

[6] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *Annual International Cryptology Conference*, pages 451–468, 2008.

[7] S. 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.

[8] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. 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.

[9] J. Blocki, A. Blum, A. Datta, and O. 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.

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

[11] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of Internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences*, 104(27):11150–11154, 2007.

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

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

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

[15] C. Chekuri, K. Quanrud, and M. 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.

[16] S. Chen and S. 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.

[17] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.

[18] M. Ciaperoni, E. Galimberti, F. Bonchi, C. Cattuto, F. Gullo, and A. Barrat. Relevance of temporal cores for epidemic spread in temporal networks. *Scientific Reports*, 10(1):12529, July 2020.

[19] A. Czygrinow, M. Hanćkowiak, and E. Szymańska. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *Algorithms and Computation*, 2009.

[20] W.-Y. Day, N. Li, and M. Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[21] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

[22] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2), Apr. 2009.

[23] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, page 265–284, 2006.

[24] T. Eden, Q. C. Liu, S. Raskhodnikova, and A. Smith. Triangle counting with edge local differential privacy, 2022. Manuscript submitted for publication.

[25] M. Eliáš, M. Kapralov, J. Kulkarni, and Y. T. Lee. Differentially private release of synthetic graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–578, 2020.

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

[27] A. Farhadi, M. T. Hajiaghai, and E. Shi. Differentially private densest subgraph. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 151, pages 11581–11597, 2022.

[28] H. Fichtenberger, M. Henzinger, and W. Ost. Differentially private algorithms for graphs under continual observation. In *29th Annual European Symposium on Algorithms*, 2021.

[29] A. Ganesh and J. 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.

[30] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1106–1125, 2010.

[31] A. Gupta, A. Roth, and J. Ullman. Iterative constructions and private data release. In *Theory of Cryptography Conference*, pages 339–356, 2012.

[32] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. *Advances in Neural Information Processing Systems*, 25, 2012.

[33] M. Hardt and G. 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.

[34] M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. In *Ninth IEEE International Conference on Data Mining*, pages 169–178, 2009.

[35] J. Healy, J. Janssen, E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *International Workshop on Algorithms and Models for the Web-Graph (WAW)*, pages 137–148, 2006.

[36] M. Henzinger, S. Neumann, and A. Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020.

[37] Z. Huang and J. 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.

[38] J. Imola, T. Murakami, and K. Chaudhuri. Locally differentially private analysis of graph statistics. In *30th USENIX Security Symposium*, pages 983–1000, 2021.

[39] J. Imola, T. Murakami, and K. Chaudhuri. Communication-Efficient triangle counting under local differential privacy. In *31st USENIX Security Symposium*, pages 537–554, 2022.

[40] M. Joseph, J. Mao, S. Neel, and A. 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.

[41] H. Kaplan and U. 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.

[42] V. Karwa, S. Raskhodnikova, A. D. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *ACM Trans. Database Syst.*, 39(3):22:1–22:33, 2014.

[43] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[44] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476, 2013.

[45] D. Kifer and B.-R. 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.

[46] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 6(11):888–893, Nov. 2010.

[47] S. Little, S. Pond, C. Anderson, J. Young, J. Wertheim, S. Mehta, S. May, and D. Smith. Using hiv networks to inform real time prevention interventions. *PloS one*, 9:e98443, 06 2014.

[48] Q. C. Liu, J. Shi, S. Yu, L. Dhulipala, and J. 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.

[49] Y. Liu, M. Tang, T. 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.

[50] W. Lu and G. 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.

[51] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis. The core decomposition of networks: theory, algorithms and applications. *VLDB J.*, 29(1):61–92, 2020.

[52] F. D. Malliaros, M.-E. G. Rossi, and M. Vazirgiannis. Locating influential nodes in complex networks. *Scientific Reports*, 6(1):19307, Jan. 2016.

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

[54] M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. C. 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.

[55] A. P. Motamed and B. Bahrak. Quantitative analysis of cryptocurrencies transaction graph. *Applied Network Science*, 4, 12 2019.

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

[57] K. Nissim, S. Raskhodnikova, and A. 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.

[58] K. Nissim, U. Stemmer, and S. 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.

[59] K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully dynamic mis in uniformly sparse graphs. *ACM Transactions on Algorithms (TALG)*, 16(2):1–19, 2020.

[60] S. Raskhodnikova and A. 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.

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

[62] A. 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.

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

[64] N. M. 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.

[65] T. Steinke and J. Ullman. Between pure and approximate differential privacy. *Journal of Privacy and Confidentiality*, 2017.

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

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

[68] H. Sun, X. Xiao, I. Khalil, Y. Yang, Z. Qin, H. W. Wang, and T. 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.

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

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

[71] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *IEEE 34th International Conference on Data Engineering (ICDE)*, pages 233–244, 2018.

[72] X. Wang, Y. Wu, L. Mao, W. Xia, W. Zhang, L. Dai, S. R. Mehta, J. O. Wertheim, X. Dong, T. Zhang, H. Wu, and D. 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.

[73] Y. Wang, X. Wu, and L. Wu. Differential privacy preserving spectral graph analysis. In J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 329–340, 2013.

[74] J. Wertheim, S. Pond, L. Forgione, S. Mehta, B. Murrell, S. Shah, D. Smith, K. Scheffler, and L. Torian. Social and genetic networks of hiv-1 transmission in new york city. *PLOS Pathogens*, 13:e1006000, 01 2017.

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

[76] Q. Ye, H. Hu, M. H. Au, X. Meng, and X. Xiao. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[77] Q. Ye, H. Hu, M. H. Au, X. Meng, and X. Xiao. Towards locally differentially private generic graph metric estimation. In *IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1922–1925, 2020.

[78] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k,r)-core computation on social networks. *Proc. VLDB Endow.*, 10(10):998–1009, June 2017.

[79] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *J. Supercomput.*, 53(2):352–369, 2010.

[80] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. 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.

[81] S. Zhang, W. Ni, and N. Fu. Differentially private graph publishing with degree distribution preservation. *Computers and Security*, 106:102285, 2021.