Differentially Private Densest Subgraph

Alireza Farhadi
Carnegie Mellon University

MohammadTaghi Hajiaghayi
University of Maryland

Elaine Shi
Carnegie Mellon University

Abstract

Given a graph, the densest subgraph problem asks for a set of vertices such that the average degree among these vertices is maximized. Densest subgraph has numerous applications in learning, e.g., community detection in social networks, link spam detection, correlation mining, bioinformatics, and so on. Although there are efficient algorithms that output either exact or approximate solutions to the densest subgraph problem, existing algorithms may violate the privacy of the individuals in the network, e.g., leaking the existence/non-existence of edges.

In this paper, we study the densest subgraph problem in the framework of differential privacy, and we derive upper and lower bounds for this problem. We show that there exists a linear-time \( \epsilon \)-differentially private algorithm that finds a 2-approximation of the densest subgraph with an extra poly-logarithmic additive error. Our algorithm not only reports the approximate density of the densest subgraph, but also reports the vertices that form the dense subgraph.

Our upper bound almost matches the famous 2-approximation by Charikar both in performance and in approximation ratio, but we additionally achieve differential privacy. In comparison with Charikar’s algorithm, our algorithm has an extra poly-logarithmic additive error. We partly justify the additive error with a new lower bound, showing that for any differentially private algorithm that provides a constant-factor approximation, a sub-logarithmic additive error is inherent.

We also practically study our differentially private algorithm on real-world graphs, and we show that in practice the algorithm finds a solution which is very close to the optimal.

1 Introduction

The densest subgraph problem (DSP) [Gol84] is a fundamental tool to many graph mining applications. Given an undirected graph \( G = (V, E) \), the density of an induced subgraph \( S \subseteq V \) is defined as \( d_G(S) = |E(S)|/|S| \), where \( E(S) \) is the set of all edges in the subgraph induced by the vertices \( S \subseteq V \). In the densest subgraph problem, the goal is to find a subset of vertices \( S \subseteq V \) with the highest density \( d_G(S) \). The densest subgraph problem (DSP) is used as a crucial tool for community detection in social network graphs. This problem also has notable applications in learning including link spam detection [GKT05], correlation mining [RPGV20], story identification [ASKS12] and bioinformatics [SHK+10] – we refer the reader to the tutorial by Gionis and Tsourakakis [GT] for more applications of DSP.

Due to its importance, the DSP problem has been studied extensively in the literature [Gol84, Cha00, KS09, SLNT12, SW20, BGP+]: it is long known that efficient, polynomial-time algorithms exist for finding the exact solution of DSP [Gol84, Cha00, KS09].

In many applications of DSP, however, the underlying graph is privacy sensitive (e.g., social network graphs). Therefore, one might be concerned that the result output by the DSP algorithm might breach the privacy of the individuals in the network, e.g., disclose the (non)-existence of friendship between pairs of individuals. In this paper, we explore how to perform community detection on sensitive graphs, while protecting individuals’ privacy. To this end, we ask the following question:

Can we construct a differentially private algorithm that computes a good approximation of the densest subgraph of a given graph \( G \)?

We first help the reader recall the notion of differential privacy [DMNS06] in a graph context. Let \( G \) and \( G' \) be
two graphs that are identical expect the existence/non-existence of a single edge. Informally, the differential privacy requires that the outputs of the (randomized) algorithm on $G$ and $G'$ are close in distribution. In this way, the output of the algorithm does not reveal meaningful information about the existence of an edge in the graph. Henceforth, we say that two undirected graphs $G$ and $G'$ are neighboring if they differ in only one edge. More formally, differential privacy (DP) is defined as follows [DMNS06].

**Definition 1.1 (($(\epsilon, \delta)$-Differential Privacy (DP))).** Let $\epsilon > 0$ and $\delta \in [0,1]$. We say that a randomized algorithm $\text{Alg}$ achieves $(\epsilon, \delta)$-differential privacy or $(\epsilon, \delta)$-DP for short, if for any two neighboring graphs $G$ and $G'$, for any subset $U$ of the output space,

$$\Pr[\text{Alg}(G) \in U] \leq e^\epsilon \cdot \Pr[\text{Alg}(G') \in U] + \delta$$

Whenever $\delta = 0$, we also say that the algorithm satisfies $\epsilon$-DP.

For the densest subgraph problem, we assume that the output contains 1) a dense subset of vertices $S \subseteq V$; and 2) an estimate of the density of $S$. Had we required the algorithm to output only an estimate of $\max_{S \subseteq V} d_G(S)$, i.e., an estimated density of the densest subgraph, then it would have been easy to devise a DP algorithm: observe that the quantity $\max_{S \subseteq V} d_G(S)$ has small global sensitivity, that is, if we flip the existence of a single edge in $G$, the quantity $\max_{S \subseteq V} d_G(S)$ changes by at most 1. As a result, we can just use the standard Laplacian mechanism [DMNS06] to output a DP estimate with good accuracy. However, we stress that most interesting applications would also want to know the dense community $S$ — simply knowing an estimate of its density would not be too useful.

The requirement to also report a dense vertex set $S \subseteq V$ makes it much more challenging to devise a DP algorithm. In our case, none of the off-the-shelf DP mechanisms would directly work to the best of our knowledge. First, observe that the output is high-dimensional, and has high global sensitivity as we explain in Appendix A. Therefore, the standard Laplacian mechanism [DMNS06, Vah17, DR14] (also called output perturbation) completely fails. Another naïve approach is randomized response [War65, Vah17, DR14] (also called input perturbation), i.e., adding some noise to obfuscate the existence of each edge. Unfortunately, as we argue in Appendix A, the randomized response approach gives poor utility. Finally, the exponential mechanism [MT07, Vah17, DR14] also fails — not only is it not polynomial-time, the standard analysis gives an error bound as large as $O(n)$ which makes the result meaningless.

### 1.1 Our Results and Contributions

We present new upper- and lower-bounds for the differentially private, densest subgraph problem. First, we give a linear-time DP approximation algorithm for the densest subgraph problem. The runtime and accuracy of our algorithm are roughly competitive to the state-of-the-art non-private approximation algorithm by Charikar [Cha00]. Specifically, let $n$ denote the number of vertices. Our algorithm is linear-time, and achieves $\epsilon$-DP and $(2, O(\frac{1}{\epsilon} \log^{2.5} n))$-approximation — here we have two approximation parameters: the first parameter $2$ is the multiplicative approximation ratio, and the second parameter $O(\frac{1}{\epsilon} \log^{2.5} n)$ is an additive error. In comparison, Charikar’s famous (non-private) linear-time algorithm achieves $(2, 0)$-approximation where the additive error is 0. We justify the extra additive error with a new lower bound, showing that to achieve any constant-multiplicative approximation, some sub-logarithmic additive error is unavoidable. Our upper- and lower-bound results are stated in the following theorems:

**Theorem 1.2 (DP approximation of densest subgraph).** Given a graph $G$, and parameters $\epsilon > 0$ and $\sigma \in [0,1]$, there exists a linear-time $\epsilon$-DP algorithm that succeeds with the probability of $1 - \sigma$ and outputs $S \subseteq V$ and an estimate $d^*$ such that

$$\text{OPT}/2 - O(\frac{1}{\epsilon} \log^{2.5} n \cdot \log \frac{1}{\sigma}) \leq d_G(S) \leq \text{OPT},$$

and $|d^* - d_G(S)| \leq O(\frac{1}{\epsilon} \log^{2.5} n \cdot \log \frac{1}{\sigma})$

where $\text{OPT}$ is the true density of the densest subgraph.

**Theorem 1.3 (Lower bound on additive error for DP densest subgraph).** Let $\alpha > 1$, $\epsilon > 0$ be arbitrary constants, $\exp(-n^{0.49}) < \sigma < 0.000001 \cdot \min(1, \epsilon, \exp(-\epsilon))$, and $0 \leq \delta \leq \frac{\sigma e}{\log \frac{1}{\sigma}}$. Then, there exists a sufficiently small $\beta = \Theta(\frac{1}{\alpha \sqrt{\frac{1}{\epsilon} \log \frac{1}{\sigma}}})$ such that there does not exist an $(\alpha, \beta)$-approximation with $1 - \sigma$ probability.

