A Related Work and Sliding Window Algorithm

Differentially private graph analysis has received a lot of attention in the static graph setting with efficient algorithms known for many functions under edge-level privacy Eliáš et al. (2020); Gupta et al. (2012); Blocki et al. (2012); Hay et al. (2010); Xiao et al. (2014); Raskhodnikova and Smith (2016). These algorithms utilize various techniques including, but not limited to, multiplicative weight updates Gupta et al. (2012), random projections Blocki et al. (2012, 2013), private mirror descent Eliáš et al. (2020), and network flow Kasiviswanathan et al. (2013); Raskhodnikova and Smith (2016). While these techniques are provably efficient and work well in the static setting, it is not clear how to extend them to a more dynamic setting such as a sliding window model, even for most basic graph analysis.

In static setting, an important class of graph problems is estimating cut functions such as cut queries, Max-Cut, Min-Cut, and Sparsest-Cut. The question of privately answering cut functions on a graph was first explored by Gupta et al. (2012) and recently improved by Eliáš et al. (2020). Their algorithms are based on an additive noise mechanism, which itself cannot guarantee edge weights to be positive. To circumvent this, Gupta et al. (2012) makes use of linear programming techniques and Eliáš et al. (2020) make use of noisy mirror descent. However, it is not known whether these algorithms or techniques can be extended to spectrum-related analysis, a generalization of cut function, on graphs. Finally, the output of Dwork et al. (2014) is a full-rank matrix that may not even be positive semidefinite, let alone Laplacian of a graph. In other words, it cannot be used for several applications considered in this paper.

Private approximation of spectrum of graphs was subsequently addressed by Blocki et al. (2012). They showed that Johnson-Lindenstrauss transform preserves differential privacy under certain conditions on the spectrum of the graph and use it on the edge-incidence matrix of the input graph. Their approach outputs a positive semidefinite matrix that approximately preserves the spectrum of Laplacian of the graph. However, the output matrix may not correspond to the Laplacian of a graph since random projections do not preserve that (see Section 1.2 in Cohen et al. (2017) for more discussion). While this allows us to calculate the cut function for a given subset of the vertex set, it is not clear how to find the partitioning of vertices that induce Max-Cut or Sparsest-Cut with optimal approximation ratio. This is because the existing approximation algorithms assume that the edge weights of the graph are non-negative[2], which is not guaranteed by the Johnson-Lindenstrauss transform.

In a recent work, Arora and Upadhyay (2019) gave a polynomial-time differentially private algorithm to output spectral sparsification of graphs. Their approach is to estimate the effective resistance privately and then sample edges from a graph overlayed with an appropriately weighted complete graph. While efficient, their technique is arguably complicated and less implementation friendly.

Moving on to the dynamic setting, to the best of our knowledge, there has been no prior work on problems arising in private graph analysis in this setting. In sliding window setting, Bolot et al. Bolot et al. (2013) (for counting queries), Chan et al. Chan et al. (2012) and Upadhyay Upadhyay (2019) (estimating heavy hitters) have focused on private data analysis. However, our work and approach are unrelated to their work.

Some work have studed degree distribution and the number of high degree nodes in the continual release model Dwork et al. (2010a); Song et al. (2018). Similarly, with recent advances in dynamic graph sparsification Kapralov et al. (2019); Kyng et al. (2017), Upadhyay Upadhyay (2013) demonstrated a way for computing cut queries. However, these approaches consider the entire data stream to be useful, whereas we consider only the last W updates to be useful.

Note that privacy is with respect to the entire stream while accuracy is with respect to the sliding window. There are some closely related privacy definitions: *continual release model* Dwork et al. (2010a) and *pan-privacy streaming model* Dwork et al. (2010b) are accurate for entire stream and *privacy with expiration* Bolot et al. (2013) classify only the current window as private.

[2]For Max-Cut, the approximation ratios achieved by SDP based algorithm when edge weights range over real or are assumed to be non-negative are remarkably different. When weights are non-negative, one has constant factor approximation Goemans and Williamson (1995); when weights are real (a problem known as quadratic programming problem), one has $O(\log(n))$ approximation Charikar and Wirth (2004) with tight integrality gap Alon et al. (2006); Khot et al. (2007).

B Notation and terminology

We use the notation $\mathbb{R}$ to denote the space of real numbers and $\mathbb{N}$ to denote the space of natural numbers.

Linear algebra. The space of n-dimensional vectors over reals is denoted $\mathbb{R}^n$. The set of non-negative vectors (also known as non-negative orthant) and the set of strictly positive vectors in $\mathbb{R}^n$ are denoted $\mathbb{R}^n_+$ and $\mathbb{R}^n_{++}$, respectively. For a vector x, we let $x^\top$ denote the transpose of the vector. We reserve the letters x, y, z to denote real vectors in $\mathbb{R}^n$. The entries of a vector $x \in \mathbb{R}^n$ is denoted as follows:

$$x = \big(x[1], x[2], \ldots, x[n]\big)^\top.$$

We let $\{\bar{e}_i : i \in [n]\}$ (where $[n] := \{1, 2, \ldots, n\}$) denote the set of standard basis vectors of $\mathbb{R}^n$. That is,

$$\bar{e}_i[j] = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

We let the vector of all 1's denoted by $\bar{e}$, which can be expressed in terms of basis vectors as follows: $\bar{e} = \bar{e}_1 + \bar{e}_2 + \cdots + \bar{e}_n$. For two vectors $x, y \in \mathbb{R}^n$, their inner product is denoted $\langle x, y\rangle$ and is defined as

$$\langle x, y\rangle := \sum_{i=1}^{n} x[i]y[i].$$

The *Euclidean norm* of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_2 := \sqrt{\langle x, x\rangle}$.

The set of real $n \times m$ matrices is denoted $\mathbb{R}^{n\times m}$. For a real matrix A, the transpose of it is denoted $A^\top$ and the entries are denoted $A[i, j]$. The linear mapping

$$\mathrm{vec} : \mathbb{R}^{n\times m} \to \mathbb{R}^{nm}$$

denotes the *vectorization* of the matrix and is defined on matrices $e_i e_j^\top$ as

$$\mathrm{vec}\left(\bar{e}_i \bar{e}_j^\top\right) = \bar{e}_i \otimes \bar{e}_j$$

where $\bar{e}_i \otimes \bar{e}_j \in \mathbb{R}^{nm}$ is tensor product of $\bar{e}_i$ and $\bar{e}_j$. In this paper, we will mostly be working with matrices in $\mathbb{R}^{n\times n}$. For a matrix $A \in \mathbb{R}^{n\times n}$, $\mathsf{Tr}(A)$ is sum of the diagonal entries of A. For two matrices $A, B \in \mathbb{R}^{n\times n}$, their inner product is denoted $\langle A, B\rangle$ and is defined $\langle A, B\rangle := \mathsf{Tr}\left(A^\top B\right)$. The following special classes of matrices are relevant to this paper.

1. A real matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric* if $A = A^\top$. The set of symmetric matrices is denoted $\mathbb{S}^n$ and forms a vector space over $\mathbb{R}$. The eigenvalues of symmetric matrices are real.
2. A symmetric matrix $A \in \mathbb{S}^n$ is *positive semidefinite* if all of its eigenvalues are non-negative. The set of such matrices is denoted $\mathbb{S}^n_+$. The notation $A \succeq 0$ indicates that A is positive semidefinite and the notations $A \succeq B$ and $B \preceq A$ indicate that $A - B \succeq 0$ for symmetric matrices A and B. We also use the notation $A \not\succeq B$ and $B \not\preceq A$ for $A, B \in \mathbb{S}^n$ to say that $A - B \notin \mathbb{S}^n_+$.
3. A positive semidefinite matrix $A \in \mathbb{S}^n_+$ is *positive definite* if all of its eigenvalues are strictly positive. The set of such matrices is denoted $\mathbb{S}^n_{++}$. The notation $A \succ 0$ indicates that A is positive definite and the notations $A \succ B$ and $B \prec A$ indicate that $A - B \succ 0$ for symmetric matrices A and B.
4. A matrix $U \in \mathbb{R}^n$ is *orthonormal* if $UU^\top = U^\top U = \mathbb{1}_n$, where $\mathbb{1}_n$ is the identity matrix. We will drop the subscript n from $\mathbb{1}_n$ when the dimension is understood from the context.

The eigenvalues of any symmetric matrix $A \in \mathbb{S}^n$ are denoted by $(\lambda_1(A), \ldots, \lambda_n(A))$ sorted from largest to smallest: $\lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_n(A)$. When discussing the largest and smallest eigenvalues, we alternately use the notation $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ to denote $\lambda_1(A)$ and $\lambda_n(A)$, respectively. Similarly, the singular values of A is denoted by the tuple $(s_1(A), \ldots, s_n(A))$ sorted from largest to smallest: $s_1(A) \geq s_2(A) \geq \cdots \geq s_n(A)$. We use the notation $s_{\max}(A)$ and $s_{\min}(A)$ to denote the largest and smallest singular values of A, respectively. It is a well known fact that for any symmetric matrix A

$$s_{\max}(A) = \max\left\{|\lambda_{\max}(A)|, |\lambda_{\min}(A)|\right\} \qquad \text{and} \qquad s_{\min}(A) = \min\left\{|\lambda_{\max}(A)|, |\lambda_{\min}(A)|\right\}.$$

For a matrix $A \in \mathbb{R}^{n\times m}$, the singular values of A is given by the tuple

$$\left(\sqrt{\lambda_1(A^\top A)}, \sqrt{\lambda_2(A^\top A)}, \ldots, \sqrt{\lambda_n(A^\top A)}\right).$$

The maximum number of non-zero singular values of $A \in \mathbb{R}^{n\times m}$ is $\min\{n, m\}$. The *spectral norm* of a matrix $A \in \mathbb{R}^{n\times m}$ is defined as

$$\|A\|_2 = \max\{\|Ax\|_2 \,:\, x \in \mathbb{R}^m,\, \|x\|_2 = 1\}.$$

The spectral norm of A is equal to the largest singular value of A.

Two types of matrix decomposition are used in this paper. The first matrix decomposition is *spectral decomposition* (or *eigenvalue decomposition*). It means that a symmetric matrix $A \in \mathbb{S}^n$ can be written as

$$A = U\Lambda U^\top = \sum_{i=1}^{n} \lambda_i(A) x_i x_i^\top$$

where U is an orthonormal matrix, Λ is a diagonal matrix with eigenvalues of A on its diagonal, and the set $\{x_i \in \mathbb{R}^n : i \in [n]\}$ are set of orthonormal vectors known as eigenvectors of A. We note that orthonormal matrices can also be decomposed in above form. The second matrix decomposition that is relevant to this paper is *singular value decomposition* (or SVD for short). Any real matrix $A \in \mathbb{R}^{n\times m}$ can be decomposed as follows:

$$A = USV^\top = \sum_{i=1}^{\min\{n,m\}} s_i(A) x_i y_i^\top.$$

Here, $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ are orthonormal matrices, S is a diagonal matrix with diagonal entries singular values of A, and the sets $\{x_i \in \mathbb{R}^n : i \in \min\{n, m\}\}$ and $\{y_j \in \mathbb{R}^m : j \in \min\{n, m\}\}$ are orthonormal sets of vectors.

Probability distributions. For a random variable $X \in \mathbb{R}$, we denote the mean and variance of X by $\mathbb{E}[X]$ and $\mathsf{Var}(X)$, respectively. The symbols μ and σ^2 are reserved to represent these quantities. That is,

$$\mu = \mathbb{E}[X] \qquad \text{and} \qquad \sigma^2 = \mathsf{Var}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right].$$

At various points in this paper, we refer to moments of random variables. The k-th moment of a random variable (if it exists) is simply $\mathbb{E}\left[X^k\right]$. The existence of k-th moment of a random variable means that $\mathbb{E}\left[|X|^k\right] < \infty$ and $\mathbb{E}\left[X^k\right] \in \mathbb{R}$.

We say that a random variable $X \in \mathbb{R}$ has Gaussian (or normal) distribution if its probability density function is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

We denote Gaussian distribution by $\mathcal{N}\left(\mu, \sigma^2\right)$ and write $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$ when X has Gaussian distribution.

Apart from the above mentioned distribution, we also make use of the an important family of distributions, known as *sub-Gaussian distribition*. A random variable $X \in \mathbb{R}$ is said to be *sub-Gaussian* if it has zero mean and its moment generating function satisfies

$$\mathbb{E}\left[\exp(aX)\right] \le \exp\left(\sigma^2 a^2/2\right) \qquad \text{for all} \qquad a \in \mathbb{R}.$$

The family of sub-Gaussian are denoted $\mathsf{SG}(0, \sigma^2)$. We slightly abuse notation and write $X \sim \mathsf{SG}\left(0, \sigma^2\right)$ to denote that the random variable X has a sub-Gaussian distribution. The notation $X \sim \mathsf{SG}\left(\mu, \sigma^2\right)$ represent families of distributions X such that $X - \mu \sim \mathsf{SG}\left(0, \sigma^2\right)$.

Throughout this paper, we discuss and work with *random matrices*. They are simply matrices with matrix entries drawn from random variables that may or may not be independent. The following is a well known fact about non-asymptotic bounds on singular values of random matrices.

Theorem 7 (Vershynin Vershynin (2012)). *Let $A \in \mathbb{R}^{n\times m}$ be a random matrix whose entries drawn from independent sub-Gaussian random variables with sub-Gaussian moments bounded by 1. Then for $t > 0$,*

$$\Pr\left[s_{\max}(A) \geq C_g(\sqrt{n}+\sqrt{m})+t\right] \leq e^{-c_g t^2}$$

where c_g and C_g are positive absolute constants.

Differential privacy. Roughly speaking, differential privacy is a robust guarantee of privacy which makes confidential data available for accurate analysis while simultaneously preserving the privacy of individual data. Achieving these two requirements at the same time appears to be paradoxical. On one hand, we expect no or limited information leakage about an individual. And on the other hand, we expect that the database answers to the queries with high accuracy.

One of the key features of differential privacy is that it is preserved under arbitrary post-processing, i.e., an analyst, without additional information about the private database, cannot compute a function that makes an output less differentially private. In other words,

Lemma 1 (Dwork et al. (2006b)). *Let $M(D)$ be an (ϵ,δ)-differential private mechanism for a database D , and let h be any function, then any mechanism $M' := h(M(D))$ is also (ϵ,δ)-differentially private for the same set of tasks.*

Let $f(\cdot)$ be a function from a class of Δ-sensitive functions. The Gaussian mechanism is

$$M(D, f(\cdot), \varepsilon) := f(D) + (X_1, \ldots, X_k),$$
$$\text{where } X_i \sim \mathcal{N}\left(0, \frac{\Delta^2}{\varepsilon^2}\log(5/4\delta)\right).$$

Dwork et al. (2006a) proved the following.

Theorem 8 (Gaussian mechanism Dwork et al. (2006a)). *Let $x, y \in \mathbb{R}^n$ be any two vectors such that $\|x-y\|_2 \leq c$. Let $g \in \mathbb{R}^n$ be a random vector whose entries are sampled from independently identically distributed Gaussian distribution as follows:*

$$g[i] \sim \mathcal{N}(0,\sigma^2) \quad \forall i \in [n] \qquad \text{where } \sigma = \frac{c}{\varepsilon}\sqrt{\log\left(\frac{1}{\delta}\right)}.$$

Then for any $S \subset \mathbb{R}^n$, $\Pr[x+g \in S] \leq \exp(\varepsilon)\Pr[y+g \in S] + \delta$.

