An SDP-Based Algorithm for Linear-Sized Spectral Sparsification

Yin Tat Lee
Microsoft Research
Redmond, USA
yile@microsoft.com

He Sun
University of Bristol
Bristol, UK
h.sun@bristol.ac.uk

ABSTRACT

For any undirected and weighted graph $G = (V, E, w)$ with n vertices and m edges, we call a sparse subgraph H of G, with proper reweighting of the edges, a $(1+\varepsilon)$-spectral sparsifier if

$$(1-\varepsilon)x^{\intercal}L_G x \le x^{\intercal}L_H x \le (1+\varepsilon)x^{\intercal}L_G x$$

holds for any $x \in \mathbb{R}^n$, where L_G and L_H are the respective Laplacian matrices of G and H. Noticing that $\Omega(m)$ time is needed for any algorithm to construct a spectral sparsifier and a spectral sparsifier of G requires $\Omega(n)$ edges, a natural question is to investigate, for any constant ε, if a $(1+\varepsilon)$-spectral sparsifier of G with $O(n)$ edges can be constructed in $\widetilde{O}(m)$ time, where the $\widetilde{O}$ notation suppresses polylogarithmic factors. All previous constructions on spectral sparsification require either super-linear number of edges or $m^{1+\Omega(1)}$ time.

In this work we answer this question affirmatively by presenting an algorithm that, for any undirected graph G and $\varepsilon > 0$, outputs a $(1+\varepsilon)$-spectral sparsifier of G with $O(n/\varepsilon^2)$ edges in $\widetilde{O}(m/\varepsilon^{O(1)})$ time. Our algorithm is based on three novel techniques: (1) a new potential function which is much easier to compute yet has similar guarantees as the potential functions used in previous references; (2) an efficient reduction from a two-sided spectral sparsifier to a one-sided spectral sparsifier; (3) constructing a one-sided spectral sparsifier by a semi-definite program.

CCS CONCEPTS

• **Mathematics of computing → Probabilistic algorithms**; • **Theory of computation → Sparsification and spanners**;

KEYWORDS

spectral graph theory, spectral sparsification

ACM Reference format:
Yin Tat Lee and He Sun. 2017. An SDP-Based Algorithm for Linear-Sized Spectral Sparsification. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 10 pages.
DOI: 10.1145/3055399.3055477

1 INTRODUCTION

A sparse graph is one whose number of edges is reasonably viewed as being proportional to the number of vertices. Since most algorithms run faster on sparse instances of graphs and it is more space-efficient to store sparse graphs, it is useful to obtain a sparse representation H of G so that certain properties between G and H are preserved, see Figure 1 for an illustration. Over the past three decades, different notions of graph sparsification have been proposed and widely used to design approximation algorithms. For instance, a *spanner* H of a graph G is a subgraph of G so that the shortest path distance between any pair of vertices is approximately preserved [6]. Benczúr and Karger [5] defined a *cut sparsifier* of a graph G to be a sparse subgraph H such that the value of any cut between G and H are approximately the same. In particular, Spielman and Teng [16] introduced a *spectral sparsifer*, which is a sparse subgraph H of an undirected graph G such that many spectral properties of the Laplacian matrices between G and H are approximately preserved. Formally, for any undirected graph G with n vertices and m edges, we call a subgraph H of G, with proper reweighting of the edges, a $(1+\varepsilon)$-spectral sparsifier if

$$(1-\varepsilon)x^{\intercal}L_G x \le x^{\intercal}L_H x \le (1+\varepsilon)x^{\intercal}L_G x$$

holds for any $x \in \mathbb{R}^n$, where L_G and L_H are the respective Laplacian matrices of G and H. Spectral sparsification has been proven to be a remarkably useful tool in algorithm design, linear algebra, combinatorial optimisation, machine learning, and network analysis.

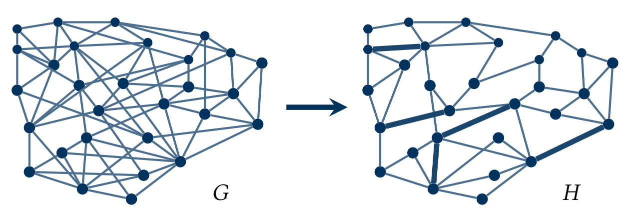

Figure 1: The graph sparsification is a reweighted subgraph H of an original graph G such that certain properties are preserved. These subgraphs are sparse, and are more space-efficient to be stored than the original graphs. The picture above uses the thickness of edges in H to represent their weights.

In the seminal work on spectral sparsification, Spielman and Teng [16] showed that, for any undirected graph G of n vertices, a spectral sparsifier of G with only $O(n \log^c n/\varepsilon^2)$ edges exists and can be constructed in nearly-linear time[1], where $c \geq 2$ is some constant. Both the runtime of their algorithm and the number of edges in the output graph involve large poly-logarithmic factors, and this motivates a sequence of simpler and faster constructions of spectral sparsifiers with fewer edges [2, 3, 10]. In particular, since any constant-degree expander graph of $O(n)$ edges is a spectral sparsifier of an n-vertex complete graph, a natural question is to study, for any n-vertex undirected graph G and constant $\varepsilon > 0$, if a $(1+\varepsilon)$-spectral sparsifier of G with $O(n)$ edges can be constructed in nearly-linear time. Being considered as one of the most important open questions about spectral sparsification by Batson et al. [4], there has been many efforts for fast constructions of linear-sized spectral sparsifiers, e.g. [2, 10], however the original problem posed in [4] has remained open.

In this work we answer this question affirmatively by presenting the first nearly-linear time algorithm for constructing a linear-sized spectral sparsifier. The formal description of our result is as follows:

Theorem 1.1. *Let G be any undirected graph with n vertices and m edges. For any $0 < \varepsilon < 1$, there is an algorithm that runs in $\widetilde{O}\left(m/\varepsilon^{O(1)}\right)$ work, $\widetilde{O}\left(1/\varepsilon^{O(1)}\right)$ depth, and produces a $(1+\varepsilon)$-spectral sparsifier of G with $O\left(n/\varepsilon^2\right)$ edges*[2].

Theorem 1.1 shows that a linear-sized spectral sparsifier can be constructed in nearly-linear time in a single machine setting, and in polylogarithmic time in a parallel setting. The same algorithm can be applied to the matrix setting, whose result is summarised as follows:

Theorem 1.2. *Given a set of m PSD matrices $\{M_i\}_{i=1}^m$, where $M_i \in \mathbb{R}^{n\times n}$. Let $M = \sum_{i=1}^m M_i$ and $Z = \sum_{i=1}^m \mathrm{nnz}(M_i)$, where $\mathrm{nnz}(M_i)$ is the number of non-zero entries in M_i. For any $0 < \varepsilon < 1$, there is an algorithm that runs in $\widetilde{O}\left((Z + n^{\omega})/\varepsilon^{O(1)}\right)$ work, $\widetilde{O}\left(1/\varepsilon^{O(1)}\right)$ depth and produces a $(1+\varepsilon)$-spectral sparsifier of M with $O\left(n/\varepsilon^2\right)$ components, i.e., there is an non-negative coefficients $\{c_i\}_{i=1}^m$ such that $|\{c_i | c_i \neq 0\}| = O\left(n/\varepsilon^2\right)$, and*

$$(1-\varepsilon)\cdot M \preceq \sum_{i=1}^m c_i M_i \preceq (1+\varepsilon)\cdot M. \tag{1}$$

Here ω is the matrix multiplication constant.

1.1 Related Work

In the seminal paper on spectral sparsification, Spielman and Teng [16] showed that a spectral sparsifier of any undirected graph G can be constructed by decomposing G into multiple nearly expander graphs, and sparsifying each subgraph individually. This method leads to the first nearly-linear time algorithm for constructing a spectral sparsifier with $O(n \log^c n/\varepsilon^2)$ edges for some $c \geq 2$. However, both the runtime of their algorithm and the number of edges in the output graph involve large poly-logarithmic factors. Spielman and Srivastava [15] showed that a $(1+\varepsilon)$-spectral sparsifier of G with $O(n\log n/\varepsilon^2)$ edges can be constructed by sampling the edges of G with probability proportional to their effective resistances, which is conceptually much simpler than the algorithm presented in [16].

Noticing that any constant-degree expander graph of $O(n)$ edges is a spectral sparsifier of an n-vertex complete graph, Spielman and Srivastava [15] asked if any n-vertex graph has a spectral sparsifier with $O(n)$ edges. To answer this question, Batson, Spielman and Srivastava [3] presented a polynomial-time algorithm that, for any undirected graph G of n vertices, produces a spectral sparsifier of G with $O(n)$ edges. At a high level, their algorithm, a.k.a. the BSS algorithm, proceeds for $O(n)$ iterations, and in each iteration *one edge* is chosen deterministically to "optimise" the change of some potential function. Allen-Zhu et al. [2] noticed that a less "optimal" edge, based on a different potential function, can be found in almost-linear time and this leads to an almost-quadratic time algorithm. Generalising their techniques, Lee and Sun [10] showed that a linear-sized spectral sparsifier can be constructed in time $O\left(m^{1+c}\right)$ for an arbitrary small constant c. All of these algorithms proceed for $\Omega(n^c)$ iterations, and every iteration takes $\Omega(m^{1+c})$ time for some constant $c > 0$. Hence, to break the $\Omega(m^{1+c})$ runtime barrier faced in all previous constructions multiple new techniques are needed.