Note that our upper bound achieves $\epsilon$-DP, and our lower bound works even for $(\epsilon, \delta)$-DP. This makes both our upper- and lower-bounds stronger. The proof of Theorem 1.3 is available in Appendix E. Finally, we conclude the paper in Section 5 by demonstrating the performance of our algorithm on real-world datasets. We show that in practice, our algorithm achieves a very accurate solution, even for small choices of the privacy parameter $\epsilon$. 
1.2 Technical Highlight

To see the intuition behind our final algorithm, it helps to break it down into several intermediate steps, to see how the various techniques are eventually woven together.

Background on Charikar’s famous algorithm. Our algorithm is inspired by a work of Charikar [Cha00]. Charikar [Cha00] shows that a simple greedy algorithm can achieve a multiplicative approximation ratio of 2 for DSP. The greedy algorithm is as follows. Let $G = (V, E)$ be an undirected graph. Initially, let $S$ be the set of all vertices. At each iteration, the algorithm finds a vertex $v_{\text{min}} \in S$ with the minimum degree in the graph induced by the vertices of $S$, and removes $v_{\text{min}}$ from $S$. Consider an algorithm that repeats the aforementioned procedure until the set $S$ becomes empty. From all of the sets $S$ encountered during the execution of the algorithm, the algorithm returns the one with the highest density. Charikar proved that this simple greedy algorithm achieves an approximation ratio of 2.

Warmup idea: a quadratic-time DP algorithm. Our first idea is as follows. In Charikar’s algorithm, in each iteration, all residual vertices $v \in S$ examine their degree within the subgraph induced by $S$ — henceforth we call the the degree of $v$ in the subgraph induced by $S$ the residual degree of $v$. Charikar’s algorithm picks the $v$ with the minimum residual degree and removes it from $S$. Our idea is to replace the residual degree with a noisy, DP counterpart. Unfortunately, naïvely adding independent noise to the true residual degree of each $v$ in the previous step would result in an $n$-fold loss in error given a fixed privacy budget $\epsilon$ (however the loss can be reduced to $\sqrt{n}$ if we allowed $(\epsilon, \delta)$-DP rather than $\epsilon$-DP and used the advanced composition theorem [DRV10, Vah17, DR14]).

Our idea is to rely on the elegant DP prefix sum mechanism by Dwork et al. [DNPR10] and Chan et al. [CSS10, CSS11]. Specifically, we can think of the problem as follows.

- Initially, every vertex computes its noisy total degree using the standard Laplacian mechanism. Although there are $n$ vertices, we only need to add noise of constant average magnitude by using parallel composition.
- Next, every vertex $v$ still remaining in $S$ maintains a noisy counter to keep track of roughly how many of its direct neighbors have departed (i.e., have been removed from $S$). If we subtract this value from the vertex’s noisy total degree, we get an estimate of its residual degree in the subgraph induced by $S$.

Therefore, the problem boils down to how to have every residual vertex $v$ maintain a noisy counter of how many of its neighbors have departed. Imagine that every time a neighbor of $v$ departs, a value of 1 is accumulated to $v$’s counter; and every time a non-neighbor of $u$ departs, 0 is accumulated to $v$’s counter. Using the elegant DP prefix sum mechanism by Dwork et al. [DNPR10] and Chan et al. [CSS10, CSS11], we can report $v$’s noisy counter value at any time step, incurring only $O(\frac{1}{\epsilon} \cdot \text{poly} \log n)$ error with all but negligible probability. The noisy counter values and the vertices’ noisy degrees are then used to determine which vertex is to depart next. Further, although it seems like there are $n$ counters, using parallel composition, we need not incur extra loss in the privacy budget due to the $n$ counters.

By extending Charikar’s proof (which we omit in this short roadmap), we can prove that this warmup algorithm achieves the desired $2(1 + \epsilon \cdot \text{poly} \log n)$-approximation. In particular, the error of the prefix sum mechanism directly contributes to the additive error term. Unfortunately, the warmup algorithm incurs $\Theta(n^2)$ runtime, since we need to update $O(n)$ noisy counters in each of the $n$ iterations.

Making it quasilinear time. Our final goal is to get an $O(m + n)$-time algorithm where $m$ denotes the number of edges and $n$ denotes the number of vertices. However, as an important stepping stone, let us first consider how to make it quasilinear time in $m + n$. The key observation is that when the graph is sparse, updates to the vertices’ noisy counters (realized by the prefix sum mechanisms) are also sparse. Most of the $n^2$ updates come with an input 0, and only $m$ of them come with an input of 1. Our idea is therefore to avoid triggering updates when the input is 0.

While the intuition seems simple, realizing this idea differentially privately is actually tricky since we need to avoid consuming too much privacy budget. At a very high level, each vertex $v \in S$ keeps track of a noisy outstanding counter $\text{Cnt}(v)$ of the number of its neighbors that departed recently, but have not been accumulated into the prefix sum mechanism yet. When a vertex $u$ gets removed from $S$, it informs its neighboring vertices to update their noisy outstanding counters. At this moment, each vertex $v$ also checks if its noisy outstanding counter $\text{Cnt}(v)$ has exceeded some predetermined polylogarithmic noisy threshold — if so, it accumulates the current outstanding counter into its prefix sum mechanism, and resets $\text{Cnt}(v)$ to 0.
The key technical challenge here is that we would be invoking with high probability the total of $O(m)$ updates to the vertices’ noisy outstanding counters, but we cannot afford an $O(m)$-fold loss in the privacy budget (or equivalently, an $O(m)$-fold loss in error when the privacy budget is fixed). To resolve this problem, our idea is in spirit reminiscent of the sparse-vector technique [DNR+09, RR10, HR10]. We show that we can reduce the privacy analysis of our algorithm to the standard sparse-vector technique. See the subsequent “proof techniques” paragraph regarding the technical challenges in the analysis and proof.

Final touches: making it linear time. The above algorithm can be implemented in $O(m + n)\log n$ time if we use a Fibonacci heap to store the residual vertices based on their residual degree (i.e., degree in the graph induced by $S$). To make the algorithm linear time, we discretize vertices residual degree into polylogarithmically sized regions, and place each vertex in a corresponding bucket based on its residual degree.

Using an idea inspired by Charikar [Cha00], one can show that if a vertex is removed from the $k$-th bucket in the current iteration, then, in the next iteration, we only need to sequentially look at the $(k - 1)$-th, $k$-th, and $(k + 1)$-th buckets. Moreover, within each bucket, all vertices are treated as having roughly the same degree, and we do not further differentiate them in picking the next vertex to remove from $S$. Using appropriate data structures to store the buckets and the vertices within the buckets, we can eventually obtain a DP-algorithm that completes in $O(m + n)$ runtime. Here, the discretization due to bucketing introduces some additive polylogarithmic error, but asymptotically we still preserve the $(2, O(\frac{1}{\epsilon} \log^2 n))$-approximation as before. We defer the detailed algorithm and analysis to Section 4.

Proof techniques. Proving our algorithm DP turns out to be rather non-trivial. Specifically, our algorithm is not a simple sequential composition of the various underlying building blocks (e.g., prefix sum mechanism, and outstanding counter threshold queries). Therefore, we cannot simply analyze each building block separately and then use standard composition theorems to get the desired DP guarantees. The problem is that the building blocks are interleaved in an adaptive way: the outcome of one step of the prefix sum mechanism corresponding to some vertex will affect the input to the next step of some outstanding counter query, which will then affect the input to the next step of prefix sum mechanism. Despite the complex and adaptive nature of our algorithm, we show that the privacy analysis of our algorithm can be reduced the privacy bounds of sparse-vector-technique.

The actual proof is involved and we defer the detailed exposition to Appendix C.1, Appendix C.2, and Appendix D. All the missing proofs are available in the appendices with the same Theorem number.

1.3 Additional Related Work

Differentially private algorithms for graphs. Early works on differentially private graph algorithms focused on computing simple statistics from graphs. The elegant work by Nissim et al. [NRS07] was the first to apply the DP notion to graph computations. Specifically, they showed how to release the cost of minimum spanning tree and the number of triangles in a graph. The work by Hay et al. [HLMJ09] considered how to release degree distribution while preserving DP. Other works consider how to release the answers to all queries belonging to some class on a given graph. For example, Gupta, Roth, and Ullman [GRU12] consider how to compute a private synthetic data structure for answering all cut queries with $O(n^{1.5})$ error where $n$ denotes the number of vertices. Gupta, Hardt, Roth, and Ullman show how to release the cut function on arbitrary graphs [GHRU11].