Graph preliminaries. For a weighted graph $\mathcal{G} = (V, E, w)$, we let n denote the size of the vertex set V and m denote the size of the edge set E. When the graph is uniformly weighted (i.e., each edge weight is either same or 0), then the graph is denoted $\mathcal{G} = (V, E)$. Without loss of generality, one can assume that all edge weights are 1. The *Laplacian* of a graph is defined as the matrix $L_{\mathcal{G}}$ with entries

$$L_{\mathcal{G}}[u,v] = -w(u,v) \quad \text{if} \quad u \neq v \qquad \text{and} \qquad L_{\mathcal{G}}[u,u] = \sum_{v\in V\setminus\{u\}} w(u,v).$$

Here, $w(u,v)$ is the weight on the edges between vertices u and v. If $(u,v) \notin E$, then $w(u,v) = 0$. When the weights associated with the edges of graph are non-negative, the Laplacian of the graph is positive semidefinite. Moreover, let $W \in \mathbb{S}^m_+$ be a diagonal matrix with non-negative edge weights on the diagonal. If we define an orientation of the edges of graph, then we can define the *signed edge-vertex incidence matrix* $A_{\mathcal{G}} \in \mathbb{R}^{m\times n}$ as follows:

$$A_{\mathcal{G}}[e,v] = \begin{cases} 1 & \text{if } v \text{ is } e\text{'s head,} \\ -1 & \text{if } v \text{ is } e\text{'s tail,} \\ 0 & \text{otherwise.} \end{cases}$$

Simple algebra establishes that

$$L_{\mathcal{G}} = A_{\mathcal{G}}^\top W A_{\mathcal{G}} = \mathcal{E}^\top \mathcal{E}$$

where $\mathcal{E} = \sqrt{W}A_{\mathcal{G}}$ is called the *weighted signed edge-vertex incidence matrix.*

The Laplacian of graphs with non-negative weights are denoted by $\mathbb{L}^n \subset \mathbb{S}^n_+$. That is,

$$\mathbb{L}^n = \left\{ X : X[i,j] \leq 0 \quad \forall i,j \in [n] : i \neq j \quad \text{and} \quad \sum_{j=1}^{n} X[i,j] = 0 \quad \forall i \in [n] \right\}. \tag{4}$$

The set of Laplacian matrices forms a closed convex cone. It is a convex cone because if $X, Y \in \mathbb{L}^n$, then $\lambda X \in \mathbb{L}^n$ and $X + Y \in \mathbb{L}^n$. We also define a convex relaxation of $\mathbb{L}^n$ paratemerized by $\varrho \geq 0$ as below:

$$\mathbb{L}^n(\varrho) = \left\{ X : X[i,j] \leq 0 \quad \forall i,j \in [n] : i \neq j \quad \text{and} \quad 0 \leq \sum_{j=1}^{n} X[i,j] \leq \varrho \quad \forall i \in [n] \right\}. \tag{5}$$

It is not hard to see that $\mathbb{L}^n(0) = \mathbb{L}^n$ and $\mathbb{L}^n(\varrho) \subset \mathbb{S}^n_+$. We use the notation $A_{\mathcal{G}} \cup A_{\mathcal{H}}$ to denote the graph whose edge sets is the union of edges in $\mathcal{G}$ and $\mathcal{H}$. We slightly abuse notation and write $A_{\mathcal{G}} \cup e$ to denote a graph formed by adding the edge e to the graph $\mathcal{G}$.

We study a wide variety of graph related problems. We formally define them next. The first and most basic problem that we study is to answer cut queries privately.

Definition 3 ((S,T)-cut)**.** *For two disjoint subsets S and T, the size of the cut (S,T)-cut is denoted $\Phi_{S,T}(\mathcal{G})$ and defined as*

$$\Phi_{S,T}(\mathcal{G}) := \sum_{u \in S, v \in T} w(u,v).$$

When $T = V \backslash S$, we denote $\Phi_{S,T}(\mathcal{G})$ as $\Phi_S(\mathcal{G})$.

The partitioning of vertex set V of a graph $\mathcal{G}$ in to two disjoint subsets S and $V \backslash S$ based on optimizing $\Phi_S(\mathcal{G})$ leads to two widely studied NP-hard combinatorial optimization problems on graphs.

Definition 4 (Max-Cut)**.** *Given a graph $\mathcal{G} = (V, E, w)$, the maximum cut of the graph is*

$$\max_{S \subseteq V} \{\Phi_S(\mathcal{G})\} = \max_{S \subseteq V} \left\{ \sum_{u \in S, v \in V \backslash S} w(u,v) \right\}.$$

Let $\mathsf{OPT}_{\mathsf{max}}(\mathcal{G})$ denote the maximum value.

Based on semidefinite programming, Goemans and Williamson Goemans and Williamson (1995) showed that Max-Cut can be approximated within a factor of α_{GW} where $\alpha_{\mathsf{GW}} \approx 0.878567$.

Theorem 9 (Goemans and Williamson Goemans and Williamson (1995))**.** *Let $\rho > 0$ be an arbitrary small constant. For an n-vertex graph $\mathcal{G}$, there is a polynomial-time algorithm that produces a set of nodes S satisfying*

$$\Phi_S(\mathcal{G}) = (\alpha_{\mathsf{GW}} - \rho)\mathsf{OPT}_{\mathsf{max}}(\mathcal{G}),$$

where $\mathsf{OPT}_{\mathsf{max}}(\mathcal{G})$ is the optimal max-cut.

Assuming unique games conjecture, the approximation factor α_{GW} is proved to be the optimal Khot et al. (2007).

When privacy is a concern, it is not possible to achieve pure multiplicative approximation factor. One can only hope for a "mixed approximation". This motivates us to study the following variant of Max-Cut:

Definition 5 ((a,b)-Max-Cut)**.** *Given a graph $\mathcal{G} = (V, E, w)$, the (a,b)-Max-Cut of the graph requires to output a partition of nodes $(S, V \backslash S)$ such that*

$$\Phi_S(\mathcal{G}) \geq a \cdot \mathsf{OPT}_{\mathsf{max}}(\mathcal{G}) - b,$$

where $\mathsf{OPT}_{\mathsf{max}}(\mathcal{G})$ denote the maximum value.

The goal of (a,b)-Max-Cut is to minimize the value of b for a given a.

Definition 6 (Sparsest-Cut). *Given a graph $\mathcal{G} = (V, E)$, the sparsest-cut of the graph is*

$$\min_{S \subseteq V} \left\{ \frac{\Phi_S(\mathcal{G})}{|S|(|V| - |S|)} \right\}.$$

Let $\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G})$ denote the minimum value.

Definition 7 (Edge-Expansion). *Given a graph $\mathcal{G} = (V, E)$, the edge-expansion of the graph is*

$$\min_{S \subseteq V |S| \leq n/2} \left\{ \frac{\Phi_S(\mathcal{G})}{|S|} \right\}.$$

Let $\mathsf{OPT}_{\mathsf{edge}}(\mathcal{G})$ denote the minimum value.

Since $n/2 \leq |S| \leq n$, up to a factor 2 computing the Sparsest-Cut is the same as computing the Edge-Expansion of the graph.

Arora et al. (2009) showed an efficient algorithm that outputs a partitioning of vertices that achieve $O(\sqrt{\log(n)})$-approximation to Sparsest-Cut problem Arora et al. (2009).

Theorem 10 (Arora et al. Arora et al. (2009)). *For an n-vertex graph $\mathcal{G}$, there is a polynomial-time algorithm that produces a set of nodes S satisfying*

$$\Phi_S(\mathcal{G}) = O(\sqrt{\log(n)})\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}),$$

where $\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G})$ is the optimal sparsest cut.

As in the case of Max-Cut, when privacy is a concern, it is not possible to achieve pure multiplicative approximation, i.e., we can only hope for a mixed approximation. This motivates us to study the following variant of Sparsest-Cut and Edge-Expansion:

Definition 8 ((a,b)-Sparsest-Cut). *Given a graph $\mathcal{G} = (V, E, w)$, the (a,b)-Sparsest-Cut of the graph requires to output a partition of nodes $(S, V \backslash S)$ such that*

$$\frac{\Phi_S(\mathcal{G})}{|S|(n - |S|)} \leq a \cdot \mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}) + b,$$

where $\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G})$ denote the minimum value.

Definition 9 ((a,b)-Edge-Expansion). *Given a graph $\mathcal{G} = (V, E, w)$, the (a,b)-Edge-Expansion of the graph requires to output a partition of nodes $(S, V \backslash S)$ such that*

$$\frac{\Phi_S(\mathcal{G})}{|S|} \leq a \cdot \mathsf{OPT}_{\mathsf{edge}}(\mathcal{G}) + b,$$

where $\mathsf{OPT}_{\mathsf{edge}}(\mathcal{G})$ denote the minimum value.

The goal of (a,b)-Sparsest-Cut, and (a,b)-Edge-Expansion is to minimize the value of b for a given a.

One of the problems that we concentrate on in this paper in context of differential privacy is spectral sparsification of graphs Spielman and Teng (2011). It is based on spectral similarity of Laplacian of graphs.

Definition 10 (ρ-spectral sparsification of graphs Spielman and Teng (2011)). *For a weighted graph $\mathcal{G} = (V, E, w)$ where edge weights are non-negative, a graph $\widetilde{\mathcal{G}} = (V, \widetilde{E}, \widetilde{w})$ is called ρ-spectral sparsification of $\mathcal{G}$ if*

$$(1 - \rho)L_{\widetilde{\mathcal{G}}} \preceq L_{\mathcal{G}} \preceq (1 + \rho)L_{\widetilde{\mathcal{G}}}.$$

Spectral sparsification of graphs has been widely studied Allen-Zhu et al. (2015); Lee and Sun (2015); Spielman and Srivastava (2011); Spielman and Teng (2011). We use the following result given by Lee and Sun Lee and Sun (2015):

Theorem 11 (Lee and Sun (2015)). *Given a graph $\mathcal{H}$ and an approximation parameter $\rho > 0$, there is an algorithm* Sparsify *such that the graph $\widetilde{\mathcal{H}}$ formed as $\widetilde{\mathcal{H}} \leftarrow$ Sparsify $(\mathcal{H})$ is ρ-spectral sparsification of $\mathcal{H}$. Further,* Sparsify *outputs $\widetilde{\mathcal{H}}$ in $O(m)$ time.*

Definition 11 ((ρ, ζ, ϑ)-spectral sparsification of graphs)**.** *For a weighted graph $\mathcal{G} = (V, E, w)$ where edge weights are non-negative, a graph $\widetilde{\mathcal{G}} = (V, \widetilde{E}, \widetilde{w})$ is called (ρ, ζ, ϑ)-spectral sparsification of $\mathcal{G}$ if*

$$(1-\rho)L_{\widetilde{\mathcal{G}}} - \zeta L_{K_n} \preceq L_{\mathcal{G}} \preceq (1+\rho)L_{\widetilde{\mathcal{G}}} + \vartheta L_{K_n}.$$

In above, ζ and ϑ can be thought of as the distortion we are willing to accept to preserve privacy. Ideally we would like ζ and ϑ to be as small as possible.

C Proof of Main Theorem for Gaussian case

In this section, we prove the main theorem when instantiated with normal Gaussian distribution. This suffices for all our differential privacy results. We prove the more general result for sub-Gaussian random variables in Section E. Let $s_{\max}(A)$ denote the largest singular value of a matrix A and $s_{\min}(A)$ denote the smallest non-zero singular value of A. For Gaussian case, we will use the following classical result by Bai and Yin Bai and Yin (2008).

Theorem 12 (Bai-Yin's law Bai and Yin (2008); Tao (2012))**.** *Let G be an $n \times n$ symmetric random matrix whose upper triangular entries are independent copies of a random variable with zero mean, unit variance, and finite fourth moment. Then with probability $1 - e^{-cn}$, $s_{\max}(G) \le c_1\sqrt{n}$ and $s_{\min}(G) \ge c_2/\sqrt{n}$ for some constants $c, c_1, c_2 > 0$.*

For the ease of the readers, we restate the theorem for the Gaussian case.

Theorem 13 (Spectral theorem for Gaussian graphs)**.** *Let $\mathcal{G}$ be a complete graph on n vertices with edge weights identical and independent copies of $\mathcal{N}(0,1)$. Then*

$$\mathsf{Pr}\left[s_{\max}(L_{\mathcal{G}}) \le C\sqrt{n\log(n)}\right] \ge 1 - 3e^{-cn}$$

for some constants $C, c > 0$.

Proof. First note that we can write the Laplacian of graphs as $L_{\mathcal{G}} = G - D + S$, where G be the random matrix whose entries are sampled i.i.d. from $\mathcal{N}(0,1)$, D and S are diagonal matrices formed in the following manner:

$$D[i,i] = G[i,i], \quad S[i,i] = \sum_{j \neq i} G[i,j].$$

Now, using subadditivity of spectral norm, we have $s_{\max}(L_{\mathcal{G}}) \le s_{\max}(G) + s_{\max}(D) + s_{\max}(S)$. In particular, $s_{\max}(G) \le 2\sqrt{n}$ with probability $1 - e^{-c_1 n}$ for Gaussian random variable using Theorem 12. For the second term, again using Theorem 12, we have with probability $1 - e^{-c_1 n}$,

$$\begin{aligned} s_{\max}(D) &= \max_{e_i} |e_i^\top D e_i| = \max_{e_i} |e_i^\top G e_i| \\ &\le \max_{v \in \mathbb{S}^{n-1}} |v^\top G v| = s_{\max}(G), \end{aligned}$$

where e_i is the i-th standard basis vector. Finally, for the last term, we note that

$$s_{\max}(S) \le \max_i \left|\sum_{j \neq i} G[i,j]\right|.$$

The standard Chernoff-Hoefding bound for the sum of random Gaussian variable gives that with probability $1 - e^{-c_2 n}$, $s_{\max}(S) \le C\sqrt{n\log(n)}$. The theorem follows by using union bound. □

A direct corollary of the above theorem is as follows:

Corollary 1. *Let $\mathcal{G}$ be a complete graph on n vertices with edge weights identical and independent copies of $\mathcal{N}(0, \sigma^2)$. Then*

$$\mathsf{Pr}\left[s_{\max}(L_{\mathcal{G}}) \le C\sigma\sqrt{n\log(n)}\right] \ge 1 - 3e^{-cn}$$

for some constants $C, c > 0$.

In our application, we would use the above corollary.

Remark 2. *Using Borel Cantelli's theorem, it is possible to achieve a tighter bound on $s_{\max}(D)$ than what is claimed above (see Exercise 2.5.10 in Vershynin (2018)). However, for our purpose, the above bound suffices.*

D Applications

In this section, we give various applications of our privacy theorem. In Section D.1, we show some applications in performing differentially private analysis on graphs. In Section D.4, we show its application in signal recovery.

D.1 Differentially Private Graph Analysis

Let $\|A\|_2$ denote the spectral norm of a matrix A. We utilize the fact that $\|L_{\mathcal{R}}\|_2 = O(\sigma\sqrt{n\log(n)})$ if the edge on the graph $\mathcal{R}$ is sampled i.i.d. from $\mathcal{N}(0,\sigma^2)$ for various differentially private analysis. In particular, this implies that

$$L_{\mathcal{R}} \preceq O\left(\sigma\sqrt{\frac{\log(n)}{n}}\right) L_{K_n}; -L_{\mathcal{R}} \preceq O\left(\sigma\sqrt{\frac{\log(n)}{n}}\right) L_{K_n}. \tag{6}$$

This is due to the fact that $L_{\mathcal{R}}$ and L_{K_n} commute with each other and hence have a spectral decomposition with same unitary matrix U. Since $L_{\mathcal{R}}e = L_{K_n}e = 0$ and rest of the eigenvalues of L_{K_n} are n, it holds that

$$L_{\mathcal{R}} \preceq \frac{\|L_{\mathcal{R}}\|_2}{n} L_{K_n} \qquad \text{and} \qquad -L_{\mathcal{R}} \preceq \frac{\|L_{\mathcal{R}}\|_2}{n} L_{K_n}.$$

Since differential privacy is preserved under arbitrary post-processing Dwork and Roth (2014), this result can be used in many graph problems. In what follows, we focus on a few key examples of the application of our results. In particular, we show its application to compute differentially private approximation to cut related tasks, learning functions on graphs, etc.