1.2 Organisation

The remaining part of the paper is organised as follows. We introduce necessary notions about matrices and graphs in Section 2. In Section 3 we overview our algorithm and proof techniques. For readability, more detailed discussions and technical proofs are deferred to Section 4.

2 PRELIMINARIES

2.1 Matrices

For any $n\times n$ real and symmetric matrix A, let $\lambda_{\min}(A) = \lambda_1(A) \leq \cdots \leq \lambda_n(A) = \lambda_{\max}(A)$ be the eigenvalues of A, where $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ represent the minimum and maximum eigenvalues of A. We call a matrix A positive semi-definite (PSD) if $x^\intercal A x \geq 0$ holds for any $x \in \mathbb{R}^n$, and a matrix A positive definite if $x^\intercal A x > 0$ holds for any $x\in\mathbb{R}^n$. For any positive definite matrix A, we define the corresponding ellipsoid by $\mathsf{Ellip}(A) \triangleq \left\{x:\ x^\intercal A^{-1} x \leq 1\right\}$.

2.2 Graph Laplacian

Let $G = (V, E, w)$ be a connected, undirected and weighted graph with n vertices, m edges, and weight function $w : E \to \mathbb{R}_{\geq 0}$. We fix an arbitrary orientation of the edges in G, and let $B \in \mathbb{R}^{m\times n}$ be the *signed edge-vertex incidence matrix* defined by

$$B_G(e,v) = \begin{cases} 1 & \text{if } v \text{ is } e\text{'s head,}\\ -1 & \text{if } v \text{ is } e\text{'s tail,}\\ 0 & \text{otherwise.}\end{cases}$$

We define an $m\times m$ diagonal matrix W_G by $W_G(e,e) = w_e$ for any edge $e \in E[G]$.

The Laplacian matrix of G is an $n\times n$ matrix L defined by

$$L_G(u,v) = \begin{cases} -w(u,v) & \text{if } u \sim v,\\ \deg(u) & \text{if } u = v,\\ 0 & \text{otherwise,}\end{cases}$$

[1] We say a graph algorithm runs in nearly-linear time if the algorithm runs in $O(m \cdot \operatorname{poly}\log n)$ time, where m and n are the number of edges and vertices of the input graph.

[2] Here, the notation $\widetilde{O}(\cdot)$ hides a factor of $\log^c n$ for some positive constant c.

where $\deg(v) = \sum_{u \sim v} w(u, v)$. It is easy to verify that

$$x^{\intercal} L_G x = x^{\intercal} B_G^{\intercal} W_G B_G x = \sum_{u \sim v} w_{u,v} (x_u - x_v)^2 \geq 0$$

holds for any $x \in \mathbb{R}^n$. Hence, the Laplacian matrix of any undirected graph is a PSD matrix. Notice that, by setting $x_u = 1$ if $u \in S$ and $x_u = 0$ otherwise, $x^{\intercal} L_G x$ equals to the value of the cut between S and $V \setminus S$. Hence, a spectral sparsifier is a stronger notion than a cut sparsifer.

2.3 Other Notations

For any sequence $\{\alpha_i\}_{i=1}^m$, we use $\mathrm{nnz}(\alpha)$ to denote the number of non-zeros in $\{\alpha_i\}_{i=1}^m$. For any two matrices A and B, we write $A \preceq B$ to represent $B - A$ is PSD, and $A \prec B$ to represent $B - A$ is positive definite. For any two matrices A and B of the same dimension, let $A \bullet B \triangleq \mathrm{tr}\,(A^{\intercal} B)$, and

$$A \oplus B = \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}.$$

3 OVERVIEW OF OUR ALGORITHM

Without loss of generality we study the problem of sparsifying the sum of PSD matrices. The one-to-one correspondence between the construction of a graph sparsifier and the following Problem 1 was presented in [3].

Problem 1. *Given a set S of m PSD matrices $M_1, \cdots, M_m$ with $\sum_{i=1}^m M_i = I$ and $0 < \varepsilon < 1$, find non-negative coefficients $\{c_i\}_{i=1}^m$ such that $|\{c_i | c_i \neq 0\}| = O\left(n/\varepsilon^2\right)$, and*

$$(1 - \varepsilon) \cdot I \preceq \sum_{i=1}^m c_i M_i \preceq (1 + \varepsilon) \cdot I. \quad (2)$$

For intuition, one can think all M_i are rank-1 matrices, i.e., $M_i = v_i v_i^{\intercal}$ for some $v_i \in \mathbb{R}^n$. Given the correspondence between PSD matrices and ellipsoids, Problem 1 essentially asks to use $O(n/\varepsilon^2)$ vectors from S to construct an ellipsoid, whose *shape* is close to be a sphere. To construct such an ellipsoid with desired shape, all previous algorithms [2, 3, 10] proceed by iterations: in each iteration j the algorithm chooses one or more vectors, denoted by $v_{j_1}, \cdots, v_{j_k}$, and adds $\Delta_j \triangleq \sum_{t=1}^k v_{j_t} v_{j_t}^{\intercal}$ to the currently constructed matrix by setting $A_j = A_{j-1} + \Delta_j$. To control the shape of the constructed ellipsoid, two barrier values, the *upper barrier* u_j and the *lower barrier* ℓ_j, are maintained such that the constructed ellipsoid $\mathsf{Ellip}(A_j)$ is sandwiched between the *outer* sphere $u_j \cdot I$ and the *inner* sphere $\ell_j \cdot I$ for any iteration j. That is, the following invariant always maintains:

$$\ell_j \cdot I \prec A_j \prec u_j \cdot I. \quad (3)$$

To ensure (3) holds, two barrier values ℓ_j and u_j are increased properly after each iteration, i.e.,

$$u_{j+1} = u_j + \delta_{u,j}, \qquad \ell_{j+1} = \ell_j + \delta_{\ell,j}$$

for some positive values $\delta_{u,j}$ and $\delta_{\ell,j}$. The algorithm continues this process, until after T iterations $\mathsf{Ellip}(A_T)$ is close to be a sphere. This implies that A_T is a solution of Problem 1, see Figure 2 for an illustration.

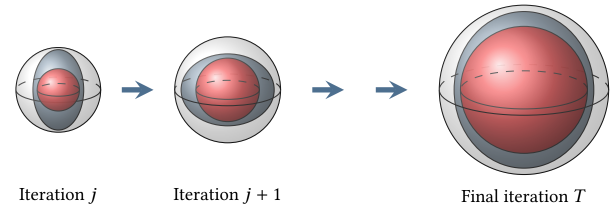

Figure 2: Illustration of the algorithms for constructing a linear-sized spectral sparsifier. Here, the light grey ball and the red ball in iteration j represent the spheres $u_j \cdot I$ and $\ell_j \cdot I$, and the blue ellipsoid sandwiched between the two balls corresponds to the constructed ellipsoid in iteration j. After each iteration j, the algorithm increases the value of ℓ_j and u_j by some $\delta_{\ell,j}$ and $\delta_{u,j}$ so that the invariant (3) holds in iteration $j+1$. This process is repeated for T iterations, so that the final constructed ellipsoid is close to be a sphere.

However, turning the scheme described above into an efficient algorithm we need to consider the following issues:

- Which vectors should we pick in each iteration?
- How many vectors can be added in each iteration?
- How should we update u_j and ℓ_j properly so that the invariant (3) always holds?

These three questions closely relate to each other: on one hand, one can always pick a single "optimal" vector in each iteration based on some metric, and such conservative approach requires a linear number of iterations $T = \Omega(n/\varepsilon^2)$ and super-quadric time for each iteration. On the other hand, one can choose multiple less "optimal" vectors to construct Δ_j in iteration j, but this makes the update of barrier values more challenging to ensure the invariant (3) holds. Indeed, the previous constructions [2, 10] speed up their algorithms at the cost of increasing the sparsity, i.e., the number of edges in a sparsifier, by more than a multiplicative constant.

To address these, we introduce three novel techniques for constructing a spectral sparsifier: First of all, we define a new potential function which is much easier to compute yet has similar guarantee as the potential function introduced in [3]. Secondly we show that solving Problem 1 with two-sided constraints in (2) can be reduced to a similar problem with only one-sided constraints. Thirdly we prove that the problem with one-sided constraints can be solved by a semi-definite program.