Closely related work. Our DSP problem can be viewed as a combinatorial optimization problem. To the best of our knowledge, there exist few works that consider how to solve combinatorial optimization problems differentially privately in graphs. The first such work was the elegant work by Gupta et al. [GLM+10]. They showed polynomial-time DP algorithms for approximating the min-cut and vertex cover problem. For min-cut, their solution can report the vertices on both sides of the cut. However, for vertex cover, their algorithm cannot report the exact set of vertices in the vertex cover — instead, it outputs a permutation of vertices, and if one knows the set of edges, one can recover a good vertex cover from this permutation.

Other notions of privacy. In this paper, we consider the notion of edge differential privacy in graphs, which was a standard notion adopted in various prior works [GLM+10, GHRU11, GRU12, NRS07, KRSY14, HLMJ09]. Some works study stronger notions. For example, Kasiviswanathan et al. [KNRS13] and Blocki et al. [BBDS13] investigate the notion of node differential privacy, where neighboring graphs are defined as two graphs that differ in one node rather than one edge. Gehrie et al. [GLP11] explore a different strengthening of differential privacy for social network graphs. An interesting future work direction is to understand whether we can design accurate DSP algorithms that
satisfy these strengthened notions of privacy.

**Concurrent work of Nguyen and Vullikanti [NV21].** In the independent and concurrent work, Nguyen and Vullikanti designed a $(\epsilon, \delta)$-DP algorithm for the densest subgraph. Their $(\epsilon, \delta)$-DP algorithm is less secure than our $\epsilon$-DP algorithm since for our algorithm we have $\delta = 0$. We also show in section 5 that in practice, our algorithm achieves a significantly more accurate solution in comparison to Nguyen and Vullikanti. It is also worth mentioning that while our algorithm runs in a linear-time, it is not clear whether the algorithm proposed by Nguyen and Vullikanti can be realized in a linear work. In fact, the algorithm SEQDENSEDP introduced in their paper [NV21] has a quadratic running time. Nguyen and Vullikanti also provide a similar sub-logarithmic lower bound on the additive error of any differentially private algorithm. They also consider a PRAM version of their algorithm, while our algorithm is in the RAM model.

2 Preliminaries

2.1 Densest Subgraph

We define the densest subgraph problem [Gol84, Cha00]. Let $G = (V, E)$ be an undirected graph and $S \subseteq V$. We define $E(S)$ to be the edges induced by $S$, i.e., $E(S) := \{(i, j) \in E : i, j \in S\}$.

**Definition 2.1.** Let $S \subseteq V$. We define the density $d(S)$ of the subset $S$ to be $d(S) := \frac{|E(S)|}{|S|^2}$. We define the density $d(G)$ of the undirected graph $G(V, E)$ to be $d(G) := \max_{S \subseteq V} d(S)$.

Observe that $2d(S)$ is simply the average degree of the subgraph induced by $S$.

We next define the notion of approximation we use to measure the algorithm’s utility.

**Definition 2.2 (Approximation algorithm for densest subgraph).** Given an undirected graph $G = (V, E)$, we want to design a randomized algorithm $\text{Alg}$ that outputs 1) a subset of vertices $S^* \subseteq V$ which is an estimate of the densest subgraph; and 2) a noisy density $d^*$, which is an estimate of $d(G)$. Let $\alpha \geq 1$, $\beta > 0$, and $\sigma \in (0, 1]$. Such an algorithm $\text{Alg}$ is said to achieve $(\alpha, \beta)$-approximation with $1 - \sigma$ probability, iff with $1 - \sigma$ probability, the following hold:

1. $d(S^*) \geq d(G)/\alpha - \beta$, i.e., the algorithm outputs a dense set $S^*$ whose (true) density is close to $d(G)$; and
2. $|d^* - d(S^*)| \leq \beta$, i.e., the estimated density $d^*$ is close to the true density of the reported subgraph $S^*$.

As mentioned in Section 1, we consider edge differential privacy in this paper. We say that two undirected graphs $G$ and $G'$ are neighboring if the adjacency matrix of $G$ and $G'$ differ in only one entry (i.e., $G$ and $G'$ are the same except for the existence/non-existence of a single edge). The notion of $(\epsilon, \delta)$-differential privacy and $\epsilon$-differential privacy were formally defined in Section 1.

2.2 Mathematical Tools

We define the symmetric geometric distribution [BV18, SCR11] which can be viewed as a discrete version of the standard Laplacian distribution [DMNSS06].

**Definition 2.3 (Symmetric geometric distribution).** Let $\gamma > 1$. The symmetric geometric distribution $\text{Geom}(\gamma)$ takes integer values such that the probability mass function at $k$ is $\frac{\gamma^{-k}}{\sum_{k'\geq1}\gamma^{-k'}}$.

We shall assume that sampling from the symmetric geometric distribution takes constant time. How to sample such noises was discussed in detail in earlier works on differential privacy [BV18].

The global sensitivity of a function $f(I)$, denoted $\Delta_f$, is defined as follows:

$$\Delta_f := \max_{\text{1, I’neighboring}} |f(I) - f(I’)|_1$$

The following fact about the geometric mechanism (which is equivalent to a discrete version of the Laplacian mechanism) was shown in previous works [DMNSS06, DNPR10, CSS10, CSS11, BV18].

**Fact 1 (Geometric mechanism).** The geometric mechanism $f'(I) := f(I) + \text{Geom}(\exp(\epsilon/\Delta_f))$ satisfies $\epsilon$-DP. Moreover, for $\sigma \in (0, 1)$, for any input $I$, the error $|f'(I) - f(I)|$ is upper bounded by $O(\frac{\Delta_f}{\epsilon} \cdot \log \frac{1}{\sigma})$ with probability $1 - \sigma$.

2.3 Building Block: Differentially Private Prefix Sum Mechanism

Dwork et al. [DNPR10] as well as Chan, Shi, and Song [CSS10, CSS11] suggest a DP prefix sum mechanism. Initially, the mechanism is initialized with $N$, which is an upper bound on the total number of values that will arrive, and at the time of initialization, the mechanism’s output is defined to be 0. Next, a sequence of at most $N$ integer values arrive one by one, and the value that arrives at time $t \in [N]$ is denoted $x_t$. In every time step $t$, the mechanism outputs an estimate $\text{PSum}_t$ of the prefix sum $\sum_{t’\leq t} x_{t’}$. The term $\text{err}_t := |\text{PSum}_t - \sum_{t’\leq t} x_{t’}|$ measures the error of the estimate at time $t$. We refer the reader to Appendix B
updated. Equivalently, we can change the sampling as
the respective time step — and if so, $PSum$

(CSS10, CSS11).]

Theorem 2.4 The earlier works prove the following theorem about
experiments.

Table 2: Running time of DP algorithms. All running
times are reported in seconds.

for more information about the prefix sum mechanism.
The earlier works prove the following theorem about
such a DP prefix-sum mechanism:

for all $t \in [N]$ and $\sigma \in (0, 1)$, with probability
$1 - \sigma$, $err_t < O(\frac{1}{\epsilon} \cdot \log^{1.5} N \cdot \log \frac{1}{\sigma})$.

2. Making $n$ updates to the prefix-sum mechanism takes total time $O(n)$, that is, the average time per update is $O(1)$.

3 A Quasilinear-Time Scheme

3.1 Detailed Construction

We first describe an algorithm that runs in time quasi-
linear in $m + n$ where $m$ denotes the number of edges
and $n$ denotes the number of vertices. The intuition
behind our algorithm has been explained in Section
1.2. Later in Section 4, we describe how to
improve the algorithm’s runtime to $O(m + n)$.

Running it in quasilinear time. In the above al-
gorithm, if we run Line (4e) naïvely as is, it will require
$\Omega(n^2)$ time. However, with an additional data
structure trick, we can run the above algorithm in time
$O(m + n \log n)$. Recall that every time a vertex $u$’s
outstanding counter $Cnt(u)$ is reset to 0, we also sam-
ples a fresh $\mathcal{E}(u)$, and subsequently until $Cnt(u)$ is
reset to 0 again, in every time step, we will check if
$Cnt(u) + \mathcal{E}(u) + \mathcal{N} > T$ where $\mathcal{N}$ is freshly sampled in
the respective time step — and if so, $PSum(u)$ must be
updated. Equivalently, we can change the sampling as
follows: any time $Cnt(u)$ gets updated (i.e., either re-
set to 0 or incremented), we sample a random variable
$\tau(u)$, which means the following: if $Cnt(u)$ does not get
updated, when is the next time step in which the event
$Cnt(u) + \mathcal{E}(u) + \mathcal{N} > T$ happens. Note that if $\tau(u) > n$,
we can simply treat $\tau(u) = \infty$. We may assume that
$\tau(u)$ can be sampled in constant time, since it follows
a geometric distribution (see earlier works [BV18] on

Remark. Below is the meta-algorithm.

immediately after the meta-algorithm description, we describe additional data
structure tricks to run it in quasilinear
time.