We first state the following about Priv-Graph.

Theorem 14 (Differential privacy of Priv-Graph)**.** *Let $\mathcal{G}$ be an n vertex graph. Then $\widetilde{\mathcal{G}} \leftarrow$ Priv-Graph$(\mathcal{G};(\epsilon,\delta))$ satisfy (ϵ,δ)-differential privacy.*

Proof. Let $\mathcal{G}$ and $\mathcal{G}'$ be two neighboring graphs that differs in exactly one edge $e := (i,j)$ by a weight $\gamma \in (0,1]$. Let vec $(A_{\mathcal{G}})$ and vec $(A_{\mathcal{G}'})$ be the vectorization of adjacency matrices of $\mathcal{G}$ and $\mathcal{G}'$, respectively. Then

$$\text{vec}\,(A_{\mathcal{G}}) - \text{vec}\,(A_{\mathcal{G}'}) = \gamma(\bar{e}_i \otimes \bar{e}_j + \bar{e}_j \otimes \bar{e}_i).$$

It follows from above that $\|\text{vec}\,(A_{\mathcal{G}}) - \text{vec}\,(A_{\mathcal{G}'})\|_2 = \gamma\sqrt{2}$. The proof of the theorem follows by invoking Theorem 8 with $\sigma = \frac{\gamma}{\epsilon}\sqrt{2\log(1/\delta)}$. □

Algorithm 7 Priv-Graph $(\mathcal{G};(\epsilon,\delta))$

Input: An input graph $\mathcal{G} = (V,E)$, privacy parameter (ϵ,δ).
Output: A laplacian of graph $L_{\widetilde{\mathcal{G}}}$.

1: **Set** $\sigma := \frac{4\sqrt{\log(1/\delta)}}{\epsilon}$.
2: **Sample** $g_{ij} \sim \mathcal{N}(0,\sigma^2)$ for $1 \le i < j \le n$.
3: **Define** a matrix $L_{\mathcal{R}} \in \mathbb{R}^{n\times n}$ such that

$$L_{\mathcal{R}}[i,j] := \begin{cases} g_{ij} & i < j \\ g_{ji} & j < i \\ -\sum_{k\neq i} g_{ik} & i = j. \end{cases}$$

4: **Compute** the laplacian, $L_{\widetilde{\mathcal{G}}} = L_{\mathcal{G}} + L_{\mathcal{R}}$.
5: **Output:** $\widetilde{\mathcal{G}}$.

Differentially private cut queries. Cut queries are arguably one of the most widely studied graph problems Blocki et al. (2012); Gupta et al. (2012, 2013); Hardt and Rothblum (2010). Let us consider the case when we are given a query of the form $(S, V\backslash S)$. Recall that for a cut query $S \subset V$ on a graph $\mathcal{G}$, we denote by $\Phi_S(\mathcal{G})$ as the size of the cut, i.e.,

$$\Phi_S(\mathcal{G}) = \sum_{u\in S, v\in V\backslash S} w(u,v),$$

where $w_{u,v}$ is the weight on the edge between nodes $u \in V$ and $v \in V$. We have the following result:

Theorem 15 ($(S, V\backslash S)$-cut)**.** *Let $\mathcal{G}$ be the input graph. Then* Priv-Graph *is an (ϵ, δ)-differentially private algorithm such that, simultaneously for all $S \subset V$, the output $\widetilde{\mathcal{G}} \leftarrow$ Priv-Graph $(\mathcal{G}; (\epsilon, \delta))$ satisfy:*

$$\left|\Phi_S(\widetilde{\mathcal{G}}) - \Phi_S(\mathcal{G})\right| \leq O\left(\frac{\sqrt{n\log(n)\log(1/\delta)}|S|}{\epsilon}\right)$$

with probability $2/3$.

There is nothing special about success probability of $2/3$. Using standard techniques, one can easily enhance it to $1-\beta$ for arbitrary small $\beta > 0$ while incurring accuracy loss by a factor of $\log(1/\beta)$ more that what is stated in Theorem 15.

We now proceed to contrast the claim of the above theorem with existing results. The current best known ϵ-differentially private algorithm by Gupta et al. (2012) takes as input a graph and outputs a graph $\widetilde{\mathcal{G}}$ via their iterative construction framework (and appropriate linear program) to show the following guarantee:

$$\left|\Phi_S(\widetilde{\mathcal{G}}) - \Phi_S(\mathcal{G})\right| = \widetilde{O}\left(\frac{n^{3/2}}{\epsilon}\right).$$

Our result improve upon their guarantee whenever $|S| = o(n)$. Moreover, our algorithm is arguably much simpler.

In the regime of sublinear additive error in n on weighted graphs, the best known differentially private algorithm is due to Blocki et al. (2012). They achieve the following guarantee:

$$|\mathsf{est}_S - \Phi_S(\mathcal{G})| \leq \alpha\cdot\Phi_S(\mathcal{G}) + O\left(\frac{\sqrt{n\log^3(1/\delta)}|S|}{\alpha^2\epsilon}\right)$$

for a small constant $\alpha > 0$. If the size of the cut is large, i.e., whenever $\Phi_S(\mathcal{G}) \gg \widetilde{\Omega}\left(\frac{\sqrt{n}}{\epsilon}\right)$, the contribution of $\Phi_S(\mathcal{G})$ on estimation error can be very large. Our result improves upon their guarantee by removing the dependence on cut size and the additive accuracy by $\frac{1}{\alpha^2}\log(1/\delta)$.

Finally, the algorithm of Dwork et al. (2014) output a matrix $C = L_{\mathcal{G}} + N$, where N is a Gaussian matrix with appropriate noise to preserve differential privacy and $L_{\mathcal{G}}$ denotes the Laplacian of the input graph, $\mathcal{G}$. For a set of q cut queries $(S, V\backslash S)$, standard concentration bounds on Gaussian distribution implies that their algporithm incur an additive error of order $O(\frac{|S|}{\epsilon}\sqrt{\log(q)})$. In particular, if we wish to answer all possible cut queries, it leads to $O(\frac{|S|}{\epsilon}\sqrt{n})$ additive error, asymptotically same as ours. However, as we will shortly see, their result cannot answer the (S, T) cut query when $T \neq V\backslash S$.

Proof of Theorem 15. First of all, we note that the proof of differential privacy follows from Lemma 1 and Theorem 14. It remains to prove the accuracy guarantee of the theorem. By definition, the matrix $L_{\mathcal{R}}$ in Priv-Graph is symmetric and satisfies $\bar{e}^\top L_{\mathcal{R}}\bar{e} = 0$. Moreover, Theorem 23 implies that $\|L_{\mathcal{R}}\|_2 = O\left(\sigma\sqrt{n\log(n)}\right)$ with high probability. For a subset $S \subseteq [n]$, let

$$\chi_S = \sum_{i\in S}\bar{e}_i.$$

It is known that for any graph $\widetilde{\mathcal{G}}$, $\Phi_S(\widetilde{\mathcal{G}}) = \chi_S^\top L_{\widetilde{\mathcal{G}}}\chi_S$. It follows that

$$\begin{aligned}\Phi_S(\widetilde{\mathcal{G}}) &= \chi_S^\top L_{\widetilde{\mathcal{G}}}\chi_S = \chi_S^\top L_{\mathcal{G}}\chi_S + \chi_S^\top L_{\mathcal{R}}\chi_S \\ &= \Phi_S(\mathcal{G}) + \chi_S^\top L_{\mathcal{R}}\chi_S\end{aligned}$$

and hence $\left|\Phi_S(\widetilde{\mathcal{G}}) - \Phi_S(\mathcal{G})\right| = \left|\chi_S^\top L_{\mathcal{R}}\chi_S\right|$. It follows from Equation (6) that

$$\begin{aligned}\left|\chi_S^\top L_{\mathcal{R}}\chi_S\right| &= O\left(\sigma\sqrt{\frac{\log(n)}{n}}\right)\left|\chi_S^\top L_{K_n}\chi_S\right| \\ &= O\left(\frac{\sigma\sqrt{\log(n)}|S|\,(n-|S|)}{\sqrt{n}}\right) \\ &= O\left(\sigma\sqrt{n\log(n)}|S|\right).\end{aligned}$$

Setting the value of σ gives the desired result as stated in Theorem 15. □

Theorem 15 can be extended to general (S,T)-cut queries, when T is not necessarily $V\backslash S$. We restate the theorem for cut queries for the ease of readers.

Theorem 16 ((S,T)-cut)**.** *Let $\mathcal{G}$ be the input graph. Then* Priv-Graph *is an (ϵ,δ)-differentially private algorithm such that for any $S \subset V$, the output $\widetilde{\mathcal{G}} \leftarrow$ Priv-Graph $(\mathcal{G};(\epsilon,\delta))$ satisfy:*

$$\left|\Phi_{S,T}(\widetilde{\mathcal{G}}) - \Phi_{S,T}(\mathcal{G})\right| \leq O\left(\frac{\sqrt{\log(n)\log(1/\delta)}|S||T|}{\epsilon\sqrt{|S|+|T|}}\right).$$

Proof. As in the case of Theorem 15, the privacy proof follows from Lemma 1 and Theorem 14. Consider the modified graph $\mathcal{G}'$ with vertex and edge sets

$$V' = S \cup T \qquad \text{and} \qquad E' = \{(u,v) : u,v \in S \cup T\},$$

respectively. Let $L_{\mathcal{R}'}$ denote the submatrix of $L_{\mathcal{R}}$ where rows and columns are indexed from $S \cup T$ such that the diagonal entries of $L_{\mathcal{R}'}$ satisfies the following relationship:

$$L_{\mathcal{R}'}[i,i] = -\sum_{j \in S\cup T : j \neq i} L_{\mathcal{R}'}[i,j].$$

Let $\widetilde{\mathcal{G}}'$ be the weighted graph obtained from the output $\widetilde{\mathcal{G}} \leftarrow$ Priv-Graph $(\mathcal{G};(\epsilon,\delta))$ with vertex and edge sets coinciding with those of $\mathcal{G}'$. Since the non-diagonal entries of $L_{\mathcal{R}'}$ are identical and independent copies of $\mathcal{N}(0,\sigma^2)$, we can use Theorem 23 to conclude that $\|L_{\mathcal{R}'}\| = O(m\log m)$ where $m = |S\cup T|$. Moreover, (S,T)-cut on graph $\mathcal{G}$ is equivalent to $(S,V'\backslash S)$-cut on subgraph $\mathcal{G}'$. Hence, from Theorem 15, we have that

$$\begin{aligned}\left|\Phi_{S,T}(\widetilde{\mathcal{G}}) - \Phi_{S,T}(\mathcal{G})\right| &= \left|\Phi_S(\widetilde{\mathcal{G}}') - \Phi_S(\mathcal{G}')\right| \\ &= O\left(\frac{\sigma\sqrt{\log m}|S|\,(m-|S|)}{\sqrt{m}}\right) \\ &\leq O\left(\frac{\sigma\sqrt{\log(n)}|S||T|}{\sqrt{|S|+|T|}}\right) \\ &= O\left(\frac{\sqrt{\log(n)\log(1/\delta)}|S||T|}{\epsilon\sqrt{|S|+|T|}}\right).\end{aligned}$$

This completes the proof of Theorem 16. □

To the best of our knowledge, the only other efficient algorithm that answers general (S,T) cut queries is due to Gupta et al. (2012). Their algorithm gives an additive error $O(\sqrt{n|S|\cdot|T|}/\epsilon)$. That is, we achieve better accuracy guarantee whenever either $|S|$ or $|T|$ is not a constant factor of n.

Algorithm 8 PRIV-SPARSIFY $(\mathcal{G}; (\epsilon, \delta))$

Input: An input graph $\mathcal{G} = (V, E)$, privacy parameter (ϵ, δ), non private algorithm for spectral sparsification of a graph, SPARSIFY$(\cdot)$.

Output: A sparse graph, $\widetilde{\mathcal{G}}$.

1: **Compute** $\widehat{\mathcal{G}} \leftarrow$ PRIV-GRAPH $(\mathcal{G}; (\epsilon, \delta))$. Let $\left\|L_{\widehat{\mathcal{G}}} - L_{\mathcal{G}}\right\|_2 = c\sigma\sqrt{n\log(n)}$ for some constant c and $\sigma = \frac{4}{\epsilon}\sqrt{\log(1/\delta)}$.

2: **Compute** $L_{\widetilde{\mathcal{H}}} \leftarrow L_{\widehat{\mathcal{G}}} + \sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n}$ for a positive constant $C > 3c$.

3: **Solve** the following semidefinite program to obtain the optimal solution pair $\left(\bar{\gamma}, \bar{L}_{\mathcal{G}'}\right)$:

$$\begin{aligned} \text{minimize: } & \gamma \\ \text{subject to: } & L_{\widetilde{\mathcal{H}}} - L_{\mathcal{G}'} \preceq \gamma \mathbb{1}_n, \\ & L_{\mathcal{G}'} - L_{\widetilde{\mathcal{H}}} \preceq \gamma \mathbb{1}_n, \\ & \gamma \geq 0, \\ & L_{\mathcal{G}'} \in \mathbb{L}^n(1/n). \end{aligned}$$

4: **Construct** $\bar{\mathcal{G}}$ from $\bar{L}_{\mathcal{G}'}$ by setting weights for each edge (u, v) of $\bar{\mathcal{G}}$ as $-\bar{L}_{\mathcal{G}'}[u, v]$.

5: **Output** $\widetilde{\mathcal{G}} \leftarrow$ SPARSIFY $(\bar{\mathcal{G}})$.

D.2 Differentially Private spectral sparsification of graphs

Spectral sparsification of graphs is a widely studied algorithmic problem. Roughly speaking, given an input (possibly) weighted graph $\mathcal{G}$, a spectral sparsification of $\mathcal{G}$ is a weighted sparse subgraph (weights are allowed to be different) $\mathcal{H}$ such that the spectra of $\mathcal{G}$ and $\mathcal{H}$ are almost same. That is,

$$(1-\rho)x^\top L_{\mathcal{G}} x \leq x^\top L_{\mathcal{H}} x \leq (1+\rho)x^\top L_{\mathcal{G}} x \tag{7}$$

for all $x \in \mathbb{R}^n$, where $L_{\mathcal{G}}$ and $L_{\mathcal{H}}$ are Laplacian matrices of $\mathcal{G}$ and $\mathcal{H}$, respectively. The above relationship implies that the spectra of output graph lies within $[1-\rho, 1+\rho]$ of the input graph. One should note that when x is restricted to be 0/1 vector, then the relationship describes the cut sparsification of graph.

Right after its inception, spectral sparsification of graph has turned out be one of the fundamental notions with wide range of applications, including but not limited to Laplacian solvers, learning Lipschitz functions on graphs, etc. Given its importance, it is imperative to study spectral sparsification from differential privacy perspective. It turns out that one cannot hope to come up with a differentially private algorithm for spectral sparsification of graph satisfying Equation (7) because the output itself reveals information about $\widetilde{O}\left(n/\rho^2\right)$ edges of the original graph.