3.1 A New Potential Function

To ensure that the constructed ellipsoid A is always inside the outer sphere $u \cdot I$, we introduce a potential function $\Phi_u(A)$ defined by

$$\Phi_u(A) \triangleq \mathrm{tr}\exp\left((uI - A)^{-1}\right) = \sum_{i=1}^n \exp\left(\frac{1}{u - \lambda_i(A)}\right).$$

It is easy to see that, when $\mathsf{Ellip}(A)$ gets closer to the outer sphere, $\lambda_i(u \cdot I - A)$ becomes smaller and the value of $\Phi_u(A)$ increases. Hence, a bounded value of $\Phi_u(A)$ ensures that $\mathsf{Ellip}(A)$ is inside the sphere $u \cdot I$. For the same reason, we introduce a potential function

$\Phi_\ell(A)$ defined by

$$\Phi_\ell(A) \triangleq \operatorname{tr}\exp\left((A-\ell I)^{-1}\right) = \sum_{i=1}^{n} \exp\left(\frac{1}{\lambda_i(A)-\ell}\right).$$

to ensure that the inner sphere is always inside $\mathsf{Ellip}(A)$. We also define

$$\Phi_{u,\ell}(A) \triangleq \Phi_u(A) + \Phi_\ell(A), \tag{4}$$

as a bounded value of $\Phi_{u,\ell}(A)$ implies that the two events occur simultaneously. Our goal is to design a proper update rule to construct $\{A_j\}$ inductively, so that $\Phi_{u_j,\ell_j}(A_j)$ is monotone non-increasing after each iteration. Assuming this, a bounded value of the initial potential function guarantees that the invariant (3) always holds.

To analyse the change of the potential function, we first notice that

$$\Phi_{u,\ell}(A+\Delta) \geq \Phi_{u,\ell}(A) + \operatorname{tr}\left(\mathrm{e}^{(uI-A)^{-1}}(uI-A)^{-2}\Delta\right) - \operatorname{tr}\left(\mathrm{e}^{(A-\ell I)^{-1}}(A-\ell I)^{-2}\Delta\right)$$

by the convexity of the function $\Phi_{u,\ell}$. We prove that, as long as the matrix Δ satisfies $0 \preceq \Delta \preceq \delta(uI-A)^2$ and $0 \preceq \Delta \preceq \delta(A-\ell I)^2$ for some small δ, the first-order approximation gives a good approximation.

Lemma 3.1. *Let A be a symmetric matrix. Let u,ℓ be the barrier values such that $u-\ell \leq 1$ and $\ell I \prec A \prec uI$. Assume that $\Delta \succeq 0$, $\Delta \preceq \delta(uI-A)^2$ and $\Delta \preceq \delta(A-\ell I)^2$ for $\delta \leq 1/10$. Then, it holds that*

$$\Phi_{u,\ell}(A+\Delta) \leq \Phi_{u,\ell}(A) + (1+2\delta)\operatorname{tr}\left(\mathrm{e}^{(uI-A)^{-1}}(uI-A)^{-2}\Delta\right) - (1-2\delta)\operatorname{tr}\left(\mathrm{e}^{(A-\ell I)^{-1}}(A-\ell I)^{-2}\Delta\right).$$

We remark that this is not the first paper to use a potential function to guide the growth of the ellipsoid. In [3], two potential functions similar to

$$\Lambda_{u,\ell,p}(A) = \operatorname{tr}\left((uI-A)^{-p}\right) + \operatorname{tr}\left((A-\ell I)^{-p}\right) \tag{5}$$

are used with $p=1$. The main drawback is that $\Lambda_{u,\ell,1}$ does not differentiate the following two cases:

- Multiple eigenvalues of A are close to the boundary (both u and ℓ).
- One of the eigenvalues of A is very close to the boundary (either u or ℓ).

It is known that, when one of the eigenvalues of A is very close to the boundary, it is more difficult to find an "optimal" vector. It was shown in [2] that this problem can be alleviated by using $p \gg 1$. However, this choice of p makes the function $\Lambda_{u,\ell,p}$ less smooth and hence one has to take a smaller step size δ, as shown in the following lemma from [2].

Lemma 3.2 ([2]). *Let A be a symmetric matrix and Δ be a rank-1 matrix. Let u,ℓ be the barrier values such that $\ell I \prec A \prec uI$. Assume that $\Delta \succeq 0$, $\Delta \preceq \delta(uI-A)$ and $\Delta \preceq \delta(A-\ell I)$ for $\delta \leq 1/(10p)$ and $p \geq 10$. Then, it holds that*

$$\Lambda_{u,\ell,p}(A+\Delta) \leq \Lambda_{u,\ell,p}(A) + p(1+p\delta)\operatorname{tr}\left((uI-A)^{-(p+1)}\Delta\right) - p(1-p\delta)\operatorname{tr}\left((A-\ell I)^{-(p+1)}\Delta\right).$$

Notice that, comparing with the potential function (5), our new potential function (4) blows up much faster when the eigenvalues of A are closer to the boundaries ℓ and u. This allows the problem of finding an "optimal" vector much easier than using $\Lambda_{u,\ell,p}$. At the same time, we avoid the problem of taking a small step $\Delta \preceq 1/p \cdot (uI-A)$ by taking a "non-linear" step $\Delta \preceq (uI-A)^2$. Since there cannot be too many eigenvalues close to the boundaries, this "non-linear" step allows us to take a large step except on a few directions.

3.2 A Simple Construction Based on `Oracle`

The second technique we introduce is the reduction from a spectral sparsifier with two-sided constraints to the one with one-sided constraints. Geometrically, it is equivalent to require the constructed ellipsoid inside another ellipsoid, instead of being sandwiched between two spheres as depicted in Figure 2. Ideally, we want to reduce the two-sided problem to the following problem: for a set of PSD matrices $\mathcal{M} = \{M_i\}_{i=1}^m$ such that $\sum_{i=1}^m M_i \preceq I$, find a sparse representation $\Delta = \sum_{i=1}^m \alpha_i M_i$ with small $\mathrm{nnz}(\alpha)$ such that $\Delta \preceq I$. However, in the reduction we use such matrix $\Delta = \sum_{i=1}^m \alpha_i M_i$ to update A_j and we need the length of $\mathsf{Ellip}(\Delta)$ is large on the direction that $\mathsf{Ellip}(A_j)$ is small. To encode this information, we define the generalised one-sided problem as follows:

Definition 3.3 (One-sided Oracle). Let $\mathbf{0} \preceq B \preceq I$, $C_+ \succeq \mathbf{0}$, $C_- \succeq \mathbf{0}$ be symmetric matrices, $\mathcal{M} = \{M_i\}_{i=1}^m$ be a set of matrices such that $\sum_{i=1}^m M_i = I$. We call a randomised algorithm $\texttt{Oracle}\,(\mathcal{M}, B, C_+, C_-)$ a one-sided oracle with speed $S \in (0,1]$ and error $\varepsilon > 0$, if the output matrix $\Delta = \sum_{i=1}^m \alpha_i M_i$ of $\texttt{Oracle}\,(\mathcal{M}, B, C_+, C_-)$ satisfies the following:

1. $\mathrm{nnz}(\alpha) \leq \lambda_{\min}(B) \cdot \operatorname{tr}\left(B^{-1}\right)$.
2. $\Delta \preceq B$ and $\alpha_i \geq 0$ for all i.
3. $\mathbb{E}\left[C \bullet \Delta\right] \geq S \cdot \lambda_{\min}(B) \cdot \operatorname{tr}(C) - \varepsilon S \cdot \lambda_{\min}(B) \cdot \operatorname{tr}(C_{|\cdot|})$, where $C = C_+ - C_-$, and $C_{|\cdot|} = C_+ + C_-$.

We show in Section 3.3 the existence of a one-sided oracle with speed $S = \Omega(1)$ and error $\varepsilon = 0$, in which case the oracle only requires C as input, instead of C_+ and C_-. However, to construct such an oracle efficiently an additional error is introduced, which depends on $C_+ + C_-$.

For the main algorithm $\texttt{Sparsify}\,(\mathcal{M}, \varepsilon)$, we maintain the matrix A_j inductively as we discussed at the beginning of Section 3. By employing the $\oplus$ operator, we reduce the problem of constructing Δ_j with two-sided constraints to the problem of constructing $\Delta_j \oplus \Delta_j$ with one-sided constraints. We also use C to ensure that the length of $\mathsf{Ellip}(\Delta)$ is large on the direction where the length of $\mathsf{Ellip}(A_j)$ is small. See Algorithm 1 for formal description.

To analyse Algorithm 1, we use the fact that the returned Δ_j satisfies the preconditions of Lemma 3.1 and prove in Lemma 4.4 that

$$\mathbb{E}\left[\Phi_{u_{j+1},\ell_{j+1}}(A_{j+1})\right] \leq \Phi_{u_j,\ell_j}(A_j)$$

for any iteration j. Hence, with high probability the bounded ratio between u_T and ℓ_T after the final iteration T implies that the $\mathsf{Ellip}(A_T)$ is close to be a sphere. In particular, for any $\varepsilon < 1/20$, a $(1+O(\varepsilon))$-spectral sparsfier can be constructed by calling `Oracle` $O\left(\frac{\log^2 n}{\varepsilon^2 \cdot S}\right)$ times, which is described in the lemma below.