Data : Let $G := (V, E)$ be the input graph.

Let $\epsilon_0 = \epsilon_1 = \epsilon_2 = \epsilon' = \epsilon/4$. Let
$T := \frac{\epsilon}{\epsilon'} \log n \log \frac{1}{\sigma}$ for a suitably large
constant $C$.

1: Every vertex $v \in V$ computes its noisy degree
$\deg(v) = \deg(v) + \mathcal{G}(\epsilon^{o}/2)$ where $\deg(v)$
denotes $v$’s true degree.

2: Every vertex $v \in V$ initializes an $\epsilon_1$-DP
prefix-sum algorithm. Henceforth we use
$PSum(v)$ denote the DP-prefix-sum instance for
the vertex $v$; moreover, the notation $PSum(v)$
also denotes the current outcome of the algorithm
$PSum(v)$.

3: Every vertex $v \in V$ initializes a counter, denoted
$Cnt(v) := 0$. Additionally, initialize a fresh noise
$\mathcal{E}(u) := \mathcal{G}(\epsilon^{o})$

4: Let $S := V$ and $d_{\max} := 0$. Repeat the following
until $S$ is empty:

(a) Find the vertex $v \in S$ whose $D(v) - PSum(v)$
is the smallest.

(b) If $d_{\max} < D(v) - PSum(v)$, then update
$d_{\max} := D(v) - PSum(v)$ and let $S^* := S$.

(c) Remove $v$ from $S$.

(d) For each $u \in S$ such that $(u, v) \in E$: let
$Cnt(u) := Cnt(u) + 1$.

(e) For each $u \in S$:

- Let $\mathcal{N} := \mathcal{G}(\epsilon^{o})$ be a fresh noise.

- If $Cnt(u) + \mathcal{E}(u) + \mathcal{N} > T$: input $Cnt(u)$ to
$PSum(u)$, reset $Cnt(u) := 0$ and initialize a
fresh noise $\mathcal{E}(u) := \mathcal{G}(\epsilon^{o})$.

5: return $S^*$ and
d$^* := \min\left(\frac{|\mathcal{E}(S^*)| + |\mathcal{G}(\exp(\epsilon'))|}{|S^*|}, |S^*|\right)$.

Table 1: General statistics of 6 networks used in our
experiments.

Table 2: Running time of DP algorithms. All running
times are reported in seconds.

- musae [RA15] 9806 133138 39.037 Social net. of Twitch (DE)
- musae [RA15] 7136 39124 13.3955 Social net. of Twitch (DE)
- socfb-Amherst41 [RA15] 22755 90954 54.5489 Social net. of Facebook (Amherst)
- socfb-Auburn71 [RA15] 14448 973818 84.5551 Social net. of Facebook (Auburn)
how to do this). Therefore, we can maintain a table $L[1..n]$ where $L[t']$ stores a linked list of vertices that want their $\text{PSum}$ updated in time step $t'$. If vertex $u$’s $\tau(u)$ value changes from $t_1$ to $t_2$ before time step $t_1$, then we remove $u$ from $L[t_1]$ and add $u$ to $L[t_2]$ — this can be accomplished in constant time if in the table $\tau(u)$ that stores the next $\text{PSum}$ update time for $u$, we also store a pointer to $u$’s position in the table $L$. Instead of executing Line 4(e) in a brute-force way, we can simply read $L[t]$ in each time step $t$ to look for the vertices that want their $\text{PSum}$ updated in time step $t$.

Moreover, we will also use a Fibonacci heap to maintain the residual noisy degree (i.e., $D(v) - \text{PSum}(v)$) of every vertex. Due to the way $T$ is chosen, for a proper constant $C$, one can show through a standard Chernoff bound that with the probability of $1 - \sigma$, there cannot be more than $3m \text{PSum}$ updates in which the true increment input to the $\text{PSum}$ instance is 0. This means that there cannot be more than $O(m) \text{PSum}$ updates in total except with $\sigma$ probability. Summarizing the above, the above algorithm can be executed in time $O(m + n \log n) \log(1 - \sigma)$.

3.2 Differential Privacy and Utility Guarantees

We now formally state the algorithm’s differential privacy guarantees as well as utility.

Theorem 3.1 (Differential privacy of the output). The algorithm in Section 3.1 satisfies $(\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon')$-DP.

Our algorithm’s utility is stated in the following theorem:

Theorem 3.2 (Utility of our algorithm). Our algorithm in Section 3.1 achieves $(2, O(\frac{1}{\epsilon} \cdot \log^2 n \cdot \log \frac{1}{\sigma}))$-approximation with probability $1 - \sigma$.

4 A Linear-Time Algorithm

As mentioned, the algorithm in Section 3.1 takes $O(m + n \log n)$ time. We suggest an improved version that runs in linear time, inspired by a trick suggested by Charikar [Cha00].

Observe that $|\min_{v \in S_{t+1}} |E(v, S_{t+1})| - \min_{v \in S_1} |E(v, S_1)|| \leq 1$. Our algorithm is maintaining a noisy estimate $D(v) - \text{PSum}(v)$ of $|E(v, S_1)|$ for all residual vertices. As we discussed, all estimates have at most $O(\frac{1}{\epsilon} \log^2 n \cdot \frac{1}{\sigma})$ error with $1 - \sigma$ probability. Henceforth let $C$ be a sufficiently large constant, and let $\text{err} := \frac{C}{\epsilon} \log^2 n \cdot \frac{1}{\sigma}$. We can discretize the value of $D(v) - \text{PSum}(v)$ into $B := \lceil \frac{n}{\text{err}} + 1 \rceil$ buckets. The $i$-th bucket will contain values from the range $[(i - 1) \cdot \text{err} - \text{err}/2, (i - 1) \cdot \text{err} + \text{err}/2]$. We can now maintain a data structure such that each vertex is placed in the right bucket depending on the current estimate $D(v) - \text{PSum}(v)$. Inside each bucket, we maintain a linked list of vertices. We also maintain an array such that each vertex stores the bucket it currently belongs to, as well as its pointer in the corresponding linked list.

With this idea, we can modify the algorithm in Section 3.1 into a variant that runs in linear time.

Theorem 4.1 (Linear-time variant). The above algorithm satisfies $\epsilon$-DP and moreover, it achieves $(2, O(\frac{1}{\epsilon} \cdot \log^2 n \cdot \frac{1}{\sigma}))$-approximation with probability $1 - \sigma$. The algorithm completes in time $O(n + m)$ with $1 - \exp(-\Omega(m))$ probability where $n = |V|$ and $m = |E|$.

5 Empirical Results

In this section, we analyze our differential private algorithm experimentally on real-world datasets and compare it to Charikar’s algorithm and other DP algorithms for the densest subgraph problem. In our experiments, We use Charikar’s algorithm as a non-differential private baseline. Table 1 shows the 6 dif-

![Figure 1: Accuracy of our linear-time algorithm as well as the accuracy of the Nguyen and Vullikanti [NV21] on different networks. The orange line shows the accuracy of our proposed algorithm, and the blue line shows the accuracy of Nguyen and Vullikanti.](image)
Algorithm 2: Differentially Private Densest Subgraph - Linear-Time Variant.

**Data**: Let $G := (V, E)$ be the input graph. Let $\epsilon_0 = \epsilon_1 = \epsilon_2 = \epsilon' = \epsilon/4$. Let $T := \frac{C}{\epsilon} \log n \log \frac{1}{\delta}$ for a suitably large constant $C$.

**Initialization**: Run the initialization steps of Algorithm 1 in Section 3.1, and moreover, place all vertices in the right bucket. Recall that each vertex stores a pointer to where it resides in the linked list of its bucket. Initially, let $idx = 1$. We may assume that the (imaginary) 0-th bucket is always empty.