We give a brief overview of the differentially private algorithm for spectral sparsification below. The first step of the algorithm calls PRIV-GRAPH $(\mathcal{G}; (\epsilon, \delta))$ to output $\widehat{\mathcal{G}}$. This step is already differentially private. Since differential privacy is preserved under arbitrary post-processing, one might think that one can run existing non-private algorithm for spectral sparsification on the output of PRIV-GRAPH. In particular, one might consider using any existing algorithms for spectral sparsification Allen-Zhu et al. (2015); Lee and Sun (2015); Spielman and Srivastava (2011); Spielman and Teng (2011). Any such algorithm (denoted SPARSIFY in Algorithm 8) takes as input a graph $\mathcal{H}$ and output a graph $\widetilde{\mathcal{H}}$ such that $\widetilde{\mathcal{H}}$ contains only $O\left(n/\rho^2\right)$ edges and approximates the spectrum of $\mathcal{H}$.

This approach is not going to work because of following reasons.

1. The output graph $\widehat{\mathcal{G}}$ will potentially have negative weights as the edge weights of this graph is sum of edge weights of original graph and a random weight drawn from $\mathcal{N}(0, \sigma^2)$.
2. The Laplacian of $\widehat{\mathcal{G}}$ may not be positive semidefinite (because of potentially negative weights).

These two properties is required to hold for existing spectral sparsification algorithms to work.

We first ensure that the Laplacian is positive semidefinite. To this end, we overlay a uniformly weighted complete graph on $\bar{\mathcal{G}}$. The weight is chosen such that the Laplacian of graph after overlaying (denoted $\widetilde{\mathcal{H}}$ in Algorithm 8) is positive semidefinite. This is accomplished in Step 2 of the algorithm PRIV-SPARSIFY $(\mathcal{G}; (\epsilon, \delta))$. Note that

$$\begin{aligned} L_{\widetilde{\mathcal{H}}} &= L_{\widehat{\mathcal{G}}} + \sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n} \\ &= L_{\mathcal{G}} + \left(L_{\mathcal{R}} + \sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n} \right). \end{aligned}$$

Since $L_{\mathcal{G}} \succeq 0$ and $\sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n} + L_{\mathcal{R}} \succeq 0$, it follows that $L_{\widetilde{\mathcal{H}}} \succeq 0$. While the positive semidefinite property of $\widetilde{\mathcal{H}}$ is satisfied, the overlaying weights of complete graph are not sufficiently large to ensure that weights of $\widetilde{\mathcal{H}}$ is non-negative. Therefore, we still cannot use the algorithm SPARSIFY.

The issue of negative weights is resolved by solving a semidefinite program described in Step 3. Recall from Equation (5) that $\mathbb{L}^n(\varrho)$ can be expressed as an intersection of cone of positive semidefinite matrices and set of affine equalities and inequalities. That is,

$$\mathbb{L}^n(\varrho) = \left\{ X : X[i,j] \leq 0 \quad \forall i,j \in [n] : i \neq j \quad \text{and} \quad 0 \leq \sum_{j=1}^{n} X[i,j] \leq \varrho \quad \forall i \in [n] \right\}.$$

Hence the optimization problem described in Step 3 is truly a semidefinite program. Moreover, the input graph $\mathcal{G}$ is already a feasible solution of the SDP with objective value $c\sigma\sqrt{n\log(n)}$ and hence $\gamma \leq c\sigma\sqrt{n\log(n)}$. Given the optimal solution $\bar{L}_{\mathcal{G}'}$ of the SDP, we have

$$\begin{aligned} \left\| \bar{L}_{\mathcal{G}'} - L_{\mathcal{G}} \right\|_2 &= \left\| \bar{L}_{\mathcal{G}'} - L_{\widetilde{\mathcal{H}}} + L_{\widetilde{\mathcal{H}}} - L_{\mathcal{G}} \right\|_2 \\ &\leq \left\| \bar{L}_{\mathcal{G}'} - L_{\widetilde{\mathcal{H}}} \right\|_2 + \left\| L_{\widetilde{\mathcal{H}}} - L_{\mathcal{G}} \right\|_2 \\ &\leq \gamma + c\sigma\sqrt{n\log(n)}. \end{aligned}$$

Hence, $\left\| \bar{L}_{\mathcal{G}'} - L_{\mathcal{G}} \right\|_2 \leq 2c\sigma\sqrt{n\log(n)}$. Finally, we ensure that the optimal solution of the SDP exists and attained as well. This is done for the sake of rigor and we defer to Lemma 5 in Section F for a proof. Step 4 constructs the actual graph from $\bar{L}_{\mathcal{G}'}$. Simple algebra shows that $\left\| L_{\bar{\mathcal{G}}} - \bar{L}_{\mathcal{G}'} \right\|_2 \leq \varrho = 1/n$ (where $L_{\bar{\mathcal{G}}}$ is the Laplacian of $\bar{\mathcal{G}}$), and hence

$$\left\| L_{\bar{\mathcal{G}}} - L_{\mathcal{G}} \right\|_2 \leq O\left(\sigma\sqrt{n\log(n)} \right).$$

One may ask why we do not optimize over $\mathbb{L}^n$ (refer to Equation (4) for definition) in Step 3. While we can indeed optimize over $\mathbb{L}^n$ by working over a smaller subspace, optimizing over $\mathbb{L}^n(\varrho)$ allows us to employ SDP algorithms (such as ellipsoid method or interior-point method based on central path) that requires existence of feasible solutions that belong to interior of cone of PSD matrices. Moreover, we do not gain any run-time performance by optimizing over $\mathbb{L}^n$.

We restate our result for spectral sparsification of graphs in a more formal manner next.

Theorem 17 (Differentially private spectral sparsification)**.** *Let $\mathcal{G}$ be the input graph on n vertices. Let L_{K_n} be the Laplacian of the unweighted complete graph. Then* PRIV-SPARSIFY$(\mathcal{G}; (\epsilon, \delta))$ *is a polynomial-time (ϵ, δ)-differentially private algorithm with respect to presence or absence of an edge. Moreover, $\widetilde{\mathcal{G}} \leftarrow$*PRIV-SPARSIFY$(\mathcal{G}; (\epsilon, \delta))$ *has $O(n/\rho^2)$ edges such that, with probability at least $9/10$,*

$$(1-\rho)\left(L_{\mathcal{G}} + \sqrt{c_1\alpha} L_{K_n}\right) \preceq L_{\bar{\mathcal{G}}} \preceq (1+\rho)\left(L_{\mathcal{G}} + \sqrt{c_2\alpha} L_{K_n}\right) \quad \text{where} \quad \alpha := \frac{\log(n)\log(1/\delta)}{n\epsilon^2}.$$

Here c_1, c_2 are absolute positive constants with $c_1 < 2C < c_2$.

Proof. We will analyze the algorithm backwards to prove the theorem. First of all, by spectral sparsification guarantee, Step 5 ensures that

$$(1-\rho) L_{\bar{\mathcal{G}}} \preceq L_{\widetilde{\mathcal{G}}} \preceq (1+\rho) L_{\bar{\mathcal{G}}}. \tag{8}$$

Since $\left\|L_{\bar{\mathcal{G}}} - \bar{L}_{\mathcal{G}'}\right\|_2 \leq \varrho$ (Step 4), it follows that $\bar{L}_{\mathcal{G}'} - \varrho\mathbb{1}_n \preceq L_{\bar{\mathcal{G}}} \preceq \bar{L}_{\mathcal{G}'} + \varrho\mathbb{1}_n$. Step 3 ensures that

$$L_{\widetilde{\mathcal{H}}} - \bar{\gamma}\mathbb{1}_n \preceq L_{\bar{\mathcal{G}}} \preceq L_{\widetilde{\mathcal{H}}} + \bar{\gamma}\mathbb{1}_n.$$

and hence

$$L_{\widetilde{\mathcal{H}}} - (\bar{\gamma} - \varrho)\mathbb{1}_n \preceq L_{\bar{\mathcal{G}}} \preceq L_{\widetilde{\mathcal{H}}} + (\bar{\gamma} + \varrho)\mathbb{1}_n$$

for $\bar{\gamma} \leq c\sigma\sqrt{n\log(n)}$. Now note that $\bar{e}^\top \left(aL_{\widetilde{H}} + bL_{\bar{\mathcal{G}}}\right)\bar{e} = 0$ for any choice of $a, b \in \mathbb{R}$, where $\bar{e}$ denotes the vector of all 1's as mentioned in Appendix B. This allows us to replace $\mathbb{1}_n$ by L_{K_n}/n:

$$L_{\widetilde{\mathcal{H}}} - \frac{\bar{\gamma} - \varrho}{n}L_{K_n} \preceq L_{\bar{\mathcal{G}}} \qquad \text{and} \qquad L_{\bar{\mathcal{G}}} \preceq L_{\widetilde{\mathcal{H}}} + \frac{\bar{\gamma} + \varrho}{n}L_{K_n}.$$

From Steps 1 and 2, we have the following relationship between $L_{\widetilde{\mathcal{H}}}$ and $L_{\mathcal{G}}$.

$$L_{\widetilde{\mathcal{H}}} = L_{\mathcal{G}} + L_{\mathcal{R}} + \sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}}L_{K_n}.$$

Here $L_{\mathcal{R}}$ is the Laplacian formed by Gaussian random variables as defined in Algorithm 7. We already know from our analysis that $L_{\mathcal{R}}$ satisfies $-c\sigma\sqrt{\frac{\log(n)}{n}}L_{K_n} \preceq L_{\mathcal{R}} \preceq c\sigma\sqrt{\frac{\log(n)}{n}}L_{K_n}$. A straightforward calculation yields the following two PSD inequalities as described in the theorem statement:

$$L_{\mathcal{G}} + \sqrt{c_1\alpha}L_{K_n} \preceq L_{\bar{\mathcal{G}}} \preceq L_{\mathcal{G}} + \sqrt{c_2\alpha}L_{K_n}$$

where $c_1 := (2C - 2c - 1)$ and $c_2 := (2C + 2c + 1)$. Here we use the fact that $\bar{\gamma} \leq c\sigma\sqrt{n\log(n)}$ and $\varrho < \sigma\sqrt{n\log(n)}$. Substituting the above relationship for $L_{\bar{\mathcal{G}}}$ in Equation (8) finishes the proof of Theorem 17. □

As a direct application of this result, we can get the same additive error as in Theorem 15 and 18 at the expense of a small multiplicative approximation. On the other side, the space and time requirement of the algorithm improves significantly. This analysis is pretty straightforward and therefore omitted.

D.3 Differentially private combinatorial optimization

Gupta et al. Gupta et al. (2010) initiated the study of differentially private combinatorial optimization. In particular, they gave a private algorithm for Min-Cut. We extend their work to several NP-hard graph problems such as Max-Cut, Sparsest-Cut, and Edge-Expansion.

We next state our result for these combinatorial problems. We have to be little careful in applying our privacy mechanism to estimate Max-Cut and Sparsest-Cut. This is because existing algorithms for these problems assume that the graph has non-negative weights. To assure that, with high probability, all the weights stays non-negative after overlaying a complete graph with weights sampled from Gaussian distribution, we use the same technique as in the case of spectral sparsification: solve an appropriate semi-definite program. Then Theorem 23 allows us to prove the following result:

Theorem 18 (Differentially private combinatorial optimization)**.** *Let $\mathcal{G}$ be the input graph. Then* Priv-Comb-Opt $(\mathcal{G}; (\epsilon, \delta); \mathsf{flag})$ *is a polynomial-time (ϵ, δ)-differentially private algorithm. Further* Priv-Comb-Opt $(\mathcal{G}; (\epsilon, \delta); \mathsf{flag})$ *outputs a set of nodes $\bar{S}$ that approximates* Max-Cut *and* Sparsest-Cut *depending on the value of Boolean variable* flag *with the following guarantees:*

1. *When* $\mathsf{flag} = 0$*, then the algorithm,* Priv-Comb-Opt $(\mathcal{G}; (\epsilon, \delta); 0)$*, approximates* Max-Cut *with following guarantee:*

$$\Phi_{\bar{S}}(\mathcal{G}) \geq (0.87 - \rho)\mathsf{OPT}_{\mathsf{max}}(\mathcal{G}) - O\left(\frac{\sqrt{n\log(n)\log(1/\delta)}\,|\bar{S}|}{\epsilon}\right).$$

Algorithm 9 PRIV-COMB-OPT $(\mathcal{G}; (\epsilon, \delta); \mathsf{flag})$

Input: An input graph $\mathcal{G} = (V, E)$, privacy parameter (ϵ, δ), flag : 0 for MAX-CUT and 1 for SPARSEST-CUT, algorithms MAX$(\cdot)$ and SPARSEST$(\cdot)$.
Output: A partition of nodes, S.

1: **Compute** $\widehat{\mathcal{G}} \leftarrow$ PRIV-GRAPH $(\mathcal{G}; (\epsilon, \delta))$. Let $\left\|L_{\widehat{\mathcal{G}}} - L_{\mathcal{G}}\right\|_2 = c\sigma\sqrt{n\log(n)}$ for some constant c and $\sigma = \frac{4}{\epsilon}\sqrt{\log(1/\delta)}$.
2: **Compute** $L_{\widetilde{\mathcal{H}}} \leftarrow L_{\widehat{\mathcal{G}}} + \sqrt{\frac{2C\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n}$ for a positive constant $C > 3c$.
3: **Solve** the following semidefinite program for $\varrho := 1/n$ to obtain the optimal solution pair $\left(\bar{\gamma}, \bar{L}_{\mathcal{G}'}\right)$:

$$\begin{aligned} \text{minimize: } & \gamma \\ \text{subject to: } & L_{\widetilde{\mathcal{H}}} - L_{\mathcal{G}'} \preceq \gamma \mathbb{1}_n, \\ & L_{\mathcal{G}'} - L_{\widetilde{\mathcal{H}}} \preceq \gamma \mathbb{1}_n, \\ & \gamma \geq 0, \\ & L_{\mathcal{G}'} \in \mathbb{L}^n(\varrho). \end{aligned}$$

4: **Construct** $\bar{\mathcal{G}}$ from $\bar{L}_{\mathcal{G}'}$ by setting weights for each edge (i, j) of $\bar{\mathcal{G}}$ as $-\bar{L}_{\mathcal{G}'}[i, j]$.
5: **if** $\mathsf{flag} = 0$ **then**
6: $\quad \bar{S} \leftarrow$ MAX$(\bar{\mathcal{G}})$
7: **else**
8: $\quad \bar{S} \leftarrow$ SPARSEST$(\bar{\mathcal{G}})$.
9: **end if**
10: **Output:** $\bar{S}$.