Algorithm 1 `Sparsify` $(\mathcal{M}, \varepsilon)$

$j = 0, A_0 = \mathbf{0}$
$\ell_0 = -\frac{1}{4}, u_0 = \frac{1}{4}$
while $u_j - \ell_j < 1$ **do**
  $B_j = (u_j I - A_j)^2 \oplus (A_j - \ell_j I)^2$
  $C_+ = (1-2\varepsilon)(A_j - \ell_j I)^{-2} \exp\left(A_j - \ell_j I\right)^{-1}$
  $C_- = (1+2\varepsilon)(u_j I - A_j)^{-2} \exp(u_j I - A_j)^{-1}$
  $\Delta_j \oplus \Delta_j = \texttt{Oracle}\left(\{M_i \oplus M_i\}_{i=1}^m, B_j, \frac{C_+ \oplus C_+}{2}, \frac{C_- \oplus C_-}{2}\right)$
  $A_{j+1} \leftarrow A_j + \varepsilon \cdot \Delta_j$
  $\delta_{u,j} = \varepsilon \cdot \frac{(1+2\varepsilon)(1+\varepsilon)}{1-4\varepsilon} \cdot S \cdot \lambda_{\min}(B_j)$
  $\delta_{\ell,j} = \varepsilon \cdot \frac{(1-2\varepsilon)(1-\varepsilon)}{1+4\varepsilon} \cdot S \cdot \lambda_{\min}(B_j)$
  $u_{j+1} \leftarrow u_j + \delta_{u,j}, \ell_{j+1} \leftarrow \ell_j + \delta_{\ell,j}$
  $j \leftarrow j+1$
end while
Return A_j

Lemma 3.4. *Let $0 < \varepsilon < 1/20$. Suppose we have one-sided oracle* `Oracle` *with speed S and error ε. Then, with constant probability the algorithm* `Sparsify` $(\mathcal{M}, \varepsilon)$ *outputs a $(1+O(\varepsilon))$-spectral sparsifier with $O\left(\frac{n}{\varepsilon^2 \cdot S}\right)$ vectors by calling* `Oracle` *$O\left(\frac{\log^2 n}{\varepsilon^2 \cdot S}\right)$ times.*

3.3 Solving `Oracle` via SDP

Now we show that the required solution of $\texttt{Oracle}(\mathcal{M}, B, C)$ indeed exists[3], and can be further solved in nearly-linear time by a semi-definite program. We first prove that the required matrix satisfying the conditions of Definition 3.3 exists for some absolute constant $S = \Omega(1)$ and $\varepsilon = 0$. To make a parallel discussion between Algorithm 1 and the algorithm we will present later, we use A to denote the output of the `Oracle` instead of Δ. We adopt the ideas between the ellipsoid and two spheres discussed before, but only consider one sphere for the one-sided case. Hence, we introduce a barrier value u_j for each iteration j, where $u_0 = 0$. We will use the potential function

$$\Psi_j = \operatorname{tr}\left(u_j B - A_j\right)^{-1}$$

in our analysis, where u_j is increased by $\delta_j \triangleq (\Psi_j \cdot \lambda_{\min}(B))^{-1}$ after iteration j. Moreover, since we only need to prove the existence of the required matrix $A = \sum_{i=1}^m \alpha_i M_i$, our process proceeds for T iterations, where *only* one vector is chosen in each iteration. To find a desired vector, we perform random sampling, where each matrix M_i is sampled with probability $\operatorname{prob}(M_i)$ proportional to $M_i \bullet C$, i.e.,

$$\operatorname{prob}(M_i) \triangleq \frac{(M_i \bullet C)^+}{\sum_{t=1}^m (M_t \bullet C)^+}, \tag{6}$$

where $x^+ \triangleq \max\{x, 0\}$. Notice that, since our goal is to construct A such that $\mathbb{E}[C \bullet A]$ is lower bounded by some threshold as stated in Definition 3.3, we should not pick any matrix M_i with $M_i \bullet C < 0$. This random sampling procedure is described in Algorithm 2, and the properties of the output matrix is summarised as follows.

[3] As the goal here is to prove the existence of `Oracle` with error $\varepsilon = 0$, the input here is C instead of C_+ and C_-.

Algorithm 2 `SolutionExistence` $(\mathcal{M}, B, C)$

$A_0 = \mathbf{0}$, $u_0 = 1$ and $T = \left\lfloor \lambda_{\min}(B) \cdot \operatorname{tr}(B^{-1}) \right\rfloor$
for $j = 0, 1, \ldots, T-1$ **do**
  repeat
    Sample a matrix M_t with probability $\operatorname{prob}(M_t)$
    Let $\Delta_j = (4\Psi_j \cdot \operatorname{prob}(M_t))^{-1} \cdot M_t$
  until $\Delta_j \preceq \frac{1}{2}(u_j B - A_j)$
  $A_{j+1} = A_j + \Delta_j$
  $\delta_j = (\Psi_j \cdot \lambda_{\min}(B))^{-1}$
  $u_{j+1} = u_j + \delta_j$
end for
Return $\frac{1}{u_T} A_T$

Lemma 3.5. *Let $\mathbf{0} \preceq B \preceq I$ and C be symmetric matrices, and $\mathcal{M} = \{M_i\}_{i=1}^m$ be a set of PSD matrices such that $\sum_{i=1}^m M_i = I$. Then* `SolutionExistence` $(\mathcal{M}, B, C)$ *outputs a matrix $A = \sum_{i=1}^m \alpha_i M_i$ such that the following holds:*

(1) $\operatorname{nnz}(\alpha) = \left\lfloor \lambda_{\min}(B) \cdot \operatorname{tr}(B^{-1}) \right\rfloor$.
(2) *$A \preceq B$, and $\alpha_i \geq 0$ for all i.*
(3) $\mathbb{E}\left[C \bullet A\right] \geq \frac{1}{32} \cdot \lambda_{\min}(B) \cdot \operatorname{tr}(C)$.

Lemma 3.5 shows that the required matrix A defined in Definition 3.3 exists, and can be found by random sampling described in Algorithm 2. Our key observation is that such matrix A can be constructed by a semi-definite program.

Theorem 3.6. *Let $\mathbf{0} \preceq B \preceq I$, C be symmetric matrices, and $\mathcal{M} = \{M_i\}_{i=1}^m$ be a set of matrices such that $\sum_{i=1}^m M_i = I$. Let $S \subseteq [m]$ be a random set of $\left\lfloor \lambda_{\min}(B) \cdot \operatorname{tr}(B^{-1}) \right\rfloor$ coordinates, where every index i is picked with probability $\operatorname{prob}(M_i)$. Let $A^\star$ be the solution of the following semidefinite program*

$$\max_{\alpha_i \geq 0} C \bullet \left(\sum_{i \in S} \alpha_i M_i\right) \text{ subject to } A = \sum_{i \in S} \alpha_i M_i \preceq B. \tag{7}$$

Then, we have $\mathbb{E}\left[C \bullet A^\star\right] \geq \frac{1}{32} \cdot \lambda_{\min}(B) \cdot \operatorname{tr}(C)$.

Taking the SDP formuation (7) and the specific constraints of the `Oracle`'s input into account, the next lemma shows that the required matrix used in each iteration of `Sparsify` $(\mathcal{M}, \varepsilon)$ can be computed efficiently by solving a semidefinite program.

Lemma 3.7. *The Oracle used in Algorithm* `Sparsify` $(\mathcal{M}, \varepsilon)$ *can be implemented in*

$$\widetilde{O}\left((Z + n^{\omega}) \cdot \varepsilon^{-O(1)}\right) \quad \text{work} \quad \text{and} \quad \widetilde{O}\left(\varepsilon^{-O(1)}\right) \text{ depth,}$$

where $Z = \sum_{i=1}^m \operatorname{nnz}(M_i)$ is the total number of non-zeros in M_i. When the matrix $\sum_{i=1}^m M_i = I$ comes from spectral sparsification of graphs, each iteration of `Sparsify` $(\mathcal{M}, \varepsilon)$ *can be implemented in*

$$\widetilde{O}\left(m\varepsilon^{-O(1)}\right) \text{ work} \quad \text{and} \quad \widetilde{O}\left(\varepsilon^{-O(1)}\right) \text{ depth.}$$

Furthermore, the speed of this one-sided oracle is $\Omega(1)$ and the error of this one-sided oracle is ε.

Combining Lemma 3.4 and Lemma 3.7 gives us the proof of the main result.