1: Let $S := V$ and $d_{\text{max}} := 0$. Repeat the following until $S$ is empty:

(a) Let $idx'$ be the bucket from which a vertex is removed in the previous time step. Check whether each bucket $idx' - 1, idx', \ldots$ is non-empty, and let $idx$ be the smallest non-empty bucket among these. Let $v$ be an arbitrary vertex from this bucket $idx$. Remove $v$ from the bucket.

(b) If $d_{\text{max}} < D(v) - P\text{Sum}(v)$, then update $d_{\text{max}} := D(v) - P\text{Sum}(v)$ and let $S^* := S$.

(c) Remove $v$ from $S$.

(d) For each $u \in S$ such that $(u, v) \in E$: let $\text{Cnt}(u) := \text{Cnt}(u) + 1$.

(e) For each $u \in S$, if $\text{Cnt}(u) + E(u) + N > T$ where $N := \text{Geom}(\epsilon^2)$ denotes a fresh noise:

```plaintext
// This line is executed efficiently as mentioned in Section 3.1.
```

- Input $\text{Cnt}(u)$ to $P\text{Sum}(u)$;
- Based on the updated value $D(u) - P\text{Sum}(u)$, relocate $u$ to a new bucket if necessary;
- Reset $\text{Cnt}(u) := 0$, and resample a fresh $E(u) := \text{Geom}(\epsilon^2)$.

2: return $S^*$ and $d^* := \min \left( \frac{E(S^*) + \text{Geom}(\exp(\epsilon'))}{|S^*|}, |S^*| \right)$.

where $S^*$ is the subgraph returned by the DP algorithm. In other words, we measure the accuracy of the algorithms based on their relative performance in comparison to Charikar’s algorithm.

Figure 1 shows the accuracy of our linear-time algorithm as well as the accuracy of the $(\epsilon, \delta)$-DP algorithm proposed by Nguyen and Vullikanti [NV21]. In our experiments, we have set the parameter $\sigma$ equal to $2^{-30} \approx 10^{-9}$ for our algorithm. We have also set $\delta = 10^{-9}$ for the $(\epsilon, \delta)$-DP algorithm proposed by Nguyen and Vullikanti. As it is shown in the figure, our proposed algorithm finds a much more accurate solution when $\epsilon$ is small. However, it achieves almost the same accuracy for large choices of $\epsilon$. Note that we have used SEQDENSEDP algorithm to measure the accuracy of the work of Nguyen and Vullikanti which has better accuracy than their parallel algorithms [NV21].

We also study the running time of our linear time algorithm and compare it with the running time of SEQDENSEDP algorithm proposed by Nguyen and Vullikanti. Table 2 shows the average running time of the algorithms where the average is taken over 100 trials. Our experimental results show that in the RAM model, our linear time algorithm is much faster than the SEQDENSEDP algorithm. In fact, our linear time algorithm is around 100 times faster than the $(\epsilon, \delta)$-DP algorithm of Nguyen and Vullikanti.

### References


Differentially Private Densest Subgraph

'07, pages 75–84, New York, NY, USA, 2007. ACM.


Failed Naïve Approaches

A standard technique for attaining differential privacy is to add noise to the answer calibrated to the global sensitivity of the function being computed [DMNS06]. We argue that this approach fails to give meaningful utility since the densest subgraph problem has high global sensitivity.

Reporting the densest subgraph has high global sensitivity. The densest subgraph problem has high global sensitivity if we require that the algorithm outputs the set of vertices that form a dense subgraph. For example, one can easily construct a family of graphs $G$ with $n$ vertices, and satisfying the following properties:

- $G$ contains two disjoint sets of vertices $S$ and $S'$ of densities $d$ and $d'$ respectively. There are no edges between $S$ and $S'$.
- The densest subgraph of $S$ is $S$ itself; and the densest subgraph of $S'$ is $S'$ itself. Furthermore, $d < d' < d + 1/|S'|$.

This means that the densest subgraph of $G$ is $S'$; however, if we remove one edge from $S'$, the densest subgraph of the resulting graph would become $S$. Therefore, the ordinary approach of perturbing the output with noise roughly proportional to global sensitivity [DMNS06] does not apply.

Randomized response gives poor utility. A naïve approach for solving the DP densest subgraph problem is to rely on randomized response [War65, DR14, Vah17] where we use $G = (V, E)$ to denote the original graph given as input:

1. First, generate a rerandomized graph $\tilde{G}$ as follows: for each $(i, j) \in V^2$ and $i < j$, flip the existence of the edge $(i, j)$ with probability $p := \frac{1}{1+e^\epsilon}$.  
2. Then, run an exact densest subgraph algorithm on the rerandomized graph $\tilde{G}$. Output the densest subgraph $S* \subseteq V$ found.
3. Output $d* := (|E(S*)| + \text{Geom}(e'))/|S*|$ as an estimate of $d(G)$.

It is not hard to show that the above algorithm satisfies $2\epsilon$-DP. However, the algorithm fails to give a good approximation. To understand why, consider the following example — henceforth we will use $d^G(\cdot)$ and $d^\tilde{G}(\cdot)$ to denote the density of a subset of vertices in the original graph $G$ and the rerandomized graph $\tilde{G}$, respectively. Suppose that $\epsilon = 1$ and thus $p = \Theta(1)$.

Consider some graph $G = (V, E)$, in which the true densest subgraph of $G$ is a clique of size $\sqrt{n}$, and therefore $d(G) = O(\sqrt{n})$. Moreover, suppose that all vertices not in $S$ do not have any edges in $G$. One can show that for any fixed subset $U \subseteq V$ of size at least $\frac{n}{\log^2 n}$, with $1 - o(1)$ probability, $d^G(U) \in [p \cdot |U|/2, 2p \cdot |U|]$. Thus with $1 - o(1)$ probability, the naïve algorithm will report a large subgraph $U^* \approx V$ containing almost all the vertices. However, when $U^* \approx V$, the true density $d^G(U^*) = O(1)$ which is a $O(\sqrt{n})$ factor smaller than the true answer $d(G)$. In other words, the reported set $U^*$ is not a good approximation of the densest subgraph of $G$.

Deferentially Private Prefix Sum Mechanism

Dwork et al. [DNPR10] as well as Chan, Shi, and Song [CSS10, CSS11] suggest a DP prefix sum mechanism. Initially, the mechanism is initialized with $N$, which is an upper bound on the total number of values that will arrive, and at the time of initialization, the mechanism’s output is defined to be 0. Next, a sequence of at most $N$ integer values arrive one by one, and the value that arrives at time $t \in [N]$ is denoted $x_t$. In every time step $t$, the mechanism outputs an estimate $\text{PSum}_t$ of the prefix sum $\sum_{t' \leq t} x_{t'}$. The term $\text{err}_t := |\text{PSum}_t - \sum_{t' \leq t} x_{t'}|$ measures the error of the estimate at time $t$.

We say that two integer sequences $x := (x_1, \ldots, x_N)$ and $x' := (x'_1, \ldots, x'_N)$ are neighboring, if the vector $x - x'$ has exactly one coordinate that is either 1 or $-1$, and all other coordinates are 0. A prefix sum mechanism for length-$N$ sequences is said to satisfy $\epsilon$-adaptive-DP, iff for any admissible (even unbounded) adversary $A$, for any set $\Gamma$, $\Pr[\text{Expt}_b^A \in \Gamma] \leq e^\epsilon \cdot \Pr[\text{Expt}_1^A \in \Gamma]$, where for $b \in \{0, 1\}$, $\text{Expt}_b^A$ is defined as follows:

$$\text{Expt}_b^A := \sum_{t} A(x_1, \ldots, x_t) \cdot I_b(x_t),$$
Fix two arbitrary neighboring graphs $G$ and $\tilde{G}$ that differ in only one edge. In the proof below, we use the notation $\Pr[$] to denote the probability when $G$ is used as the input, and we the notation $\tilde{\Pr}[$] to denote the probability when $\tilde{G}$ is used as input. In our algorithm, every time a vertex is removed from the residual set $S$, we call it a time step denoted $t$. Let $U_t$ be the set of vertices whose $\text{PSum}$ instance is updated during time step $t$. For a vertex $u \in U_t$, let $\Sigma_t(u)$ be the new output of $\text{PSum}(u)$ after the update. Henceforth, we use the notation $\text{pre}^u_t$ to denote the time steps up to $t$ (inclusive) in which $\text{PSum}(u)$ is updated, the increment passed as input to $\text{PSum}(u)$ during each of these updates, and the new outcome of $\text{PSum}(u)$ after each update. We use $I_u$ to denote the time steps in which $u$ is updated.