2. *When* $\mathsf{flag} = 1$*, then the algorithm,* PRIV-COMB-OPT $(\mathcal{G}; (\epsilon, \delta); 1)$*, approximates* SPARSEST-CUT *with following guarantee:*

$$\frac{\Phi_{\bar{S}}(\mathcal{G})}{|\bar{S}|\left(n - |\bar{S}|\right)} \leq O(\sqrt{\log(n)})\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}) + O\left(\sqrt{\frac{\log^2 n \log(1/\delta)}{\epsilon^2 n}}\right).$$

Here $\mathsf{OPT}_{\mathsf{max}}(\mathcal{G})$ *and* $\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G})$ *are the optimum value of* MAX-CUT *and* SPARSEST-CUT *of* $\mathcal{G}$*, respectively.*

Proof. We note that Algorithm 8 and Algorithm 9 are exactly same until Step 4. Hence, the relationship between $L_{\bar{\mathcal{G}}}$ and $L_{\mathcal{G}}$ presented in Theorem 17 holds here too:

$$L_{\mathcal{G}} + \sqrt{\frac{c_1\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n} \preceq L_{\bar{\mathcal{G}}} \preceq L_{\mathcal{G}} + \sqrt{\frac{c_2\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n}. \tag{9}$$

For any set $S \subseteq [n]$, let χ_S denotes its indicator vector:

$$\chi_S = \sum_{i \in S} \bar{e}_i.$$

Assume the Boolean variable flag is set to 0. In this case, Algorithm 9 aims to privately approximate MAX-CUT. Let S be the vertex set that induce the maximum cut in $\mathcal{G}$. The approximation guarantee for MAX-CUT (refer to Theorem 9) and Step 6 gives the following:

$$\Phi_{\bar{S}}\left(\bar{\mathcal{G}}\right) \geq (\alpha_{\mathsf{GW}} - \rho)\chi_{\bar{S}}^\top L_{\bar{\mathcal{G}}}\chi_{\bar{S}} \geq (\alpha_{\mathsf{GW}} - \rho)\chi_S^\top L_{\bar{\mathcal{G}}}\chi_S, \tag{10}$$

where the second inequality is due to optimality. Using the first semidefinite inequality of Equation (9), we have $\chi_S^\top L_{\bar{\mathcal{G}}}\chi_S$ is lower bounded by

$$\chi_S^\top \left(L_{\mathcal{G}} + \sqrt{\frac{c_1\log(n)\log(1/\delta)}{n\epsilon^2}} L_{K_n}\right)\chi_S \geq \chi_S^\top L_{\mathcal{G}}\chi_S = \mathsf{OPT}_{\mathsf{max}}(\mathcal{G}).$$

Here we have used the fact that $\chi_S^\top L_{K_n} \chi_S \geq 0$. Using the above lower bound for $\chi_S^\top L_{\bar{\mathcal{G}}} \chi_S$ in Equation (10), we obtain

$$\Phi_{\bar{S}}\left(\bar{\mathcal{G}}\right) \geq (\alpha_{\mathsf{GW}} - \rho)\mathsf{OPT}_{\mathsf{max}}(\mathcal{G}).$$

Moreover, the second semidefinite inequality of Equation (9) gives the following chain of inequalities:

$$\begin{aligned}
\Phi_{\bar{S}}(\bar{\mathcal{G}}) &\leq \chi_{\bar{S}}^\top L_{\bar{\mathcal{G}}} \chi_{\bar{S}} \\
&\leq \chi_{\bar{S}}^\top \left(L_{\mathcal{G}} + \sqrt{\frac{c_2 \log(n) \log(1/\delta)}{n\epsilon^2}} L_{K_n} \right) \chi_{\bar{S}} \\
&\leq \Phi_{\bar{S}}(\mathcal{G}) + \left(\sqrt{\frac{c_2 \log(n) \log(1/\delta)}{n\epsilon^2}} \right) n|\bar{S}|.
\end{aligned}$$

as $\left(n - |\bar{S}|\right) < n$.

Combining the lower and upper bounds for $\Phi_{\bar{S}}(\bar{\mathcal{G}})$, we get the desired relationship:

$$\Phi_{\bar{S}}(\mathcal{G}) \geq (\alpha_{\mathsf{GW}} - \rho)\mathsf{OPT}_{\mathsf{max}}(\mathcal{G}) - O\left(\frac{\sqrt{n \log(n) \log(1/\delta)}\, |\bar{S}|}{\epsilon} \right).$$

This completes part 1 of Theorem 18.

Now assume that the Boolean variable flag is set to 1. In this case, Algorithm 9 aims to privately approximate Sparsest-Cut. Recall that Sparsest-Cut for a graph $\mathcal{G}$ over vertex set $[n]$ asks for a $(S, V \backslash S)$-cut that minimizes

$$\min_{S \subseteq [n]} \left\{ \frac{\Phi_S(\mathcal{G})}{|S|(n - |S|)} \right\}.$$

Let S be the vertex set that induce the sparsest cut in $\mathcal{G}$. Then we have

$$\mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}) := \frac{\Phi_S(\mathcal{G})}{|S|(n - |S|)}$$

Also, by approximation guarantee of Sparsest-Cut as mentioned in Theorem 10, we have

$$\begin{aligned}
\frac{\Phi_{\bar{S}}(\bar{\mathcal{G}})}{|\bar{S}|(n - |\bar{S}|)} &\leq O\left(\sqrt{\log(n)}\right) \min_{S' \subseteq [n]} \left\{ \frac{\Phi_{S'}\left(\bar{\mathcal{G}}\right)}{|S'|\left(n - |S'|\right)} \right\} && (11) \\
&\leq O\left(\sqrt{\log(n)}\right) \frac{\Phi_S(\bar{\mathcal{G}})}{|S|\left(n - |S|\right)}. && (12)
\end{aligned}$$

From the second semidefinite inequality of Equation (9), one can conclude that

$$\begin{aligned}
\Phi_S(\bar{\mathcal{G}}) &= \chi_S^\top L_{\bar{\mathcal{G}}} \chi_S \\
&\leq \chi_S^\top \left(L_{\mathcal{G}} + \sqrt{\frac{c_2 \log(n) \log(1/\delta)}{n\epsilon^2}} L_{K_n} \right) \chi_S \\
&= \Phi_S(\mathcal{G}) + \left(\sqrt{\frac{c_2 \log(n) \log(1/\delta)}{n\epsilon^2}} \right) |S|(n - |S|),
\end{aligned}$$

and therefore

$$\frac{\Phi_S(\bar{\mathcal{G}})}{|S|\left(n - |S|\right)} \leq \frac{\Phi_S(\mathcal{G})}{|S|\left(n - |S|\right)} + \left(\sqrt{\frac{c_2 \log(n) \log(1/\delta)}{n\epsilon^2}} \right).$$

Since S induces the sparsest cut in $\mathcal{G}$, we obtain the following from Equation (11) and above:

$$\frac{\Phi_{\bar{S}}(\bar{\mathcal{G}})}{|\bar{S}|(n-|\bar{S}|)} \leq \mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}) + O\left(\sqrt{\frac{\log^2(n)\log(1/\delta)}{n\epsilon^2}}\right).$$

It is clear from the first semidefinite inequality of Equation (9) that $\Phi_{\bar{S}}(\mathcal{G}) \leq \Phi_{\bar{S}}(\bar{\mathcal{G}})$. Therefore

$$\frac{\Phi_{\bar{S}}(\mathcal{G})}{|\bar{S}|(n-|\bar{S}|)} \leq \frac{\Phi_{\bar{S}}(\bar{\mathcal{G}})}{|\bar{S}|(n-|\bar{S}|)}.$$

Combining the above two inequalities, we obtain the desired bound for Sparsest-Cut:

$$\frac{\Phi_{\bar{S}}(\mathcal{G})}{|\bar{S}|(n-|\bar{S}|)} \leq \mathsf{OPT}_{\mathsf{sparsest}}(\mathcal{G}) + O\left(\sqrt{\frac{\log^2(n)\log(1/\delta)}{n\epsilon^2}}\right).$$

This finishes the proof of Theorem 18. □

In contrast to previous results of Gupta et al. (2012)[3], our result gives an instance based accuracy bound.

Since $n/2 \leq |V \backslash S| \leq n$, computing the Sparsest-Cut is the same as computing the Edge-Expansion of the graph up to a factor of 2. As a result, we get the following corollary to Theorem 18:

Corollary 2 (Edge expansion). *Let $\mathcal{G}$ be the input graph. Then* Priv-Comb-Opt $(\mathcal{G}; (\epsilon, \delta); \mathsf{flag})$ *is a polynomial-time (ϵ, δ)-differentially private algorithm that, on setting* $\mathsf{flag} = 1$*, outputs a set of nodes $\bar{S}$ such that with probability* $2/3$*,*

$$\frac{\Phi_{\bar{S}}(\mathcal{G})}{|\bar{S}|} \leq O(\sqrt{\log(n)}) \min_{S \subseteq V | S| \leq n/2} \left\{\frac{\Phi_S(\mathcal{G})}{|S|}\right\} + O\left(\sqrt{\frac{\log^2 n \log(1/\delta)}{\epsilon^2 n}}\right).$$

D.4 Recovery problems

One of the important class of problems studied in the domain of recovery problems is *angular synchronization problem* Singer (2011). It is widely used in many areas of statistics. In this problem, one is required to estimate a collection of n phases $e^{i\alpha_1}, \ldots, e^{i\alpha_n}$, given noisy measurements of pairwise relative phases $e^{i(\alpha_k - \alpha_j)}$. This problem is also interesting when we work in real number instead of complex number, for example, in time-synchronization of distributed networks Karp et al. (2003), signal reconstruction Agrawal et al. (2006), correlation clustering Bansal et al. (2004), and stochastic block model Javanmard et al. (2016) The angular synchronization recovery problem is as follows:

Definition 12 (Signal recovery). *Given a noisy measurement $C = zz^\top + \sigma R$, recover $z \in \{\pm 1\}^n$.*

When R is a matrix formed by subgaussian random variable, we have the following result.

Theorem 19 (Recovery from noisy measurement). *There is an efficient algorithm that exactly recover $z \in \{\pm 1\}^n$ from noisy measurements $C = zz^\top + \sigma R$ if $\sigma = O(\sqrt{n})$ for R is a symmetric matrix with upper triangular entries independent copies of 0 mean unit variance subgaussian random variable.*

Proof. It is a well known fact that if $\|\sigma L_R\|_2 \leq n$ where R is a symmetric matrix with $L_R \bar{e} = 0$, then the corresponding signal can be exactly recovered using semidefinite programming. As before, $\bar{e}$ denotes the vector of all 1's. A straightforward application of Theorem 23 gives us that one can exactly recover the signal whenever $\sigma \leq O(\sqrt{n})$. This completes the proof of Theorem 19. □

[3] Gupta et al. (2012) do not present this algorithm, but their output can be also used to approximate Max-Cut and Sparsest-Cut.

D.5 Private graph analysis in the sliding window

One of the reasons why existing techniques do not render to the sliding window model is that they use subroutines that are not extendable to the sliding window model. For example, Gupta et al. (2012) require a linear program followed by the iterative framework and Blocki et al. (2012) uses random projections. There are no existing techniques to solve linear programs or perform the iterative construction of Gupta et al. (2012) in the sliding window model. On the other hand, random projections cannot be used in the sliding window model because it does not allow to revert the effect of the expired stream. Finally, Dwork et al. (2014) outputs a matrix that may not be a positive semi-definite and definitely not a Laplacian.

Likewise, it is not clear whether we can extend the techniques used in the sliding window model to preserve privacy. Known data-structures in sliding window consider real-valued functions[4]. Further, it is far from clear how to update the graph in the currently known techniques of the sliding window to reflect the removal of the contribution of streamed edge that has expired.

In particular, we maintain a data structure consisting of ℓ tuples $\{(\mathcal{G}_i, t_i)\}_{i=1}^{\ell}$. We require that these tuples satisfy a set of properties, which we called *smooth Laplacian property*. Recall the definition of *smooth Laplacian property*:

Definition 13 (smooth Laplacian property)**.** *A data structure $\mathfrak{D}$ satisfy* smooth Laplacian property *if there exists an $\ell = \text{poly}(n, \log(W))$ such that $\mathfrak{D}$ satisfy the following conditions*

1. $\mathfrak{D}$ consists of ℓ timestamps $\mathsf{I} := \{t_1, \ldots, t_\ell\}$ and the corresponding graphs $\mathsf{S} := \{\mathcal{G}_1, \ldots, \mathcal{G}_s\}$.
2. At least one of the following holds:
   (a) For $1 \le i \le \ell - 1$, if $t_{i+1} = t_i + 1$, then $(1-\rho)\, L_{\mathcal{G}_i} \not\preceq L_{\mathcal{G}_{i+1}}$.
   (b) Both of the following properties for $1 \le i \le \ell - 2$:
      i. $\mathsf{Property}_1$: $(1-\rho) L_{\mathcal{G}_i} \preceq L_{\mathcal{G}_{i+1}}$.
      ii. $\mathsf{Property}_2$: $(1-\rho)\, L_{\mathcal{G}_i} \not\preceq L_{\mathcal{G}_{i+2}}$.
   for some constant $0 < \rho < 1$.
3. $L_{\mathcal{G}_2} \preceq L_W \preceq L_{\mathcal{G}_1}$, where L_W is the Laplacian of the graph formed by the window W.

We restate the result for the ease of readers.

Theorem 20. *Given the privacy parameter (ϵ, δ), approximation parameter $\rho \in (0, 1/2)$, and window size W, at time T, let L_W denote the Laplacian of the graph streamed in the time epochs $[T - W + 1, T]$. Then we have the following:*

1. (Privacy guarantee). Sliding-Priv-Graph*, described in Algorithm 10, is an efficient (ϵ, δ)-differentially private algorithm under event level privacy.*
2. (Spectral guarantee) Sliding-Priv-Graph *allows us to efficiently outputs a graph $\widetilde{\mathcal{G}}$ at the end of the stream with the following guarantee:*

$$(1-\rho)\left(L_W + c_1 \beta L_{K_n}\right) \preceq L_{\widetilde{\mathcal{G}}} \preceq (1+\rho)\left(L_W + c_2 \beta L_{K_n}\right).$$

   Here $c_1 < c_2$ are large constants and L_W is the Laplacian of the graph formed by the sliding window of size W and

$$\beta := \sqrt{\frac{\log(n)\log(W/\delta)}{n\epsilon^2}}.$$

3. (Space guarantee) *The space required by* Sliding-Priv-Graph *is $O\left(\frac{n^3}{\rho}\log(W)\right)$.*

Proof. We break the proof down to two steps:

[4] Positive semi-definite ordering is a partial order; therefore, $A \not\preceq B$ does not imply that $B \preceq A$. On the other hand, $f(x) \not\le f(y)$ implies that $f(x) \ge f(y)$ – this fact is used in a non-trivial manner in the existing results.

1. It is not clear if we can perform spectral sparsification of a graph in space sublinear in W. In the first step, we prove accuracy without any private constraints, but give an efficient $O\left(\frac{n^3}{\rho}\log(W)\right)$ space algorithm.

2. In the second step, we include the privacy constraints without increasing the space requirement in the above stage.

Algorithm 10 SLIDING-PRIV-GRAPH($\mathfrak{D}_{\text{priv}}; (e_t, w_t)$)

Input: A new edge-weight pair (e_t, w_t) and $\mathfrak{D}_{\text{priv}}$ containing ℓ tuples $\left\{(\widetilde{\mathcal{G}}_1, t_1), \ldots, (\widetilde{\mathcal{G}}_\ell, t_\ell)\right\}$.
Output: Updated $\mathfrak{D}_{\text{priv}}$.