Proof of Theorem 1.1 and Theorem 1.2. Lemma 3.7 shows that we can construct `Oracle` with $\Omega(1)$ speed and ε error that runs in $\widetilde{O}\left((Z+n^{\omega})\cdot\varepsilon^{-O(1)}\right)$ work and $\widetilde{O}\left(\varepsilon^{-O(1)}\right)$ depth for the matrix setting, and $\widetilde{O}\left(m\varepsilon^{-O(1)}\right)$ work and $\widetilde{O}\left(\varepsilon^{-O(1)}\right)$ depth for the graph setting. Combining this with Lemma 3.4, which states that it suffices to call `Oracle` $\widetilde{O}\left(1/\varepsilon^2\right)$ times, the main statements hold. □

3.4 Further Discussion

Before presenting a more detailed analysis of our algorithm, we compare our new approach with the previous ones for constructing a linear-sized spectral sparsifier, and see how we address the bottlenecks faced in previous constructions. Notice that all previous algorithms require super poly-logarithmic number of iterations, and super linear-time for each iteration. For instance, our previous algorithm [10] for constructing a sparsifier with $O(pn)$ edges requires $\Omega(n^{1/p})$ iterations and $\Omega(n^{1+1/p})$ time per iteration for the following reasons:

- $\Omega\left(n^{1+1/p}\right)$ time is needed per iteration: Each iteration takes $n/g^{\Omega(1)}$ time to pick the vector(s) when $(\ell+g)I \preceq A \preceq (u-g)I$. To avoid eigenvalues of A getting too close to the boundary u or ℓ, i.e., g being too small, we choose the potential function whose value dramatically increases when the eigenvalues of A get close u or ℓ. As the cost, we need to scale down the added vectors by an $n^{1/p}$ factor.
- $\Omega\left(n^{1/p}\right)$ iterations are needed: By random sampling, we choose $O\left(n^{1-1/p}\right)$ vectors each iteration and use the matrix Chernoff bound to show that the "quality" of added $O\left(n^{1-1/p}\right)$ vectors is just $p=\Theta(1)$ times worse than adding a single vector. Hence, this requires $\Omega\left(n^{1/p}\right)$ iterations.

In contrast, our new approach breaks these two barriers through the following way:

- A "non-linear" step: Instead of rescaling down the vectors we add uniformly, we pick much fewer vectors on the direction that blows up, i.e., we impose the condition $\Delta \preceq (uI-A)^2$ instead of $\Delta \preceq 1/p\cdot(uI-A)$. This allows us to use the new potential function (4) with form $\exp\left(x^{-1}\right)$ to control the eigenvalues in a more aggressive way.
- SDP filtering: By matrix Chernoof bound, we know that the probability that we sample a few "bad" vectors is small. Informally, we apply semi-definite programming to filter out those bad vectors, and this allows us to add $\Omega\left(n/\log^{O(1)}(n)\right)$ vectors in each iteration.

4 DETAILED ANALYSIS

In this section we give detailed analysis for the statements presented in Section 3.

4.1 Analysis of the Potential Function

Now we analyse the properties of the potential function (4), and prove Lemma 3.1. The following two facts from matrix analysis will be used in our analysis.

Lemma 4.1 (Woodbury Matrix Identity). *Let $A\in\mathbb{R}^{n\times n}$, $U\in\mathbb{R}^{n\times k}$, $C\in\mathbb{R}^{k\times k}$ and $V\in\mathbb{R}^{k\times n}$ be matrices. Suppose that A, C and $C^{-1}+VA^{-1}U$ are invertible, it holds that*

$$(A+UCV)^{-1}=A^{-1}-A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}.$$

Lemma 4.2 (Golden-Thompson Inequality). *It holds for any symmetric matrices A and B that*

$$\operatorname{tr}\left(\mathrm{e}^{A+B}\right)\leq\operatorname{tr}\left(\mathrm{e}^{A}\cdot\mathrm{e}^{B}\right).$$

Proof of Lemma 3.1. We analyse the change of $\Phi_u(\cdot)$ and $\Phi_\ell(\cdot)$ individually. First of all, notice that

$$\begin{aligned}&(uI-A-\Delta)^{-1}\\&=(uI-A)^{-1/2}\left(I-(uI-A)^{-1/2}\Delta(uI-A)^{-1/2}\right)^{-1}(uI-A)^{-1/2}.\end{aligned}$$

We define

$$\Pi=(uI-A)^{-1/2}\Delta(uI-A)^{-1/2}.$$

Since $0\preceq\Delta\preceq\delta(uI-A)^2$ and $u-\ell\leq 1$, it holds that

$$\Pi\preceq\delta(uI-A)\preceq\delta(uI-\ell I)\preceq\delta I,$$

and therefore

$$(I-\Pi)^{-1}\preceq I+\frac{1}{1-\delta}\cdot\Pi.$$

Hence, it holds that

$$\begin{aligned}(uI-A-\Delta)^{-1}&\preceq(uI-A)^{-1/2}\left(I+\frac{1}{1-\delta}\cdot\Pi\right)(uI-A)^{-1/2}\\&=(uI-A)^{-1}+\frac{1}{1-\delta}\cdot(uI-A)^{-1}\Delta(uI-A)^{-1}.\end{aligned}$$

By the fact that $\operatorname{tr}\exp$ is monotone and the Golden-Thompson inequality (Lemma 4.2), we have that

$$\begin{aligned}&\Phi_u(A+\Delta)\\&=\operatorname{tr}\exp\left((uI-A-\Delta)^{-1}\right)\\&\leq\operatorname{tr}\exp\left((uI-A)^{-1}+\frac{1}{1-\delta}\cdot(uI-A)^{-1}\Delta(uI-A)^{-1}\right)\\&\leq\operatorname{tr}\left(\exp(uI-A)^{-1}\exp\left(\frac{1}{1-\delta}\cdot(uI-A)^{-1}\Delta(uI-A)^{-1}\right)\right).\end{aligned}$$

Since $0\preceq\Delta\preceq\delta(uI-A)^2$ and $\delta\leq 1/10$ by assumption, we have that $(uI-A)^{-1}\Delta(uI-A)^{-1}\preceq\delta I$, and

$$\begin{aligned}&\exp\left(\frac{1}{1-\delta}\cdot(uI-A)^{-1}\Delta(uI-A)^{-1}\right)\\&\preceq I+(1+2\delta)\cdot(uI-A)^{-1}\Delta(uI-A)^{-1}.\end{aligned}$$

Hence, it holds that

$$\begin{aligned}&\Phi_u(A+\Delta)\\&\leq\operatorname{tr}\left(\mathrm{e}^{(uI-A)^{-1}}\cdot\left(I+(1+2\delta)(uI-A)^{-1}\Delta(uI-A)^{-1}\right)\right)\\&=\Phi_u(A)+(1+2\delta)\cdot\operatorname{tr}(\mathrm{e}^{(uI-A)^{-1}}(uI-A)^{-2}\Delta).\end{aligned}$$

By the same analysis, we have that

$$\begin{aligned}&\Phi_\ell(A+\Delta)\\&\leq\operatorname{tr}\left(\mathrm{e}^{(A-\ell I)^{-1}}\cdot\left(I-(1-2\delta)(A-\ell I)^{-1}\Delta(A-\ell I)^{-1}\right)\right)\\&=\Phi_\ell(A)-(1-2\delta)\cdot\operatorname{tr}(\mathrm{e}^{(A-\ell I)^{-1}}(A-\ell I)^{-2}\Delta).\end{aligned}$$

Combining the analysis on $\Phi_u(A+\Delta)$ and $\Phi_\ell(A+\Delta)$ finishes the proof. □

Lemma 4.3. *Let A be a symmetric matrix. Let u, ℓ be the barrier values such that $u - \ell \leq 1$ and $\ell I \prec A \prec uI$. Assume that $0 \leq \delta_u \leq \delta \cdot \lambda_{\min}(uI - A)^2$ and $0 \leq \delta_\ell \leq \delta \cdot \lambda_{\min}(A - \ell I)^2$ for $\delta \leq 1/10$. Then, it holds that*

$$\Phi_{u+\delta_u, \ell+\delta_\ell}(A) \leq \Phi_{u,\ell}(A) - (1-2\delta)\delta_u \cdot \operatorname{tr}\left(\mathrm{e}^{(uI-A)^{-1}}(uI-A)^{-2}\right) + (1+2\delta)\delta_\ell \cdot \operatorname{tr}\left(\mathrm{e}^{(A-\ell I)^{-1}}(A-\ell I)^{-2}\right).$$

Proof. Since $0 \leq \delta_u \leq \delta \cdot \lambda_{\min}(uI - A)^2$ and $0 \leq \delta_\ell \leq \delta \cdot \lambda_{\min}(A - \ell I)^2$, we have that $\delta_u \cdot I \preceq \delta \cdot (uI - A)^2$ and $\delta_\ell \cdot I \preceq \delta \cdot (A - \ell I)^2$. The statement follows by a similar analysis for proving Lemma 3.1. □

4.2 Analysis of the Reduction

Now we present the detailed analysis for the reduction from a spectral sparsifier to a one-sided oracle. We first analyse Algorithm 1, and prove that in expectation the value of the potential function is not increasing. Based on this fact, we will give a proof of Lemma 3.4, which shows that a $(1+O(\varepsilon))$-spectral sparsifier can be constructed by calling `Oracle` $O\left(\frac{\log^2 n}{\varepsilon^2 \cdot S}\right)$ times.