For an execution of the algorithm, we define the trace as $\{D(v)\}_{v \in V}, \{U_t, \{\Sigma_t(u)\}_{u \in U_t}\}_t$, where $t$ is different time steps of the algorithm. Note that given the trace, one can uniquely determine the sequence of the vertices removed in the densest subgraph algorithm. Throughout the proof, we fix an arbitrary trace $\text{tr} = \{D(v)\}_{v \in V}, \{U_t, \{\Sigma_t(u)\}_{u \in U_t}\}_t$, and we show that $e^{-\epsilon} \cdot \tilde{\Pr}[\text{tr}] \leq \Pr[\text{tr}] \leq e^\epsilon \cdot \Pr[\text{tr}]$, hence the algorithm is $\epsilon$-DP.

First, consider when $G$ is the input. Let $\text{pre}_0 := \{D(v)\}_{v \in V}$ and for $t > 0$, let $\text{pre}_t := (\text{pre}_{t-1}, U_t, \{\Sigma_t(u)\}_{u \in U_t})$. We have the following:

$$\Pr[\{D(v)\}_{v \in V}, \{U_t, \{\Sigma_t(u)\}_{u \in U_t}\}_t] = \Pr[\text{pre}_0] \cdot \prod_{t > 0} \Pr[U_t | \text{pre}_{t-1}] \cdot \prod_{u \in U_t} \Pr[\text{PSum}(u) \text{ outputs } \Sigma_t(u) \text{ on new input}]$$

$$\text{Cnt}^G_t(u) | (\text{pre}_{t-1}, U_t) \right) \right) = \Pr[\text{pre}_0] \cdot \prod_{t > 0} \left( \Pr[U_t | \text{pre}_{t-1}] \right)$$

$$\prod_{u \in U_t} \Pr[\text{PSum}(u) \text{ outputs } \Sigma_t(u) \text{ on new input}]$$

$$\text{Cnt}^G_t(u) | (\text{pre}_{t-1})$$

$$\prod_{u \in U_t} \Pr[\text{PSum}(u) \text{ outputs } \Sigma_t(u) \text{ on new input}]$$

$$\{\text{Cnt}^G_t(u)\}_{t \in I_u}$$

Similarly, we have that

$$\Pr[\{D(v)\}_{v \in V}, \{U_t, \{\Sigma_t(u)\}_{u \in U_t}\}_t] = \Pr[\text{pre}_0] \cdot \prod_{t > 0} \Pr[U_t | \text{pre}_{t-1}] \cdot \prod_{u \in V} \Pr[\text{PSum}(u) \text{ outputs } \Sigma_t(u) \text{ on updates } \{\text{Cnt}^G_t(u)\}_{t \in I_u}]$$

Claim C.1. $e^{-\epsilon\sigma} \cdot \Pr[\text{pre}_0] \leq \tilde{\Pr}[\text{pre}_0] \leq e^{\epsilon\sigma} \cdot \Pr[\text{pre}_0]$

Proof. Observe that $G$ and $\tilde{G}$ differ in at most one edge, and the (non)-existence of every edge affects the
degree of at most two vertices. The claim therefore follows from Fact 1.

\[ e^{-c_2} \cdot \prod_{t>0} \Pr[U_t|\text{pre}_{t-1}] \leq \prod_{t>0} \Pr[U_t|\text{pre}_{t-1}] \leq e^{c_2} \cdot \prod_{t>0} \Pr[U_t|\text{pre}_{t-1}] \]

\[ \prod_{t>0} \Pr[U_t|\text{pre}_{t-1}] = \prod_{0<t'<t} \Pr[U_t|\text{pre}_{t-1}]. \]  

Note that the equality above follows from the fact that given \( \text{pre}_0 \), the algorithm has the same set of noisy degrees \( \{D(v)\} \) for both graphs \( G \) and \( \tilde{G} \).

Consider the time step \( t' \) where the algorithm removes \( i \) from \( S \). When we run the algorithm on \( G \), vertex \( j \) is a neighbor of \( i \). Thus, the algorithm increases the \( \text{Cnt}(j) \) in this time step. However, this is not the case in \( \tilde{G} \), and the \( \text{Cnt}(j) \) remains the same when we run the algorithm on \( \tilde{G} \). We consider two cases for the rest of the proof.

- **Case 1:** The first case is when \( j \notin U_t \) for any \( t \geq t' \). This means that the algorithm never updates the \( P\text{Sum}(j) \) after the step \( t' \). Let \( t'' \) be the time step where the algorithm removes \( j \) from \( S \) in Step 4 of the algorithm. It is easy to see that given the trace of the algorithm, we can uniquely determine \( t'' \). We then have

\[ \prod_{t' \leq t \leq t''} \Pr[U_t|\text{pre}_{t-1}] = \prod_{t' \leq t \leq t''} \left( \prod_{u \in U_t} \Pr[u \in U_t|\text{pre}_{t-1}] \cdot \prod_{u \notin U_t} \Pr[u \notin U_t|\text{pre}_{t-1}] \right) \]

Similarly, for the graph \( \tilde{G} \), we have

\[ \prod_{t' \leq t \leq t''} \Pr[U_t|\text{pre}_{t-1}] = \prod_{t' \leq t \leq t''} \left( \prod_{u \in U_t} \Pr[u \in U_t|\text{pre}_{t-1}] \cdot \prod_{u \notin U_t} \Pr[u \notin U_t|\text{pre}_{t-1}] \right). \]

In order to complete the proof of this case, we first claim that for every vertex \( u \neq j \), we have

\[ \prod_{t \in [t', t'']} \Pr[u \in U_t|\text{pre}_{t-1}] \cdot \prod_{t \in [t', t'']} \Pr[u \notin U_t|\text{pre}_{t-1}] = \prod_{t \in [t', t'']} \tilde{\Pr}[u \notin U_t|\text{pre}_{t-1}]. \]

The claim clearly holds for every vertex \( u \neq i, j \), since \( u \) has exactly the same set of neighbors in both graphs. Therefore, given \( \text{pre}_{t-1} \) we have \( \text{Cnt}^G(u) = \text{Cnt}^G(u) \), thus the claim holds. Also, considering the vertex \( i \), the algorithm removes \( i \) from \( S \) at the time step \( t' \), and it never updates the prefix sum for \( i \) after that. Therefore \( \Pr[i \in U_t|\text{pre}_{t-1}] = \tilde{\Pr}[i \in U_t|\text{pre}_{t-1}] = 0 \) for any \( t \geq t' \). Similarly, we have \( \Pr[i \notin U_t|\text{pre}_{t-1}] = \tilde{\Pr}[i \notin U_t|\text{pre}_{t-1}] = 1 \) for any \( t \geq t' \) which implies our claim for the vertex \( i \). We now give the following bound for vertex \( j \).

**Claim C.3.** For Case 1, let \( p = \prod_{t \in [t', t'']} \Pr[j \in U_t|\text{pre}_{t-1}] \cdot \prod_{t \in [t', t'']} \Pr[j \notin U_t|\text{pre}_{t-1}] \) and \( \bar{p} = \prod_{t \in [t', t'']} \Pr[j \in U_t|\text{pre}_{t-1}] \cdot \prod_{t \in [t', t'']} \Pr[j \notin U_t|\text{pre}_{t-1}] \).
Differentially Private Densest Subgraph

\[ U_t [\text{pre}_{t-1}] \cdot \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \]

\[ \leq \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \cdot e^{-c_2} \cdot \Pr [j \notin U_t [\text{pre}_{t-1}]] \]

\[ \leq e^{-c_2} \cdot \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \cdot \Pr [j \notin U_t [\text{pre}_{t-1}]] \cdot e^{-c_2} \cdot \tilde{p} \]

\[ \leq \prod_{t' \leq t' \leq t''} \tilde{\Pr} [U_t [\text{pre}_{t-1}]] = \prod_{t' \leq t''} \tilde{\Pr} [U_t [\text{pre}_{t-1}]] \cdot \tilde{p} \]

The equality above along with equalities (1), (4) and also Claim C.3 implies Claim C.2 for this case.