Stage 1: Include new edge (e_t, w_t) in the data structure, $\mathfrak{D}_{\text{priv}}$

if $t_2 < t - W + 1$ **then**
  For $1 \le j \le \ell - 1$, set $t_j = t_{j+1}, \widetilde{G}_j = \widetilde{G}_{j+1}$. Update $\ell \leftarrow \ell - 1$.
end if
Construct a complete graph $\mathcal{R}_{\ell+1}$ with edge weights sampled i.i.d. from $\mathcal{N}\left(0, \frac{8\log 1/\delta}{\epsilon^2}\right)$.
Privatization step. Let $\mathcal{H}_t$ be a single-edge graph with weight w_t on the edge e_t. Set $t_{\ell+1} = t$ and

$$\widetilde{\mathcal{G}}_{\ell+1} := \mathcal{H}_t \cup \mathcal{R}_{\ell+1} \cup \left(C\sqrt{\frac{\log(n)}{n\epsilon^2}\log\left(\frac{W}{\delta}\right)}K_n\right).$$

.
Update $\mathfrak{D}_{\text{priv}} \leftarrow \mathfrak{D}_{\text{priv}} \cup (\widetilde{\mathcal{G}}_{\ell+1}, t_{\ell+1})$, $\ell = \ell + 1$.
for $i = 1, \ldots, \ell - 1$ **do**
  Compute $\widetilde{\mathcal{G}}_i \leftarrow \mathcal{H}_t \cup \widetilde{\mathcal{G}}_i$. ▷ Update the graphs with new edge.
end for

Stage 2: Update $\mathfrak{D}_{\text{priv}}$ to maintain smooth Laplacian property

Mantain PSD ordering. That is, find

$$j := \min\left\{p : L_{\mathcal{G}_p} \not\preceq L_{\mathcal{G}_\ell}\right\}$$

and delete $L_{\mathcal{G}_p}, \ldots, L_{\mathcal{G}_{\ell-1}}$.
Set $L_{\mathcal{G}_p} = L_{\mathcal{G}_\ell}, \ell = p$.
for $i = 1, \ldots, \ell - 2$ **do**
  Find $\widetilde{\mathcal{G}}_{i+1}, \ldots, \widetilde{\mathcal{G}}_j$ such that

$$(1-\rho)\, L_{\widetilde{\mathcal{G}}_i} \preceq L_{\widetilde{\mathcal{G}}_j} \quad \text{and} \quad (1-\rho)\, L_{\widetilde{\mathcal{G}}_i} \not\preceq L_{\widetilde{\mathcal{G}}_{j+1}}.$$

  Delete $\widetilde{\mathcal{G}}_{i+1}, \ldots, \widetilde{\mathcal{G}}_{j-1}$ and set $k = 1$.
  while $j + k < \ell$ **do** ▷ Reorder the indices
    Update $\widetilde{\mathcal{G}}_{i+k} = \widetilde{\mathcal{G}}_{j+k-1}$, $t_{i+k} = t_{j+k-1}$
    Update $k := k + 1$.
  end while
  $\ell := \ell + i - j + 1$. ▷ Update the number of checkpoints.
end for
Output: $\mathfrak{D}_{\text{priv}} := \left\{(\widetilde{\mathcal{G}}_1, t_1), \ldots, (\widetilde{\mathcal{G}}_\ell, t_\ell)\right\}$.

Proving existence of low-space algorithm The first step towards giving a private space efficient algorithm for graph analysis is to prove that a space efficient algorithm exists. For this, we assume that the weighted complete graphs are not overlaid on the input stream in line. In other words, every checkpoint stores the exact graph formed from the timestamp of the checkpoints until the current time epoch. We prove that this algorithm maintains the *smooth Laplacian* property. Let us call this modified algorithm SLIDING-UPDATE.

Lemma 2. *Let* SLIDING-UPDATE *be the algorithm defined above that at time t get as input an edge e_t and a data structure $\mathfrak{D}_{space}$ satisfying the* smooth Laplacian *property. Then the output $\mathfrak{D}_{space} \leftarrow$*SLIDING-SPECTRAL$(e_t; \mathfrak{D}_{space})$ *also satisfy the* smooth Laplacian *property.*

Proof. The proof is by case analysis. Let $\mathsf{I} = \{t_1, \ldots, t_\ell\}$ be the timestamps before the updates and $\mathsf{I}' = \{\widetilde{t}_1, \ldots, \widetilde{t}_{\ell'}\}$ be the timestamps after the updates. Similarly, let $\mathcal{G}_1, \ldots, \mathcal{G}_\ell$ and $\mathcal{H}_1, \ldots, \mathcal{H}_{\ell'}$ be the corresponding matrices before and after an update. That is, if H and H' are the set of graphs before and after the update, then

$$\mathsf{H} := \{\mathcal{G}_1, \ldots, \mathcal{G}_\ell\}, \quad \mathsf{I} = \{t_1, \ldots, t_\ell\}$$
$$\mathsf{H}' := \{\mathcal{H}_1, \ldots, \mathcal{H}_{\ell'}\}, \quad \mathsf{I}' = \{\widetilde{t}_1, \ldots, \widetilde{t}_{\ell'}\}.$$

Now consider a timestamp t that is in both I and I', i.e., a timestamp that is not deleted. Suppose this timestamp is t_j for $t_j \in I$ and $\widetilde{t}_\ell$ for $\widetilde{t}_\ell \in I'$. We have following cases to consider:

Case 1: $t_{j+1} \notin \mathsf{I}'$

Case 2: $\begin{cases} t_{j+1} \in \mathsf{I}' & t_{j+1} = \widetilde{t}_p, t_{j+1} > t_j + 1 \\ t_{j+1} \in \mathsf{I}' & t_{j+1} = t_j + 1 \end{cases}$

Intuitively, Case 1 occurs when a timestamp is erased during the maintenance of the data structure following a stream update, and otherwise Case 2 occurs.

We first consider the case when a succeeding checkpoint gets deleted during an update. In this case, $\mathsf{Property}_1$ and $\mathsf{Property}_2$ is preserved merely due to the construction. The formal claim follows:

Claim 1 (Succeeding checkpoint gets deleted)**.** *Let* I *and* I' *be as defined above. Let* t_j *be as defined above. If* $t_{j+1} \notin \mathsf{I}'$*, i.e., the succeeding checkpoint is deleted on an update, then* $\mathsf{Property}_1$ *and* $\mathsf{Property}_2$ *holds.*

Proof. When $t_{j+1} \notin \mathsf{I}'$, i.e., $t_{j+1} \notin \{\widetilde{t}_{\ell+1}, \ldots, \widetilde{t}_{\ell'}\}$. Consider $\widetilde{t}_{\ell+1}$ and $t_{\ell+2}$ along with its corresponding matrices $\mathcal{H}_{\ell+1}$ and $\mathcal{H}_{\ell+2}$. The update rule step ensures that $L_{\mathcal{H}_{\ell+1}} \succeq (1-\rho)L_{\mathcal{H}_\ell}$ and that $L_{\mathcal{H}_{\ell+2}} \not\succeq (1-\rho)L_{\mathcal{H}_\ell}$. Therefore, $\mathsf{Property}_1$ and $\mathsf{Property}_2$ are maintained. □

Now we consider Case 2, i.e., when a succeeding checkpoint is not deleted on an update. Even if it is not deleted, due to an update, it is possible that the update leads to a temporary violation of the properties required by *smooth Laplacian*. Here, intuitively, the "smooth" behavior of positive semi-definite ordering comes to our rescue. In particular, the intuition is that since the properties were maintained before the updates and the checkpoints became a successor at some point in the past, because of "smooth" behavior of positive semidefinite matrices, the properties are maintained after the update. The proof is more subtle and we break it in two parts for the ease of the readers.

Claim 2 (Succeeding checkpoint is not deleted and not consecutive)**.** *Let* I *and* I' *be as defined above. Let* t_j *be as defined above. If* $t_{j+1} \in \mathsf{I}'$ *and* $t_{j+1} > t_j + 1$*, i.e., the checkpoints is in the updated data structure, then* $\mathsf{Property}_1$ *and* $\mathsf{Property}_2$ *holds.*

Proof. Since t_{j+1} is a successor of t_j, there must be a time $t' \leq t$ when it became a successor. Let $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$ be the graphs at the timestamps for time t'. Then we know that $L_{\mathcal{F}_{j+1}} \succeq (1-\rho)L_{\mathcal{F}_j}$ due to update rule. Now since $A \preceq B$ implies $A+C \preceq B+C$ for positive semidefinite matrices A, B, and C, we deduce that $L_{\mathcal{H}_{j+1}} \succeq (1-\rho)L_{\mathcal{H}_j}$, giving us $\mathsf{Property}_1$. Now let $p > \ell + 1$ be the index such that $\widetilde{t}_p = t_{j+1}$. $\mathsf{Property}_2$, i.e., $L_{\mathcal{H}_{p+1}} \not\succeq (1-\rho)\, L_{\mathcal{H}_\ell}$ follows from the update rule and the fact that $\mathcal{G}_{j+1}$ was not deleted and got updated to $\mathcal{H}_p$. □

For the final part, we assume that $(1-\rho)\, L_{\mathcal{H}_j} \preceq L_{\mathcal{H}_{j+1}}$. If not, then we are done because the property stated in item 2a holds. So suppose this is not the case. The following claims shows that in this case, both $\mathsf{Property}_1$ and $\mathsf{Property}_2$ holds.

Claim 3 (Succeeding checkpoint is not deleted and is consecutive)**.** *Let* I *and* I' *be as defined above. Let* t_j *be as defined above. If* $t_{j+1} \in \mathsf{I}'$ *and* $t_{j+1} = t_j + 1$*, i.e., the checkpoints is in the updated data structure, then* $\mathsf{Property}_1$ *and* $\mathsf{Property}_2$ *holds.*

Proof. $\mathsf{Property}_1$ holds because of the assumption that $(1-\rho)L_{\mathcal{H}_j} \preceq L_{\mathcal{H}_{j+1}}$. The proof that $\mathsf{Property}_2$ also holds simliarly as in the case when $t_{j+1} \neq t_j + 1$ (Claim 2). □

Combining Claim 1 to Claim 3 completes the proof of Lemma 2. □

Incorporating privacy Let $\mathcal{G}_i$ be the graph formed by streaming edge during the time $[t_i, T]$, $\mathcal{R}_i$ be a graph with weights sampled i.i.d. from $\mathcal{N}\left(0, \frac{2\log(W/\delta)}{\epsilon^2}\right)$, and $w = C\sqrt{\frac{\log(n)\log(W)/\delta}{n\epsilon^2}}$ for C to be defined later. Let

$$\widetilde{\mathcal{G}}_i = \mathcal{G}_i \cup wK_n \cup \mathcal{R}_i \tag{13}$$

be the n-vertex graph stored in the private data-structure corresponding to time stamp t_i. That is, $\mathfrak{D}_{\text{priv}} := \left\{(t_i, A_{\widetilde{\mathcal{G}}_i})\right\}$. Let $L_{\widetilde{\mathcal{G}}_i}$ be the corresponding Laplacian. This algorithm is presented as Sliding-Priv-Graph. The following is straightforward from the proof of Lemma 2 by replacing $\mathcal{G}_i$ by $\widetilde{\mathcal{G}}_i$.

Lemma 3. *Let* Sliding-Priv-Graph *be the algorithm defined in Algorithm 10 that at time t get as input an edge e_t and a data structure $\mathfrak{D}_{priv}$ satisfying the* smooth Laplacian *property. Then the output $\mathfrak{D}_{priv} \leftarrow$*Sliding-Priv-Graph$(e_t; \mathfrak{D}_{priv})$ *also satisfy the* smooth Laplacian *property.*

In what follows next, let

$$\beta := \sqrt{\frac{\log(n)\log(W/\delta)}{n\epsilon^2}}$$

for the sake of simplifying the expressions below.

While the smooth Laplacian property is straightforward, we now need to argue about the accuracy provided by this algorithm. Let L_W be the Laplacian of graph formed by the current window. $\mathcal{G}_1$ and $\mathcal{G}_2$ denote the input graph streamed between $[t_1, T]$ and $[t_2, T]$, respectively. By construction of the algorithm, the window is sandwiched between the first and second timestamp; therefore,

$$L_{\mathcal{G}_2} \preceq L \preceq L_{\mathcal{G}_1}. \tag{14}$$

Furthermore, $\mathsf{Property}_1$ implies that

$$(1-\rho)L_{\widetilde{\mathcal{G}}_1} \preceq L_{\widetilde{\mathcal{G}}_2}, \tag{15}$$

where $\widetilde{\mathcal{G}}_1$ and $\widetilde{\mathcal{G}}_2$ are the graphs maintained by Sliding-Priv-Graph in the data structure $\mathfrak{D}_{\text{priv}}$. Furthermore, by the definition of $\widetilde{\mathcal{G}}_1$ and $\widetilde{\mathcal{G}}_2$ in Equation (13) and the spectral bound on complete graph with Gaussian weights, we have

$$\begin{aligned} L_{\mathcal{G}_1} + (C-c)\beta L_{K_n} \preceq L_{\widetilde{\mathcal{G}}_1} \preceq L_{\mathcal{G}_1} + (C+c)\beta L_{K_n} \\ L_{\mathcal{G}_2} + (C-c)\beta L_{K_n} \preceq L_{\widetilde{\mathcal{G}}_2} \preceq L_{\mathcal{G}_2} + (C+c)\beta L_{K_n} \end{aligned} \tag{16}$$

Combining Equations 14 to 16 gives us

$$(1-2\rho)\left(L_W + c_1\beta L_{K_n}\right) \preceq L_{\widetilde{\mathcal{G}}_1} \preceq \frac{1}{(1-\rho)}\left(L_W + c_2\beta L_{K_n}\right) \preceq (1+2\rho)\left(L_W + c_2\beta L_{K_n}\right)$$

for sufficiently small ρ.

In other words, setting $\widetilde{\mathcal{G}} := \widetilde{\mathcal{G}}_1$ provides asymptotically the same level of accuracy as in the static case up to logarithmic factor. In particular, we have Theorem 20. □

Theorem 20 assumes that $W \gg n^3$, where W represents the number of events. This is a reasonable assumption in many scenarios. For instance, in the case when the edges of the graph represent financial transaction between two companies, there are only a small set of companies (nodes); however, in a window of a single day, there can be significantly more events when a transaction happens (the size of the window). Likewise, for social graphs, an edge represents interaction and nodes represent people. There can be significantly many interactions in a given time frame than the number of people. Therefore, this assumption is valid in practical scenarios where we would like to perform graph analysis. We would like to remark that our algorithm as presented is not optimized for run-time; however, one can use input-sparsity time algorithms to improve the run-time by a factor of n with a further $(1 \pm \rho)$ factor loss in the accuracy. In fact, the best known existing non-private algorithm Braverman et al. (2020) also takes aysmptotically the same time. Hence, the privacy overhead is constant.

D.5.1 Extension to continual release

Until now, we consider only one-shot algorithm as described by Dwork et al. Dwork et al. (2010a). Our one-shot algorithm computes spectral approximation with additive error of L_{K_n} with weights $\tau = O\left(\sqrt{\frac{\log(n)\log(W/\delta)}{n\varepsilon^2}}\right)$. Translating this accuray bound over the entire window, we get a total accuracy loss of $W\tau L_{K_n}$. In this section, we show how our one-shot algorithm can be converted to an algorithm that computes an approximation to graph Laplacian continually with $o(W\tau)L_{K_n}$ additive error over the entire window.

The continual release model was proposed by Dwork et al. Dwork et al. (2010a). In contrast to our setting, continual release model consider the entire data useful and does not put any space constraints. We provide two different protocols, in both of which we consider accuracy for only the update that came during the current window.

To get an intuition of our new algorithm, we first see SLIDING-PRIV-GRAPH in a slightly different manner. For this, it is helpful to revisit the concept of partial sums defined by Chan et al. Chan et al. (2011a). In partial sum, the goal is to evaluate a function on the values streamed in a particular time interval.