Lemma 4.4. *Let A_j and A_{j+1} be the matrices constructed by Algorithm 1 in iteration j and $j+1$, and assume that $0 \leq \varepsilon \leq 1/20$. Then, it holds that*

$$\mathbb{E}\left[\Phi_{u_{j+1},\ell_{j+1}}(A_{j+1})\right] \leq \Phi_{u_j,\ell_j}(A_j).$$

Proof. By the description of Algorithm 1 and Definition 3.3, it holds that $\Delta_j \oplus \Delta_j \preceq (u_j I - A_j)^2 \oplus (A_j - \ell_j I)^2$, which implies that $\Delta_j \preceq (u_j I - A_j)^2$ and $\Delta_j \preceq (A_j - \ell_j I)^2$. Since $u_j - \ell_j \leq 1$ by the algorithm description and $0 \leq \varepsilon \leq 1/20$, by setting $\Delta = \varepsilon \cdot \Delta_j$ in Lemma 3.1, we have

$$\begin{aligned}
&\Phi_{u_j,\ell_j}(A_j + \varepsilon \cdot \Delta_j)\\
&\leq \Phi_{u_j,\ell_j}(A_j) + \varepsilon(1+2\varepsilon) \cdot \operatorname{tr}\left(\mathrm{e}^{(u_jI-A_j)^{-1}}(u_jI-A_j)^{-2}\Delta_j\right)\\
&\qquad - \varepsilon(1-2\varepsilon) \cdot \operatorname{tr}\left(\mathrm{e}^{(A_j-\ell_jI)^{-1}}(A_j-\ell_jI)^{-2}\Delta_j\right)\\
&= \Phi_{u_j,\ell_j}(A_j) - \varepsilon \cdot C \bullet \Delta_j.
\end{aligned}$$

Notice that the matrices $\{M_i \oplus M_i\}_{i=1}^m$ as the input of `Oracle` always satisfy $\sum_{i=1}^m M_i \oplus M_i = I \oplus I$. Using this and the definition of `Oracle`, we know that

$$\mathbb{E}\left[C \bullet \Delta_j\right] \geq S \cdot \lambda_{\min}(B_j) \cdot \operatorname{tr}(C) - \varepsilon \cdot S \cdot \lambda_{\min}(B_j) \cdot \operatorname{tr}\left(C_{|\cdot|}\right),$$

Let $\alpha_j = \varepsilon \cdot S \cdot \lambda_{\min}(B_j)$. Then we have that

$$\begin{aligned}
&\mathbb{E}\left[\Phi_{u_j,\ell_j}(A_{j+1})\right]\\
&\leq \Phi_{u_j,\ell_j}(A_j) - \varepsilon \cdot \mathbb{E}\left[C \bullet \Delta_j\right]\\
&\leq \Phi_{u_j,\ell_j}(A_j) + (1+2\varepsilon)(1+\varepsilon) \cdot \alpha_j \cdot \operatorname{tr}\left(\mathrm{e}^{(u_jI-A_j)^{-1}}(u_jI-A_j)^{-2}\right)\\
&\qquad - (1-2\varepsilon)(1-\varepsilon) \cdot \alpha_j \cdot \operatorname{tr}\left(\mathrm{e}^{(A_j-\ell_jI)^{-1}}(A_j-\ell_jI)^{-2}\right).
\end{aligned} \tag{8}$$

On the other hand, using that $0 \leq \varepsilon \leq 1/20$, $S \leq 1$ and $\Delta_j \preceq (u_j I - A_j)$, we have that

$$\delta_{u,j} \leq \varepsilon \cdot \frac{(1+2\varepsilon)(1+\varepsilon)}{1-4\varepsilon} \cdot \lambda_{\min}(u_jI - A_j)^2 \leq 2\varepsilon \cdot \lambda_{\min}(u_jI - A_{j+1})^2$$

and

$$\delta_{\ell,j} \leq \varepsilon \cdot \frac{(1-2\varepsilon)(1-\varepsilon)}{1+4\varepsilon} \cdot \lambda_{\min}(A_j - \ell_jI)^2 \leq 2\varepsilon \cdot \lambda_{\min}(A_{j+1} - \ell_jI)^2.$$

Hence, Lemma 4.3 shows that

$$\begin{aligned}
&\Phi_{u_j+\delta_{u,j},\ell_j+\delta_{\ell,j}}(A_{j+1})\\
&\leq \Phi_{u_j,\ell_j}(A_{j+1}) - (1-4\varepsilon)\delta_{u,j} \cdot \operatorname{tr}\left(\mathrm{e}^{(u_jI-A_{j+1})^{-1}}(u_jI-A_{j+1})^{-2}\right)\\
&\qquad + (1+4\varepsilon)\delta_{\ell,j} \cdot \operatorname{tr}\left(\mathrm{e}^{(A_{j+1}-\ell_jI)^{-1}}(A_{j+1}-\ell_jI)^{-2}\right)\\
&\leq \Phi_{u_j,\ell_j}(A_{j+1}) - (1-4\varepsilon)\delta_{u,j} \cdot \operatorname{tr}\left(\mathrm{e}^{(u_jI-A_j)^{-1}}(u_jI-A_j)^{-2}\right)\\
&\qquad + (1+4\varepsilon)\delta_{\ell,j} \cdot \operatorname{tr}\left(\mathrm{e}^{(A_j-\ell_jI)^{-1}}(A_j-\ell_jI)^{-2}\right).
\end{aligned} \tag{9}$$

By combining (8), (9), and setting

$$(1-4\varepsilon)\delta_{u,j} = (1+2\varepsilon)(1+\varepsilon)\alpha_j,$$
$$(1+4\varepsilon)\delta_{\ell,j} = (1-2\varepsilon)(1-\varepsilon)\alpha_j,$$

we have that

$$\mathbb{E}\left[\Phi_{u_{j+1},\ell_{j+1}}(A_{j+1})\right] \leq \Phi_{u_j,\ell_j}(A_j).$$

□

Proof of Lemma 3.4. We first bound the number of times the algorithm calls the oracle. Notice that

$$\Phi_{u_0,\ell_0} = 2 \cdot \operatorname{tr}\exp\left(\left(\frac{1}{4}I\right)^{-1}\right) = 2\mathrm{e}^4 \cdot n.$$

Hence, by Lemma 4.4 we have $\mathbb{E}\left[\Phi_{u_j,\ell_j}(A_j)\right] = O(n)$ for any iteration j. By Markov's inequality, it holds that $\Phi_{u_j,\ell_j}(A_j) = n^{O(1)}$ with high probability in n. In the remainder of the proof, we assume that this event occurs.

Since $B_j = (u_jI - A_j)^2 \oplus (A_j - \ell_jI)^2$ by definition, it holds that

$$\exp\left(\left(\lambda_{\min}\left(B_j\right)\right)^{-1/2}\right) \leq \Phi_{u_j,\ell_j}(A_j) = n^{O(1)},$$

which implies that

$$\lambda_{\min}\left(B_j\right) = \Omega\left(\log^{-2} n\right). \tag{10}$$

On the other hand, in iteration j the gap between u_j and ℓ_j is increased by

$$\delta_{u,j} - \delta_{\ell,j} = \Omega\left(\varepsilon^2 \cdot S \cdot \lambda_{\min}(B_j)\right). \tag{11}$$

Combining this with (10) gives us that

$$\delta_{u,j} - \delta_{\ell,j} = \Omega\left(\frac{\varepsilon^2 \cdot S}{\log^2 n}\right)$$

for any j. Since $u_0 - \ell_0 = 1/2$ and the algorithm terminates once $u_j - \ell_j > 1$ for some j, with high probability in n, the algorithm terminates in $O\left(\frac{\log^2 n}{\varepsilon^2 \cdot S}\right)$ iterations.

Next we prove that the number of M_i's involved in the output is at most $O\left(\frac{n}{\varepsilon^2 \cdot S}\right)$. By the properties of `Oracle`, the number of matrices in iteration j is at most $\lambda_{\min}(B_j) \cdot \operatorname{tr}(B_j^{-1})$. Since $x^{-2} \leq \exp\left(x^{-1}\right)$ for all $x > 0$, it holds for any iteration j that

$$\operatorname{tr}\left(B_j^{-1}\right) = \operatorname{tr}\left(\left(u_jI - A_j\right)^{-2}\right) + \operatorname{tr}\left(\left(A_j - \ell_jI\right)^{-2}\right) \leq \Phi_{u_j,\ell_j}(A_j)$$

By (11), we know that for added matrix M_i in iteration j, the average increase of the gap $u_j - \ell_j$ for each added matrix is $\Omega\left(\frac{\varepsilon^2 \cdot S}{\Phi_{u_j,\ell_j}(A_j)}\right)$. Since $\mathbb{E}\left[\Phi_{u_j,\ell_j}(A_j)\right] = O(n)$, for every new added matrix, in expectation the gap between u_j and ℓ_j is increased by $\Omega\left(\frac{\varepsilon^2 \cdot S}{n}\right)$. By the ending condition of the algorithm, i.e., $u_j - \ell_j > 1$, and Markov's inequality, the number of matrices picked in total is at most $O\left(\frac{n}{\varepsilon^2 \cdot S}\right)$ with constant probability.