- **Case 2:** The second case is when \( j \in U_t \) for some \( t \geq t' \). Let \( t'' \geq t' \) be the smallest index such that \( j \in U_t \). It is easy to verify that equations (2) and (3) still hold in Case 2. Also, Equation (4) holds for every vertex \( u \neq j \). We now give the following bound for vertex \( j \).

**Claim C.4.** For Case 2, let \( p = \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \cdot \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \) and \( \tilde{p} = \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \), we then have \( e^{-c_2} \cdot \tilde{p} \leq p \leq e^{c_2} \cdot \tilde{p} \).

\[ \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \cdot \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] = \Pr [j \notin U_t [\text{pre}_{t-1}]] \cdot \prod_{t' \leq t' \leq t''} \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \]

\[ \leq \tilde{\Pr} [j \notin U_t [\text{pre}_{t-1}]] \cdot \Pr [j \notin U_t [\text{pre}_{t-1}]] \cdot e^{-c_2} \cdot \tilde{p} \]

\[ \leq e^{-c_2} \cdot \tilde{p} \leq p \leq e^{c_2} \cdot \tilde{p} \]

Considering any time step \( t > t'' \), the algorithm has removed both vertices \( i \) and \( j \), and the induced
subgraph between the vertices in \( S \) is exactly the same for both graphs \( G \) and \( \tilde{G} \). The algorithm also resets the counters \( \text{Cnt}^G(j) \) and \( \text{Cnt}^{\tilde{G}}(j) \) at the time step \( t'' \). Furthermore, the \( \text{PSum} \) values are exactly the same for \( G \) and \( \tilde{G} \) conditioning on \( \text{pre}_{\epsilon''} \). Thus, the algorithm does not see any difference between \( G \) and \( \tilde{G} \) after the time step \( t'' \) and we have

\[
\prod_{t'' < t} \Pr[U_t|\text{pre}_{t-1}] = \prod_{t'' < t} \tilde{\Pr}[U_t|\text{pre}_{t-1}].
\]

The equality above along with equalities (1), (4) and also Claim C.4 implies Claim C.2 for this case.

This completes the proof for both Case 1 and Case 2 and proves Claim C.2.

**Claim C.5.** Let

\[ p := \prod_{u \in V} \Pr[\text{PSum}(u) \text{ outputs}] \]

\[ \{\Sigma_t(u)\}_{t \in I_u} \text{ on updates } \{\text{Cnt}^G_t(u)\}_{t \in I_u} \]

and let

\[ \tilde{p} := \prod_{u \in V} \tilde{\Pr}[\text{PSum}(u) \text{ outputs}] \]

\[ \{\Sigma_t(u)\}_{t \in I_u} \text{ on updates } \{\text{Cnt}^{\tilde{G}}_t(u)\}_{t \in I_u} \]

It must be that \( e^{-\epsilon_1} p \leq \tilde{p} \leq e^{\epsilon_1} p \).

**Proof.** Since \( G \) and \( \tilde{G} \) are neighboring, there is at most one \((u, t)\) pair where \( t \in I_u \) such that \( \text{Cnt}^G_t(u) \) and \( \text{Cnt}^{\tilde{G}}_t(u) \) differ by 1. For all other \((u, t)\) pair where \( t \in I_u \), \( \text{Cnt}^G_t(u) \) and \( \text{Cnt}^{\tilde{G}}_t(u) \) must be the same. The claim therefore follows from the \( \epsilon_1 \)-DP of the prefix sum mechanism.

**Proof of Theorem 3.1.** Let \( S^* \) be the subgraph output by the algorithm. Observe that \( S^* \) is uniquely determined by trace \( \{D(v)\}_{v \in V}, \{U_t, \{\Sigma_t(u)\}_{u \in U_t}\}_t \). Therefore, given Claims C.1, C.2, and C.5, we conclude that for any \( S^* \), \( e^{-(\epsilon_0 + \epsilon_1 + \epsilon_2)} \cdot \Pr[S^*] \leq \Pr[S^{*}] \leq e^{(\epsilon_0 + \epsilon_1 + \epsilon_2)} \cdot \tilde{\Pr}[S^{*}] \). Besides \( S^* \), the algorithm also needs to output \( d^* \). Since \( G \) and \( \tilde{G} \) differ in at most one edge, and due to the distribution of the noise in computing \( d^* \), it follows that for any \( d^* \) and \( S^* \),

\[ e^{-\epsilon} \Pr[d^*|S^*] \leq \tilde{\Pr}[d^*|S^{*}] \leq e^{\epsilon} \Pr[d^*|S^*] \]

Therefore, for any \( S^* \) and \( d^* \), we have that \( e^{-(\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon')} \cdot \Pr[S^*, d^*] \leq \Pr[S^*, d^*] \leq e^{(\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon')} \cdot \tilde{\Pr}[S^*, d^*] \).
where \( S \) is the current residual set — henceforth, we use \( \text{TruePSum}(v) \) to mean the true prefix sum of all inputs that have been sent to \( \text{PSum}(v) \):

\[
\begin{align*}
|D(v) - \text{PSum}(v)| & = |D(v) - \text{PSum}(v) - (\deg(v) - |E(v, V\setminus S)|)| \\
& \leq |D(v) - \deg(v)| + |E(v, V\setminus S)| - \text{PSum}(v) \\
& = |D(v) - \deg(v)| + \text{Cnt}(v) + \text{TruePSum}(v) - \text{PSum}(v) \\
& \leq |D(v) - \deg(v)| + \text{Cnt}(v) + |\text{TruePSum}(v) - \text{PSum}(v)| \\
& \leq O\left(\frac{1}{\epsilon} \log n \cdot \log \frac{1}{\sigma}\right) + T + \\
& O\left(\frac{1}{\epsilon} \log n \cdot \log \frac{1}{\sigma}\right) + O\left(\frac{1}{\epsilon} \log^{2.5} n \cdot \log \frac{1}{\sigma}\right) \\
& \leq O\left(\frac{1}{\epsilon} \log^{2.5} n \cdot \log \frac{1}{\sigma}\right) + T
\end{align*}
\]

Claim C.8. Consider some execution of our algorithm, and let and let \( \text{err} := \max_{t \in S_t} |D(v) - \text{PSum}_t(v)| - |E(v, S_t)| \) where \( S_t \) denotes the residual set at the beginning of time step \( t \), and \( \text{PSum}_t(v) \) denotes the output of \( \text{PSum}(v) \) at the beginning of time step \( t \). Then, \( d(S^*) \geq (d(G) - 4 \cdot \text{err})/2 \).

Proof. Fix an arbitrary time step \( t \), it holds that \( \min_{v \in S_t} |E(v, S_t)| \leq 2|E(S_t)|/|S_t| \). Let \( v_t \in S_t \) be the actual vertex that is removed in time step \( t \), i.e., \( v_t := \arg \min_{v \in S_t} |D(v) - \text{PSum}(v)| \). It therefore holds that \( |E(v_t, S_t)| - \min_{v \in S_t} |E(v, S_t)| \leq 2 \cdot \text{err} \).

Now, suppose that as a vertex \( v_t \) gets removed from the residual graph \( S_t \), all edges \( E(v_t, S_t) \) are oriented towards \( v_t \). Observe that the \( d_{\text{max}} := \max_{v \in S_t} |D(v) - \text{PSum}(v)| \) value at the end of the algorithm is a good estimate of \( \max_{v \in S_t} |E(v, S_t)| \). Specifically, \( d_{\text{max}} \geq \max_{v \in S_t} |E(v, S_t)| - 2 \cdot \text{err} \). Henceforth let \( t^* := \arg \max_{v \in S_t} |D(v) - \text{PSum}(v)| \), and therefore, our algorithm’s output \( S^* := S_{t^*} \).

By Lemma C.6, it holds that \( d(G) \leq \max_{v \in S_t} |E(v, S_t)| \leq d_{\text{max}} + 2 \cdot \text{err} = D(v_{t^*}) - \text{PSum}(v_{t^*}) + 2 \cdot \text{err} \leq \min_{v \in S_{t^*}} |E(v, S_{t^*})| + 2 \cdot \text{err} + 2 \cdot \text{err} = 2|E(S^*)|/|S_{t^*}| + 4 \cdot \text{err} = 2|E(S^*)|/|S^*| + 4 \cdot \text{err} \). In other words, \( d(S^*) \geq (d(G) - 4 \cdot \text{err})/2 \).