Definition 14 (Partial sum Chan et al. (2011a))**.** *A* partial sum*, denoted by $\mathfrak{P}(t_1, t_2)$-sum, is the sum of consecutive items streamed between time t_1 and t_2. That is, for $1 \le t_1 \le t_2$ and stream $(x_t)_{t\ge1}$, the partial sum is*

$$\mathfrak{P}(t_1, t_2)((x_t)) := \sum_{k=t_1}^{t_2} x_k.$$

We extend this definition with respect to any given input function, f, instead of simply the summation.

Definition 15 (Partial evaluation)**.** *Let $1 \le i \le j \le t$ and $(x_t)_{t\ge1}$ be the stream. Then a $\mathfrak{P}(f, i, j)$-function corresponding to a function f is a partial evaluation of the function on updates caused by consecutive items during time epoch i and j. It is defined as the partial sum for substream $x_{[i,j]}$ is denoted by $\mathfrak{P}(f, i, j) := f(x_i, \ldots, x_j)$.*

For example, in the case of graph Laplacian, the function in the question is the Laplacian L_{e_t} of the graph formed by the stream edge e_t with weight $w_t \ge 0$, and partial sum is defined as:

$$\mathfrak{P}(L, i, j) := \sum_{t=i}^{j} L_{e_t}.$$

Here L_{e_t} is a symmetric matrix whose projection over the co-ordinates (u, v), where u and v are the end vertices of the edge e_t, is

$$\begin{pmatrix} w_t & -w_t \\ -w_t & w_t, \end{pmatrix}$$

while every other entries are 0.

Now we can cast Algorithm 10 in terms of partial sums. Every single element in the window can appear in W release of the output. Therefore, the sensitivity of the output is W, and to preserve privacy, each partial evaluation must be perturbed with appropriate complete graph. Now consider an alternate algorithm that uses W space: we first privatize and store every entry in the stream and then use it to compute the output graph. In this case, we need W partial evaluation with every element having sensitivity 1. Using Theorem 8, we would get the following approximaation bound for the output graph $\widetilde{\mathcal{G}}$:

$$L_{\widetilde{\mathcal{G}}} \succeq (1-\rho)L_{\mathcal{G}} - O\left(\sqrt{\frac{W\log(n)\log(W/\delta)}{n\varepsilon^2}}\right) L_{K_n}$$

and

$$L_{\widetilde{\mathcal{G}}} \preceq (1+\rho)L_{\mathcal{G}} + O\left(\sqrt{\frac{W\log(n)\log(W/\delta)}{n\varepsilon^2}}\right) L_{K_n}.$$

This is a large loss in accuracy if W is large. Note that the space requirement is still low as we need to just store the checkpoints.

Looking at these two extreme ideas, we note that, if an algorithm uses p partial evaluations, such that every element in the stream can appear in at most s number of times in any of these sums, then it is easy to see that we get the following bound:

$$L_{\widetilde{\mathcal{G}}} \succeq (1-\rho)L_{\mathcal{G}} - O\left(s\sqrt{\frac{p\log(n)\log(Wp/\delta)}{n\varepsilon^2}}\right) L_{K_n}$$

and

$$L_{\widetilde{\mathcal{G}}} \preceq (1+\rho)L_{\mathcal{G}} + O\left(s\sqrt{\frac{p\log(n)\log(Wp/\delta)}{n\varepsilon^2}}\right) L_{K_n}.$$

The question now is whether we can further reduce this error. The basic idea is to build binary tree with leaves being the graphs at the checkpoint.

The first approach uses the same binary tree method used in Dwork et al. Dwork et al. (2010a) and Chan et al. Chan et al. (2011b). However, we depart from their technique in the sense that we only build the binary tree over the last W updates. Let $s_{T-W+1}, \ldots, s_T$ be the updates at any time T, where $s_i = (e_i, w_i)$ contains the edge and the weight information. Now we construct a binary tree as follows:

1. Every leaf node $1 \le i \le W$ consist of the graph formed by executing Priv-Graph (Algorithm 7) on the i-th update in the window.
2. Every node other the leaf node contains the private version of the graph formed by the edges in its subtree. That is, for a node n, if the leaves of its subtree contains the privatized version of graphs $\mathcal{G}_1, \ldots, \mathcal{G}_m$, then the node contains the graph $\mathcal{G}_n$ =Priv-Graph$(\mathcal{G}_1 \cup \cdots \cup \mathcal{G}_m)$.

The privacy budget in the algorithm is chosen as described in Chan et al. Chan et al. (2011b). This allows the error to scale as $\min\left\{\log^{3/2}(T), \log^{3/2}(W)\right\}$, when total T updates are made.

The space requirement of this construction is Wn^2. We can reduce the space requirement by running Priv-Sparsify (Algorithm 8) on the graphs in each nodes of the tree. This reduces the space on every node to be $O(n/\rho)$ For the accuracy guarantee, note that the Laplacian of graphs $\mathcal{H}$ formed by overlaying graphs $\mathcal{G}$ and $\bar{\mathcal{G}}$ is simply

$$L_{\mathcal{H}} = L_{\mathcal{G}} + L_{\bar{\mathcal{G}}}.$$

Further, if $L_{\mathcal{G}_1} \preceq \cdots \preceq L_{\mathcal{G}_\ell}$ satisfies the PSD ordering, then for any new graph $\mathcal{G}$,

$$L_{\mathcal{G}_1} + L_{\mathcal{G}} \preceq \cdots \preceq L_{\mathcal{G}_\ell} + L_{\mathcal{G}}.$$

Therefore, using Chan et al. Chan et al. (2011b), we get the following result:

Theorem 21. *Given the privacy parameter (ϵ, δ), approximation parameter $\rho \in (0, 1/2)$, and window size W, at time T, let L_W denote the Laplacian of the graph streamed in the time epochs $[T-W+1, T]$. Then we have the following:*

1. (Privacy guarantee). Continual-Sliding-Priv-Graph, *described in Algorithm 11, is an efficient (ϵ, δ)-differentially private algorithm under the sliding window model.*
2. (Spectral guarantee) Continual-Sliding-Priv-Graph *allows us to efficiently and continually output a graph $\widetilde{\mathcal{G}}$ with the following guarantee:*

$$(1-\rho)\left(L_W + c_1\tau L_{K_n}\right) \preceq L_{\widetilde{\mathcal{G}}} \preceq (1+\rho)\left(L_W + c_2\tau L_{K_n}\right).$$

   Here $c_1 < c_2$ are large constants and L_W is the Laplacian of the graph formed by the sliding window of size W and

$$\tau = \sqrt{\frac{\log(n)\log^3(W/\delta)}{n\epsilon^2}}.$$

3. (Space guarantee) *The space required by* Continual-Sliding-Priv-Graph *is $O\left(\frac{nW}{\rho}\right)$.*

D.6 Extension to continual release case with sublinear space

The space required by the previous algorithm is linear in W. While continual release does not prohibit using space linear in W, we give another algorithm that decreases the space at the cost of increasing the error. Depending on the requirement, one can use either of these two algorithms. The main idea is to divide the window in to small subwindow of size $\sqrt{W}$. We then run our algorithm on each of these subwindows separately. Let $\widetilde{\mathcal{G}}_1^{(j)}, \ldots, \widetilde{\mathcal{G}}_\ell^{(j)}$ be the graphs stored in the data structure for the j-th subwindow. For this, we fix some notation:

1. $\widetilde{\mathfrak{T}}^{(j)}$ be the binary tree formed by the leaves $\widetilde{\mathcal{G}}_1^{(j)}, \ldots, \widetilde{\mathcal{G}}_\ell^{(j)}$.
2. $\widetilde{\mathfrak{T}}_n^{(j)}$ be the subtree of the internal node n of the tree $\widetilde{\mathfrak{T}}^{(j)}$.
3. $\widetilde{\mathfrak{L}}_n^{(j)}$ be the leaves of the subtree $\widetilde{\mathfrak{T}}_n^{(j)}$. These forms a subset of the set $\left\{\widetilde{\mathcal{G}}_1^{(j)}, \ldots, \widetilde{\mathcal{G}}_\ell^{(j)}\right\}$.

We define a binary tree for each of these subwindow. For now, we drop the superscript (j). For the j-th subwindow, let Build-Binary-Tree be the subroutine that takes a number of graphs as input and constructs a graph as follows:

1. The leaf node contains of privatized graphs $\widetilde{\mathcal{G}}_1, \ldots, \widetilde{\mathcal{G}}_\ell$.
2. For every internal node, n, let $\mathfrak{L}_n$ be the graphs from the set $\{\mathcal{H}_1, \ldots, \mathcal{H}_\ell\}$ corresponding to the graphs in the set $\widetilde{\mathfrak{L}}_n$. Then the graph stored in the node n is the graph formed by first overlaying all the graphs in $\mathfrak{L}_n$ over each other and then privatizing it as in Step 8.
3. Delete all the internal nodes whose leaves contains graphs formed before time t_1.

The details of these changes are marked in red in Algorithm 11. Continual-Sliding-Priv-Graph maintains all these subwindows. We delete a subwindow if the leaves of its binary tree are before $T - W + 1$.

In other words, the above construction maintains the partial sum. Now, at every time epoch, we output the graph, $\widetilde{\mathcal{G}}$, stored in the first leaf in the first sub-window tree $\mathfrak{T}^{(1)}$. Since the number of checkpoints is $\ell = O\left(\frac{n}{\rho}\log W\right)$, combining Theorem 20 with that of Dwork et al. Dwork et al. (2010a), we have the following theorem:

Theorem 22. *Given the privacy parameter (ϵ, δ), approximation parameter $\rho \in (0, 1/2)$, and window size W, at time T, let L_W denote the Laplacian of the graph streamed in the time epochs $[T - W + 1, T]$. Then we have the following:*

1. (Privacy guarantee). Continual-Sliding-Priv-Graph, *described in Algorithm 11, is an efficient (ϵ, δ)-differentially private algorithm under the sliding window model.*
2. (Spectral guarantee) Continual-Sliding-Priv-Graph *allows us to efficiently and continually output a graph $\widetilde{\mathcal{G}}$ with the following guarantee:*

$$(1-\rho)\left(L_W + c_1\tau L_{K_n}\right) \preceq L_{\widetilde{\mathcal{G}}} \preceq (1+\rho)\left(L_W + c_2\tau L_{K_n}\right).$$

Here $c_1 < c_2$ are large constants and L_W is the Laplacian of the graph formed by the sliding window of size W and

$$\tau = W^{3/4}\sqrt{\frac{\log(n)\log^3(W/\delta)\log^3(n\log(W))}{n\rho^3\epsilon^2}}.$$

3. (Space guarantee) *The space required by* Continual-Sliding-Priv-Graph *is $O\left(\frac{n^3}{\rho}\sqrt{W}\log(W)\right)$.*

E Spectral Norm of Random Graph with Subgaussian Weights

In this section, we consider the following graph, $\mathcal{G}$. Every edge is an independent copy of a zero mean unit variance subgaussian random variable with probability p, and is identically 0, otherwise. We use the notation $\mathcal{SG}(\mu, \sigma^2)$ to denote a subgaussian distribution with mean μ and variance σ^2. As before, we can rewrite the Laplacian of $\mathcal{G}$ as sum of three random matrices (which are dependent):

$$L_{\mathcal{G}} := S - G + D,$$

where S and D are formed as before from G and, in turn, G is formed as below: for all $i < j$, let $g_{ij} \sim \mathcal{SG}(0, 1)$. Then

$$G[i, j] = \begin{cases} g_{ij} & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases}$$

For $i > j$, we set $G[i, j] = G[i, j]$.

Before we bound the spectrum of random graph generated as above, we give a bound on the spectrum of complete graph with subgaussian weights:

Theorem 23 (Subgaussian entries)**.** *Let $\mathcal{G}$ be a graph whose weights are sampled i.i.d. from $\mathcal{SG}(0, 1)$, then $\|L_{\mathcal{G}}\|_2 = O(\sqrt{n \log(n)})$ with probability $1 - 3e^{-cn}$ for an absolute constant $c > 0$.*

Proof. The proof of Theorem 23 follows similarly as in the proof of Theorem 13 by noting that Theorem 12 holds for any subgaussian random variable, and by invoking Chernoff-Hoeffding for subgaussian distribution in the proof of Theorem 13.

We show the result separately for each of two cases. In each of the case, let us consider $L_{\mathcal{G}} = G - D + S$, where G be the random matrix whose entries are sampled either i.i.d. from $\mathcal{N}(0, 1)$ or $\mathsf{Lap}(0, 1)$, D and S are diagonal matrices formed in the following manner:

$$D[i, i] = G[i, i] \qquad \text{and} \qquad S[i, i] = \sum_j G[i, j].$$

Now, using subadditivity of spectral norm, we have

$$s_{\max}(L_{\mathcal{G}}) \leq s_{\max}(G) + s_{\max}(D) + s_{\max}(S).$$

In particular, $s_{\max}(G) = \widetilde{O}(\sqrt{n})$ for both Gaussian random variable as they both have finite fourth moment. Now we bound the rest of the two terms separately. We have

$$\begin{aligned} s_{\max}(D) &= \max_{e_i} |e_i^\top D e_i| = \max_{e_i} |e_i^\top G e_i| \\ &\leq \max_{v \in \mathbb{S}^{n-1}} |v^\top G v| = s_{\max}(G). \end{aligned}$$

Finally, we bound the last term. We first note that

$$s_{\max}(S) \leq \max_i \left| \sum_{j \neq i} G[i, j] \right|.$$

The standard Chernoff-Hoefding bound for the sum of random Gaussian variable gives that with probability $1 - e^{-cn}$, $s_{\max}(S) \leq C\sqrt{n} \log(n)$. The theorem follows by using union bound. □

We now move to the case of random graphs with subgaussian weights defined earlier. As before to give a high level idea, we use $\mathcal{N}(0, 1)$ as an example. The Laplacian $L_{\mathcal{G}}$ is a Hermittian matrix and therefore, its singular value can be bounded using the following result by Tropp Tropp (2015).

Theorem 24 (Tropp Tropp (2015)). *Consider a finite sequence $\{X_k\}$ of independent, random, Hermitian matrices with dimension d. Assume that $\mathbb{E}[X_k] = 0$ and $\lambda_{\max}(X_k) \leq L$ for each index k. Introduce the random matrix $Y = \sum_k X_k$. Let $v(Y)$ be the matrix variance statistic of the sum: $v(Y) = \left\|\sum_k \mathsf{E}[Y^2]\right\|$. Then for all $t \geq 0$,*

$$\Pr\left[\|Y\|_2 \geq t\right] \leq d \cdot \exp\left(\frac{-t^2/2}{v + Lt/3}\right).$$

Lemma 4 (Erdos-Renyi graph with subgaussian entries). *The spectral norm of the Laplacian of the graph formed as above is $\widetilde{O}(p\sqrt{n})$, almost surely.*

Proof. We use Theorem 24 to prove that G satisfies the required property. For this we define random matrix variable X_k to be an all zero entry except the (i,j)-th entry which is G_{ij}, where $k = (i-1)n + j$ for $i < j$. We now need to bound $v(Y)$ and $\mathsf{E}[X_k]$.

It is easy to verify that $\mathsf{E}[X_k] = 0$ and $\left\|\sum_k \mathsf{E}[X_k^2]\right\| = \widetilde{O}(p\sqrt{n})$. In other words, setting $t = cp\sqrt{n}\log(n)$ in Theorem 24 and $d = n$, we have

$$\begin{aligned}\Pr\left[\|Y\|_2 \geq cp\sqrt{n}\log(n)\right] &\leq n \cdot \exp\left(-cp\sqrt{n}\log(n)\right) \\ &\leq \exp(-cp\sqrt{n}).\end{aligned}$$

The proof that S satisfies the required property is similar as before using the Hoeffding's bound. This completes the proof of Lemma 4. □

F Projection on a positive semi-definite cone

In many applications, it is desirable that graphs have only positive weights; however, Theorem 17 does not provide any such guarantee. Our second result is a general technique that allows us to output a graph with positive weights if the input graph has small enough spectrum and is positive semi-definite.