Finally we prove that the output is a $(1+O(\varepsilon))$-spectral sparsifier. Since the condition number of the output matrix A_j is at most

$$\frac{u_j}{\ell_j} = \left(1 - \frac{u_j - \ell_j}{u_j}\right)^{-1},$$

it suffices to prove that $(u_j - \ell_j)/u_j = O(\varepsilon)$ and this easily follows from the ending condition of the algorithm and

$$\frac{\delta_{u,j} - \delta_{\ell,j}}{\delta_{u,j}} = O(\varepsilon). \qquad \square$$

4.3 Existence Proof for `Oracle`

Proof of Lemma 3.5. The property on nnz (α) follows from the algorithm description. For the second property, notice that every chosen matrix Δ_j in iteration j satisfies $\Delta_j \preceq \frac{1}{2}(u_j B - A_j)$, which implies that $A_j \preceq u_j B$ holds for any iteration j. Hence,

$$A = \frac{1}{u_T} A_T \preceq B,$$

and $\alpha_i \geq 0$ since $\Psi_j \geq 0$.

Now we prove the third statement. Let

$$\beta = \sum_{i=1}^{m} (M_i \bullet C)^+ .$$

Then, for each matrix M_{i_j} picked in iteration j, $C \bullet A_j$ is increased by

$$C \bullet \Delta_j = \frac{1}{4\Psi_j \cdot \mathrm{prob}(M_{i_j})} \cdot C \bullet M_{i_j} = \frac{\beta}{4\Psi_j}.$$

On the other hand, it holds that

$$u_T = u_0 + \sum_{j=0}^{T-1} \delta_j = 1 + \sum_{j=0}^{T-1} \left(\Psi_j \cdot \lambda_{\min}(B)\right)^{-1}.$$

Hence, we have that

$$\begin{aligned} C \bullet A &= \frac{1}{u_T} \cdot C \bullet \left(\sum_{j=0}^{T-1} \Delta_j\right) = \frac{\sum_{j=0}^{T-1} \beta \cdot (4\Psi_j)^{-1}}{1 + \sum_{j=0}^{T-1} (\Psi_j \cdot \lambda_{\min}(B))^{-1}} \\ &= \frac{\beta \lambda_{\min}(B)}{4} \cdot \frac{\sum_{j=0}^{T-1} \Psi_j^{-1}}{\lambda_{\min}(B) + \sum_{j=0}^{T-1} \Psi_j^{-1}} \\ &\geq \frac{\beta \lambda_{\min}(B)}{4} \cdot \frac{\sum_{j=0}^{T-1} (\Psi_j + \Psi_0)^{-1}}{\lambda_{\min}(B) + \sum_{j=0}^{T-1} (\Psi_j + \Psi_0)^{-1}} \\ &\geq \frac{\beta \lambda_{\min}(B)}{4} \cdot \frac{\sum_{j=0}^{T-1} (\Psi_j + \Psi_0)^{-1}}{\lambda_{\min}(B) + T \cdot \Psi_0^{-1}} \\ &\geq \frac{\beta}{8} \cdot \sum_{j=0}^{T-1} \left(\Psi_j + \Psi_0\right)^{-1}, \end{aligned} \tag{12}$$

where the last inequality follows by the choice of T. Hence, it suffices to bound Ψ_j.

Since $\Delta_j \preceq \frac{1}{2}(u_j B - A_j) \preceq \frac{1}{2}(u_{j+1} B - A_j)$, we have that

$$\Psi_{j+1} \leq \mathrm{tr}\left((u_{j+1}B - A_j)^{-1}\right) + 2 \cdot \Delta_j \bullet (u_{j+1}B - A_j)^{-2}. \tag{13}$$

Since $\mathrm{tr}(uB - A_j)^{-1}$ is convex in u, we have that

$$\begin{aligned} &\mathrm{tr}\left((u_j B - A_j)^{-1}\right) \\ &\geq \mathrm{tr}\left((u_{j+1}B - A_j)^{-1}\right) + \delta_j \mathrm{tr}\left((u_{j+1}B - A_j)^{-2} B\right) \end{aligned} \tag{14}$$

Combining (13) and (14), we have that

$$\begin{aligned} \Psi_{j+1} &\leq \mathrm{tr}\left(\left(u_j B - A_j\right)^{-1}\right) - \delta_j \cdot \mathrm{tr}\left(\left(u_{j+1}B - A_j\right)^{-2} B\right) \\ &\qquad + 2 \cdot \Delta_j \bullet (u_{j+1}B - A_j)^{-2} \\ &= \Psi_j - \delta_j \cdot \lambda_{\min}(B) \cdot \mathrm{tr}\left((u_{j+1}B - A_j)^{-2}\right) \\ &\qquad + 2 \cdot \Delta_j \bullet (u_{j+1}B - A_j)^{-2} \end{aligned} \tag{15}$$

Let $\mathcal{E}_j$ be the event that $\Delta_j \preceq \frac{1}{2}(u_j B - A_j)$. Notice that our picked Δ_j in each iteration always satisfies $\mathcal{E}_j$ by algorithm description. Since

$$\begin{aligned} &\mathbb{E}\left[\Delta_j \bullet (u_j B - A_j)^{-1}\right] \\ &= \sum_{i:(M_i \bullet C)^+ > 0} \mathrm{prob}(M_i) \cdot \frac{1}{4\Psi_j \cdot \mathrm{prob}(M_i)} \cdot M_i \bullet (u_j B - A_j)^{-1} \\ &\leq \frac{1}{4}, \end{aligned}$$

by Markov inequality it holds that

$$\mathbb{P}\left[\mathcal{E}_j\right] = \mathbb{P}\left[\Delta_j \preceq \frac{1}{2}(u_j B - A_j)\right] \geq \frac{1}{2},$$

and therefore

$$\begin{aligned} &\mathbb{E}\left[\Delta_j \bullet (u_{j+1}B - A_j)^{-2} \mid \mathcal{E}_j\right] \\ &\leq \frac{\mathbb{E}\left[\Delta_j \bullet (u_{j+1}B - A_j)^{-2}\right]}{\mathbb{P}\left(\mathcal{E}_j\right)} \\ &\leq 2 \cdot \mathbb{E}\left[\Delta_j \bullet (u_{j+1}B - A_j)^{-2}\right] \\ &= \frac{1}{2\Psi_j} \cdot \sum_{i:(M_i \bullet C)^+ > 0} M_i \bullet (u_{j+1}B - A_j)^{-2} \\ &\leq \frac{1}{2\Psi_j} \cdot \mathrm{tr}(u_{j+1}B - A_j)^{-2}. \end{aligned}$$

Combining the inequality above, (15), and the fact that every Δ_j picked by the algorithm satisfies $\mathcal{E}$, we have that

$$\mathbb{E}\left[\Psi_{j+1}\right] \leq \Psi_j + \left(\frac{1}{\Psi_j} - \delta_j \cdot \lambda_{\min}(B)\right) \cdot \mathrm{tr}\left(u_{j+1}B - A_j\right)^{-2}.$$

By our choice of δ_j, it holds for any iteration j that $\mathbb{E}\left[\Psi_{j+1}\right] \leq \Psi_j$, and

$$\mathbb{E}\left[\left(\Psi_{j+1} + \Psi_0\right)^{-1}\right] \geq \mathbb{E}\left(\Psi_j + \Psi_0\right)^{-1} \geq \frac{1}{2 \cdot \Psi_0}.$$

Combining this with (12), it holds that

$$\mathbb{E}\left[C \bullet A\right] \geq \frac{\beta}{8}\sum_{j=0}^{T-1}\mathbb{E}\left[(\Psi_j+\Psi_0)^{-1}\right] \geq \frac{\beta}{16}\cdot\frac{T}{\Psi_0}$$
$$= \frac{\beta}{16}\cdot\frac{T}{\mathrm{tr}\left(B^{-1}\right)} \geq \frac{\mathrm{tr}(C)}{16}\cdot\frac{T}{\mathrm{tr}\left(B^{-1}\right)}.$$

The result follows from the fact that $T \geq \lambda_{\min}(B)\mathrm{tr}\left(B^{-1}\right)/2$. □

Using the lemma above, we can prove that such A can be solved by a semidefinite program.