Finally, observe that by the definition of \( d^* \) and due to Fact 1, with probability \( 1 - 0.1\sigma \), \( |d^* - d(S^*)| \leq O(\frac{1}{\epsilon} \log \frac{1}{\sigma}) \). Theorem 3.2 now follows from this fact, as well as Claims C.7 and C.8, and the choice of \( T \).

D Deferred Proofs for the Linear-Time Algorithm

In this section, we prove Theorem 4.1. The DP proof is the same as Theorem 3.1. For the utility analysis, we may assume that whenever \( \text{PSum}(u) \) is updated from \( \Sigma \) to \( \Sigma' \) for any vertex \( u \), it holds that \( \Sigma' - \Sigma < \text{err}/2 \) where \( \text{err} \) is the discretization parameter used in the bucketing — using the same type of arguments as the proof of Claim C.7, one can show that this indeed holds except with 0.1\( \sigma \) probability. This means that no vertex \( v \)'s noisy residual degree \( D(v) - \text{PSum}(v) \) should shrink by more than \( \text{err}/2 + 1 \) in two adjacent time steps, i.e., whenever a vertex \( v \) moves buckets, it cannot move left by more than 1 bucket. Therefore, effectively, the only difference between this linear-time variant and the earlier algorithm in Section 3.1 is the following: in this new variant, we do not necessarily pick \( \arg \min_{v \in S_t} |E(v, S_t)| \) in every time step \( t \). We could pick a vertex \( v_t \) in time step \( t \) such that \( |v_t - \arg \min_{v \in S_t} |E(v, S_t)|| \leq \text{err} \). This introduces an additive \( \text{err} \) term in the proof of Theorem 3.2. Due to the choice of \( \text{err} \), and redoing Theorem 3.2 with the extra additive \( \text{err} \) term, we conclude that the above algorithm achieves \( (2, O(\frac{1}{\epsilon} \log^{2.5} n \cdot \log \frac{1}{\sigma})) \)-approximation with probability \( 1 - \sigma \).

We now bound the algorithm’s runtime. First, not counting the runtime associated with \( \text{PSum} \) updates, the rest of the algorithm is easily seen to take only \( O(n + m) \) time — specifically, the total work spent in Line 1 is at most \( O(n) \) due to the same reason as Charika’s non-DP, linear-time algorithm. Due to the runtime of \( \text{PSum} \) stated in Theorem B.1, to prove the statement about the runtime, it suffices to show that the number of \( \text{PSum} \) updates is upper bounded by \( 4m \) with probability \( \exp(-\Omega(m)) \). Consider running the algorithm till it makes exactly \( 4m \) \( \text{PSum} \) updates — if the algorithm ends before making \( 4m \) \( \text{PSum} \) updates, we can simply pad it to exactly \( 4m \) number of \( \text{PSum} \) updates by appending filler \( \text{PSum} \) updates at the end of the algorithm which does not affect the outcome. In this way, there is exactly one noise \( \mathcal{E} \) associated with each of the \( 4m \) \( \text{PSum} \) updates. Due to the choice of \( T \), the probability that each \( \mathcal{E} \) is greater than \( T \) is at most \( \sigma/n \). Due to the Chernoff bound, the probability that there exist \( 3m \) or more noises \( \mathcal{E} \) that are greater than \( T \) is \( \exp(-\Omega(m)) \). This means that for \( m \) of these \( \text{PSum} \) updates, the true increment input to the \( \text{PSum} \) instance must be at least 1, and thus one edge must be consumed for each of these \( m \) \( \text{PSum} \) updates. In other words, except with \( \exp(-\Omega(m)) \) probability, the algorithm must have ended after having made \( 4m \) or fewer \( \text{PSum} \) updates.
E  Lower Bound

Theorem 1.3. Let \( \alpha > 1, \epsilon > 0 \) be arbitrary constants, \( \exp(-n^{0.49}) < \sigma < 0.000001 \cdot \min(1, \epsilon, \exp(-\epsilon)) \), and \( 0 \leq \delta \leq \frac{\sigma \epsilon}{4 \sigma} \). Then, there exists a sufficiently small \( \beta = \Theta(\frac{1}{\alpha} \sqrt{\frac{1}{2 \log \frac{1}{\sigma}}} \) such that there does not exist an \( (\epsilon, \delta) \)-DP mechanism that achieves \( (\alpha, \beta) \)-approximation with \( 1 - \sigma \) probability.

Proof. Suppose there exists an \( (\epsilon, \delta) \)-DP mechanism denoted \( M \) that achieves \( (\alpha, \beta) \)-approximation with \( 1 - \sigma \) probability, for the parameters stated above. Let \( \beta = \frac{1}{1000} \sqrt{\frac{1}{\epsilon} \log \frac{1}{4\sigma}} \). We will reach a contradiction below. Thus, no \( (\epsilon, \delta) \)-DP mechanism can achieve a \( (\alpha, \beta) \)-approximation.

Consider a graph over \( n \) vertices \( V := [n] \). Suppose that a subset \( A \subseteq V \) of size \( 4\alpha\beta + 1 \) of the vertices form a clique, and there are no other edges in \( G \). Therefore, the densest subgraph of \( G \) is \( A \), and its density is \( \frac{|A|-1}{2} \) which is \( 2\alpha\beta \). Note that for our choices of parameters, we always have \( \alpha\beta \geq 1 \).

If we run the mechanism \( M \) over this graph \( G \), we know that with probability \( 1 - \sigma \), the true density of set of vertices output is at least \( \beta \). This means that with probability \( 1 - \sigma \), at most \( 4\alpha\beta(4\alpha\beta + 1)/(2\beta) \leq 16\alpha^2\beta \) vertices are output, since the graph \( G \) has \( (4\alpha\beta)(4\alpha\beta + 1)/2 \) total number of edges. In other words, the expected number of vertices output is upper bounded by

\[
(1 - \sigma) \cdot 16\alpha^2\beta + \sigma \cdot n. \tag{5}
\]

We claim that there must exist a set \( B \) of size \( 4\alpha\beta + 1 \) that is disjoint from \( A \) such that with probability at least \( 1/2 \), no vertex in \( B \) is output. Suppose that this is not the case, then, the expected number of vertices contained in the output is at least \( s := \frac{n - (4\alpha\beta + 1)}{4\alpha\beta + 1} \cdot \frac{1}{2} \).

For suitable choice of parameters as stated in the theorem statement, \( s \) would be greater than Equation (5) for sufficiently large \( n \), which leads to a contradiction.

Now, consider another graph \( G' \) in which the set \( B \) of vertices form a clique and there are no other edges in \( G' \). \( G' \) can be obtained by making \( 2 \cdot (4\alpha\beta + 1)(4\alpha\beta)/2 \leq 32\alpha^2\beta^2 \) edge modifications starting from \( G \).

We now use the group privacy theorem to derive a lower bound on the probability that mechanism \( M \) does not output any of the vertices in \( B \) if we run it on \( G' \).

Theorem E.1 (Group Privacy). Let \( M \) be a \( (\epsilon, \delta) \)-DP mechanism, and \( G, G' \) be two datasets. Then, for any subset \( U \) of the output space,

\[
\Pr[\text{Alg}(G) \in U] \leq e^{\kappa} \cdot \Pr[\text{Alg}(G') \in U] + \delta \cdot k \cdot e^{\kappa},
\]

where \( k \) is the hamming distance between \( G \) and \( G' \).

By \( (\epsilon, \delta) \)-DP and the group privacy theorem, it holds that if we run the mechanism \( M \) on the graph \( G' \), the probability that none of \( B \) is selected is at least

\[
\left( \frac{1}{2} - 32\alpha^2\beta^2\delta \exp(32\alpha^2\beta^2\epsilon) \right) \cdot \exp(-\epsilon \cdot 32\alpha^2\beta^2)
\]

\[
= \frac{1}{2} \exp(-32\epsilon \cdot \alpha^2\beta^2) - 32\alpha^2\beta^2\delta
\]

\[
> 2\sigma - \sigma = \sigma,
\]

where the inequality above is because \( \beta \leq \frac{1}{1000} \sqrt{\frac{1}{\epsilon} \log \frac{1}{4\sigma}} \) and \( \delta \leq \frac{\sigma \epsilon}{1000} \). Thus, with larger than \( \sigma \) probability, the true density of the set of vertices output is 0. Therefore, it is impossible that the mechanism \( M \) gives \( (\alpha, \beta) \)-approximation with \( 1 - \sigma \) probability (over any graph).