Theorem 25. *Given a graph $\mathcal{G}$ with edge weights either positive or negative whose Laplacian is a PSD and spectral norm bounded by $O\left(\sqrt{\frac{n\log(n)\log(1/\delta)}{\epsilon^2}}\right)$, there is an efficient algorithm that outputs a graph $\widehat{\mathcal{G}}$ with positive edge weights such that*

$$\left\|L_{\widehat{\mathcal{G}}} - L_{\mathcal{G}}\right\|_2 \leq C\sqrt{\frac{n\log(n)\log(1/\delta)}{\epsilon^2}},$$

where C is an absolute constant and $\|\cdot\|$ is the spectral norm.

The proof of Theorem 25 requires a spectral projection onto a semidefinite cone. This is in contrast with previous approaches that were based on projection-based methods using ℓ_2 projection onto a convex cone after specific convex relaxation and employing Frank-Wolfe method Dwork et al. (2015); Nikolov et al. (2013). On the other hand, we are required to show that an appropriate semidefinite program has an optimal solution. This we believe will have applications beyond what is listed in this paper.

The Laplacian of graphs with non-negative weights are denoted by $\mathbb{L}^n \subset \mathbb{S}^n_+$. That is,

$$\mathbb{L}^n = \left\{X : X[i,j] \leq 0 \quad \forall i,j \in [n] : i \neq j \quad \text{and} \quad \sum_{j=1}^n X[i,j] = 0 \quad \forall i \in [n]\right\}. \tag{17}$$

The set of Laplacian matrices forms a closed convex cone. We relax the last set of constraints $\sum_{j=1}^n X[i,j] = 0 \quad \forall i \in [n]$ to get a convex relaxation of $\mathbb{L}^n$. Let $\mathbb{L}^n(\varrho)$ is convex relaxation of set of Laplacian of graphs with positive weights. This set can be expressed as an intersection of cone of positive semidefinite matrices and affine space.

$$\mathbb{L}^n(\varrho) = \left\{X : X[i,j] \leq 0 \quad \forall i,j \in [n] : i \neq j \quad \text{and} \quad 0 \leq \sum_{j=1}^n X[i,j] \leq \varrho \quad \forall i \in [n]\right\}.$$

To make this argument, we need to show that the corresponding semi-definite program (SDP(1) defined below) has an attainable optimal solution. We demonstrate this using the duality theory of semidefinite optimization (see Lemma 5).

$$\begin{aligned} & \underline{\mathsf{SDP}(1)} \\ \text{minimize: } & \gamma \\ \text{subject to: } & L_{\widetilde{\mathcal{H}}} - L_{\mathcal{G}'} \preceq \gamma \mathbb{1}_n, \\ & L_{\mathcal{G}'} - L_{\widetilde{\mathcal{H}}} \preceq \gamma \mathbb{1}_n, \\ & \gamma \geq 0, \\ & L_{\mathcal{G}'} \in \mathbb{L}^n(\varrho). \end{aligned}$$

The attainability and existence of an optimal solution are guaranteed by what is known as *Slater conditions*. We refer to Boyd and Vandenberghe (2004) for a rigorous definition on Slater conditions. In our case, satisfying Slater conditions reduces to finding *strictly feasible solution* for the dual of SDP(1) as described in Lemma 5 below.

To write the dual program, we first introduce two sets of matrices $\{F_i : i \in [n]\}$ and $\{E_{ij} : i, j \in [n]\}$. These matrices can be expressed in terms of standard basis vectors $\{\bar{e}_i : i \in [n]\}$ and vector of all 1's, $\bar{e}$, as follows:

$$F_i = \bar{e}_i \bar{e}^\top + \bar{e}\bar{e}_i^\top \qquad \text{and} \qquad E_{ij} = \bar{e}_i \bar{e}_j^\top + \bar{e}_j \bar{e}_i^\top.$$

Armed with these matrices, we can write the dual program as below:

$$\begin{aligned} & \underline{\text{Dual of } \mathsf{SDP}(1)} \\ \text{maximize: } & \langle L_{\widetilde{\mathcal{H}}}, X_1 - X_2 \rangle + \langle \varrho \bar{e}, z \rangle \\ \text{subject to: } & \langle \mathbb{1}_n, X_1 \rangle + \langle \mathbb{1}_n, X_2 \rangle \leq 1 \\ & X_1 - X_2 \preceq \sum_{i=1}^{n} (z[i] - y[i])\, F_i + \sum_{1 \leq i < j \leq n} x[(i,j)] E_{ij} \\ & X_1, X_2 \in \mathbb{S}_+^n, \\ & y, z \in \mathbb{R}_+^n, \\ & x \in \mathbb{R}_+^{n(n-1)/2}. \end{aligned}$$

Lemma 5. *The Slater conditions for* SDP(1) *is satisfied. That is, there exist*

$$\bar{X}_1, \bar{X}_2 \in \mathbb{S}_{++}^n \text{ and } \bar{y}, \bar{z} \in \mathbb{R}_{++}^n, \bar{x} \in \mathbb{R}_{++}^{n(n-1)/2}$$

such that

$$\begin{aligned} & \langle \mathbb{1}_n, \bar{X}_1 \rangle + \langle \mathbb{1}_n, \bar{X}_2 \rangle < 1 \quad \text{and} \\ & \bar{X}_1 - \bar{X}_2 \prec \sum_{i=1}^{n} (\bar{z}[i] - \bar{y}[i])\, F_i + \sum_{1 \leq i < j \leq n} \bar{x}[(i,j)] E_{ij}. \end{aligned} \tag{18}$$

Proof. We will construct $\bar{X}_1, \bar{X}_2 \in \mathbb{S}_{++}^n$, $\bar{y}, \bar{z} \in \mathbb{R}_{++}^n$, and $\bar{x} \in \mathbb{R}_{++}^{n(n-1)/2}$ that satisfy Equation (18). Note that the set

$$\{\bar{X}_1, \bar{X}_2, \bar{y}, \bar{z}, \bar{x}\}$$

form a feasible solution of dual of SDP(1). For $\alpha, \beta > 0$, let $\bar{y} = (\alpha, \alpha, \ldots, \alpha)^\top \in \mathbb{R}_{++}^n$, $\bar{z} = 2\bar{y}$, and $\bar{x} =$

$(\beta, \beta, \ldots, \beta)^\top \in \mathbb{R}^{n(n-1)/2}_{++}$. For such choices of $(\bar{x}, \bar{y}, \bar{z})$, it holds that

$$\begin{aligned}
&\sum_{i=1}^{n} (\bar{z}[i] - \bar{y}[i])\, F_i + \sum_{1 \le i < j \le n} \bar{x}[(i,j)] E_{ij} \\
&= \alpha \sum_{i=1}^{n} \left(\bar{e}\bar{e}_i^\top + \bar{e}_i \bar{e}^\top \right) + \beta \sum_{1 \le i < j \le n} \bar{e}_i \bar{e}_j^\top + \bar{e}_j \bar{e}_i^\top \\
&= (2\alpha + \beta)\, \bar{e}\bar{e}^\top - \beta \sum_{i=1}^{n} \bar{e}_i \bar{e}_i^\top = (2\alpha + \beta)\, \bar{e}\bar{e}^\top - \beta\, I_n.
\end{aligned}$$

Let

$$\begin{aligned}
&\alpha = \frac{1}{16n}, \qquad \beta = \frac{1}{16n}, \\
&\bar{X}_1 = (2\alpha + \beta)\, \bar{e}\bar{e}^\top + \beta \mathbb{1}_n, \\
&\bar{X}_2 = 3\beta \mathbb{1}_n.
\end{aligned}$$

Clearly $\bar{X}_1 - \bar{X}_2 = (2\alpha + \beta)\, \bar{e}\bar{e}^\top - 2\beta \mathbb{1}_n$ and hence

$$\bar{X}_1 - \bar{X}_2 \prec \sum_{i=1}^{n} (\bar{z}[i] - \bar{y}[i])\, F_i + \sum_{1 \le i < j \le n} \bar{x}[(i,j)] E_{ij}.$$

Moreover, $\langle \mathbb{1}_n, \bar{X}_1 \rangle + \langle \mathbb{1}_n, \bar{X}_2 \rangle = \langle \mathbb{1}_n, \bar{X}_1 + \bar{X}_2 \rangle = \mathsf{Tr}\left(\bar{X}_1 + \bar{X}_2\right) = (2\alpha + 5\beta)n = \frac{7}{16} < 1$. Hence we have constructed $\bar{X}_1, \bar{X}_2 \in \mathbb{S}^n_{++}$and $\bar{y}, \bar{z} \in \mathbb{R}^n_{++}$and $\bar{x} \in \mathbb{R}^{n(n-1)/2}_{++}$ such that Equation (18) is satisfied. This proves that the Slater conditions for $\mathsf{SDP}(1)$ are satisfied, and the optimal solution of $\mathsf{SDP}(1)$ exists and is attained. This completes the proof of Lemma 5. □

It is not difficult to see that the Slater conditions for the dual of $\mathsf{SDP}(1)$ are satisfied as well. For suffiently small $\epsilon \in (0, 1/2)$, we instantiate a feasible solution (that satisfies Slater conditions) of $\mathsf{SDP}(1)$ as follows:

$$\bar{L}_{\mathcal{G}'}[i,j] = \begin{cases} -\epsilon/n^2 & \text{if } i \neq j, \\ 2\epsilon/n & \text{if } i = j. \end{cases}$$

We let $\bar{\gamma} = 2\left(\left\|L_{\widetilde{\mathcal{H}}}\right\|_2 + \left\|\bar{L}'_{\mathcal{G}}\right\|_2\right)$. These choices of SDP variables satisfy the Slater condition for the dual of $\mathsf{SDP}(1)$.

G Open Problems

One of the problems that we leave for future research is characterizing the space complexity of the private spectral approximation of graphs in the sliding window model. It is well known that one requires $\Omega(n^2)$ space to approximate the spectrum of the graph Laplacian even in the static setting without any privacy constraints. We believe that $O\left(\frac{\log W}{\rho}\right)$ overhead is necessary in the sliding window model even without privacy. It remains open whether the dependence on n is cubic (as in this paper) or quadratic (matching the non-private setting).

Another line of work that is of interest is a spectral approximation of matrices, an important numerical linear algebra problem. It subsumes spectral approximation of a graph as a special case. In the non-private setting, Braverman et al. (2020) used a variant of ridge-leverage score to compute a spectral approximation. However, it is not clear if their technique can be extended to the private setting. Similarly, our analysis and algorithm rely crucially on the structural properties of Laplacian of graphs, something that is not present in a more general case. As such, we believe that one would need an entirely different technique for private spectral approximation of matrices.

Algorithm 11 CONTINUAL-SUBWINDOW-PRIV-GRAPH($\mathfrak{D}_{\text{priv}}; (e_t, w_t)$)

Input: A new edge-weight pair (e_t, w_t) and $\mathfrak{D}_{\text{priv}}$ containing ℓ tuples

$$\left\{(\widetilde{\mathcal{G}}_1, \mathcal{H}_1, t_1), \ldots, (\widetilde{\mathcal{G}}_\ell, \mathcal{H}_\ell, t_\ell)\right\},$$

a binary tree $\mathfrak{T}$ with $\left\{\widetilde{\mathcal{G}}_1, \ldots, \widetilde{\mathcal{G}}_\ell\right\}$ as leaves.

Output: Updated $\mathfrak{D}_{\text{priv}}$ and $\mathfrak{T}$.

Stage 1: Include new edge (e_t, w_t) in the data structure, $\mathfrak{D}_{\text{priv}}$

if $t_2 < t - W + 1$ **then**
  Set $t_j = t_{j+1}$.
  Set $\widetilde{\mathcal{G}}_j = \widetilde{\mathcal{G}}_{j+1}$.
  Set $\mathcal{H}_i = \mathcal{H}_{j+1}$ for $1 \le j \le \ell - 1$ ▷ Delete the expired timestamp.
  Set $\ell = \ell - 1$.
end if
Construct a complete graph $\mathcal{R}_{\ell+1}$ with edge weights sampled i.i.d. from $\mathcal{N}\left(0, \frac{4 \log 1/\delta}{\epsilon^2}\right)$.
Privatization step. Let $\mathcal{H}_t$ be a single-edge graph with weight w_t on the edge e_t. Set

$$t_{\ell+1} = t, \quad \widetilde{\mathcal{G}}_{\ell+1} := \mathcal{H}_t \cup \mathcal{R}_{\ell+1} \cup \left(C\sqrt{\frac{\log(n)}{n\epsilon^2}\log\left(\frac{W}{\delta}\right)}K_n\right).$$

.
Update $\mathfrak{D}_{\text{priv}} \leftarrow \mathfrak{D}_{\text{priv}} \cup (\widetilde{\mathcal{G}}_{\ell+1}, \mathcal{H}_t, t_{\ell+1})$, $\ell = \ell + 1$.
for $i = 1, \ldots, \ell - 1$ **do**
  Compute $\widetilde{\mathcal{G}}_i \leftarrow \mathcal{H}_t \cup \widetilde{\mathcal{G}}_i$, $\mathcal{H}_i = \mathcal{H}_i \cup \mathcal{H}_t$.
end for

Stage 2: Update $\mathfrak{D}_{\text{priv}}$ to maintain smooth Laplacian property

Find $j := \min\left\{p : L_{\mathcal{G}_p} \not\succeq L_{\mathcal{G}_\ell}\right\}$ and delete $L_{\mathcal{G}_p}, \ldots, L_{\mathcal{G}_{\ell-1}}$.
Update $L_{\mathcal{G}_p} = L_{\mathcal{G}_\ell}, \ell = p$.
for $i = 1, \ldots, \ell - 2$ **do**
  Find $\widetilde{\mathcal{G}}_{i+1}, \ldots, \widetilde{\mathcal{G}}_j$ such that

$$(1-\rho)\, L_{\widetilde{\mathcal{G}}_i} \preceq L_{\widetilde{\mathcal{G}}_j} \quad \text{and} \quad (1-\rho)\, L_{\widetilde{\mathcal{G}}_i} \not\preceq L_{\widetilde{\mathcal{G}}_{j+1}}.$$

  Delete $\widetilde{\mathcal{G}}_{i+1}, \ldots, \widetilde{\mathcal{G}}_{j-1}$.
  Set k=1
  while $j + k < \ell$ **do** ▷ Reorder the indices
    Update $\widetilde{\mathcal{G}}_{i+k} = \widetilde{\mathcal{G}}_{j+k-1}$,
    Update $\mathcal{H}_{i+k} = \mathcal{H}_{j+k-1}$.
    Update $t_{i+k} = t_{j+k-1}$.
    Update $k := k + 1$.
  end while
  $\ell := \ell + i - j + 1$.
end for
$\mathfrak{T} \leftarrow$ BUILD-BINARY-TREE($\widetilde{\mathcal{G}}_1, \ldots, \widetilde{\mathcal{G}}_\ell, \mathcal{H}_1, \ldots, \mathcal{H}_\ell$).
Output: $\mathfrak{D}_{\text{priv}} := \left\{(\widetilde{\mathcal{G}}_1, \mathcal{H}_1, t_1), \ldots, (\widetilde{\mathcal{G}}_\ell, \mathcal{H}_\ell, t_\ell)\right\}$ and $\mathfrak{T}$.