Proof of Theorem 3.6. Note that the probability we used in the statement is the same as the one used in `SolutionExistence` $(\mathcal{M}, B, C)$. Therefore, Lemma 3.5 shows that there is a matrix A of the form $\sum_{i=1}^m \alpha_i M_i$ such that

$$\mathbb{E}\left[C \bullet A\right] \geq \frac{1}{32}\cdot\lambda_{\min}(B)\cdot\mathrm{tr}\left(C\right),$$

The statement follows by the fact that $A^\star$ is the solution of the semidefinite program (7) that maximises $C \bullet A$. □

4.4 Implementing the SDP in Nearly-Linear Time

Now, we discuss how to solve the SDP (7) in nearly-linear time. Since this SDP is a packing SDP, it is known how to solve it in nearly-constant depth [1, 8, 14]. The following result will be used in our analysis.

Theorem 4.5 ([1]). *Given a SDP*

$$\max_{x\geq 0} c^\intercal x \text{ subject to } \sum_{i=1}^m x_i A_i \preceq B$$

with $A_i \succeq \mathbf{0}$, $B \succeq \mathbf{0}$ and $c \in \mathbb{R}^m$. Suppose that we are given a direct access to the vector $c \in \mathbb{R}^m$ and an indirect access to A_i and B via an oracle $\mathcal{O}_{L,\delta}$ which inputs a vector $x \in \mathbb{R}^m$ and outputs a vector $v \in \mathbb{R}^m$ such that

$$v_i \in \left(1\pm\frac{\delta}{2}\right)\left[A_i \bullet B^{-1/2}\exp\left(L\cdot B^{-1/2}\left(\sum_i x_iA_i - B\right)B^{-1/2}\right)B^{-1/2}\right]$$

in $\mathcal{W}_{L,\delta}$ work and $\mathcal{D}_{L,\delta}$ depth for any x such that $x_i \geq 0$ and $\sum_{i=1}^m x_i A_i \preceq 2B$. Then, we can output x such that

$$\mathbb{E}\left[c^\intercal x\right] \geq (1-O(\delta))\mathsf{OPT} \quad \textit{with} \quad \sum_{i=1}^m x_iA_i \preceq B$$

in

$$O\left(\mathcal{W}_{L,\delta}\log m\cdot\log\left(nm/\delta\right)/\delta^3\right)$$

work, and

$$O\left(\mathcal{D}_{L,\delta}\log m\cdot\log(nm/\delta)/\delta\right)$$

depth, where $L = (4/\delta)\cdot\log(nm/\delta)$.

Since we are only interested in a fast implementation of the one-sided oracle used in `Sparsify` $(\mathcal{M}, \varepsilon)$, it suffices to solve the SDP (7) for this particular situation.

Proof of Lemma 3.7. Our basic idea is to use Theorem 4.5 as the one-sided oracle. Notice that each iteration of `Sparsify` $(\mathcal{M}, \varepsilon)$ uses the one-sided oracle with the input

$$C_+ = \frac{1-2\varepsilon}{2}\left((A-\ell I)^{-2}\exp(A-\ell I)^{-1}\oplus(A-\ell I)^{-2}\exp(A-\ell I)^{-1}\right),$$

$$C_- = \frac{1+2\varepsilon}{2}\left((uI-A)^{-2}\exp(uI-A)^{-1}\oplus(uI-A)^{-2}\exp(uI-A)^{-1}\right),$$

and

$$B = (uI-A)^2\oplus(A-\ell I)^2,$$

where we drop the subscript j indicating iterations here for simplicity. To apply Theorem 3.6, we first sample a subset $S \subseteq [m]$, then solve the SDP

$$\max_{\beta_i\geq 0,\,\beta_i=0 \text{ on } i\notin S}(C_+-C_-)\bullet\left(\sum_{i=1}^m \beta_iM_i\oplus M_i\right)$$

subject to

$$\sum_{i=1}^m \beta_iM_i\oplus M_i \preceq B.$$

By ignoring the matrices M_i with $i \notin S$, this SDP is equivalent to the SDP

$$\max_{\beta_i\geq 0} c^\top\beta \quad \text{subject to } \sum_{i=1}^m \beta_iM_i\oplus M_i \preceq B$$

where $c_i = (C_+-C_-)\bullet(M_i\oplus M_i)$. Now assume that (i) we can approximate c_i with $\delta(C_++C_-)\bullet(M_i\oplus M_i)$ additive error, and (ii) for any x such that $\sum_{i=1}^m x_iM_i\oplus M_i \preceq 2B$ and $L = \widetilde{O}(1/\delta)$, we can approximate

$$(M_i\oplus M_i)\bullet B^{-1/2}\exp\left(L\cdot B^{-1/2}\left(\sum_i x_iM_i\oplus M_i - B\right)B^{-1/2}\right)B^{-1/2}$$

with $1\pm\delta$ multiplicative error. Then, by Theorem 4.5 we can find a vector β such that

$$\mathbb{E}\left[c^\intercal\beta\right] \geq (1-O(\delta))\mathsf{OPT} - \delta\sum_{i=1}^m \beta_i(C_++C_-)\bullet(M_i\oplus M_i)$$
$$\geq (1-O(\delta))\mathsf{OPT} - \delta\sum_{i=1}^m \beta_i(C_++C_-)\bullet B$$
$$\geq \mathsf{OPT} - O(\delta)(C_++C_-)\bullet B.$$

where we used that $M_i\oplus M_i \preceq B$ and $\mathsf{OPT} \leq (C_++C_-)\bullet B$. Since $u-\ell \leq 1$, we have that $B \preceq I\oplus I$ and hence

$$\mathbb{E}\left[c^\intercal\beta\right] \geq \mathsf{OPT} - O(\delta)\cdot\mathrm{tr}(C_++C_-)$$
$$\geq \frac{1}{32}\lambda_{\min}(B)\cdot\left(\mathrm{tr}(C) - O(\delta\log^2 n)\cdot\mathrm{tr}(C_++C_-)\right)$$

where we apply Theorem 3.6 and (10) at the last line. Therefore, this gives an oracle with speed $1/32$ and ε error by setting $\delta = \varepsilon/\log^2 n$.

The problem of approximating sample probabilities, $\{c_i\}$, as well as implementing the oracle $\mathcal{O}_{L,\delta}$ is similar with approximating leverage scores [15], and relative leverage scores [2, 10]. All these references use the Johnson-Lindenstrauss lemma to reduce the problem of approximating matrix dot product or trace to matrix vector multiplication. The only difference is that, instead of computing $(A - \ell I)^{-(q+1)}x$ and $(uI - A)^{-(q+1)}x$ for a given vector x in other references, we compute $(A - \ell I)^{-2}\exp(A - \ell I)^{-1}x$ and $(uI - A)^{-2}\exp(uI - A)^{-1}x$. These can be approximated by Taylor expansion and the number of terms required for Taylor expansion depends on how close the eigenvalues of A are to the boundary (u or ℓ). In particular, we show in Section 4.5 that $\widetilde{O}(1/g^2)$ terms in Taylor expansion suffices, where the gap g is the largest number such that

$$(\ell + g)I \preceq A \preceq (u - g)I.$$

Since $1/g^2 = \widetilde{O}(1)$ by (10), each iteration can be implemented via solving $\widetilde{O}\left(1/\varepsilon^{O(1)}\right)$ linear systems and $\widetilde{O}\left(1/\varepsilon^{O(1)}\right)$ matrix vector multiplication. For the matrices coming from graph sparsification, this can be done in nearly-linear work and nearly-constant depth [9, 13]. For general matrices, this can be done in input sparsity time and nearly-constant depth [7, 11, 12]. □

4.5 Taylor Expansion of $x^{-2}\exp(x^{-1})$

Theorem 4.6 (Cauchy's Estimates). *Suppose f is holomorphic on a neighbourhood of the ball*

$$B \triangleq \{z \in \mathbb{C} \ : \ |z - s| \leq r\},$$

then it holds that

$$\left|f^{(k)}(s)\right| \leq \frac{k!}{r^k}\sup_{z\in B}|f(z)|.$$

Lemma 4.7. *Let $f(x) = x^{-2}\exp(x^{-1})$. For any $0 < x \leq 1$, we have that*

$$\left|f(x) - \sum_{k=0}^{d}\frac{1}{k!}f^{(k)}(1)(x-1)^k\right| \leq 8(d+1)e^{\frac{5}{x} - xd}.$$

In particular, if $d \geq \frac{c}{x^2}\log(\frac{1}{x\varepsilon})$ for some large enough universal constant c, we have that

$$\left|f(x) - \sum_{k=0}^{d}\frac{1}{k!}f^{(k)}(1)(x-1)^k\right| \leq \varepsilon.$$

Proof. By the formula of the remainder term in Taylor series, we have that

$$f(x) = \sum_{k=0}^{d}\frac{1}{k!}f^{(k)}(1)(x-1)^k + \frac{1}{d!}\int_1^x f^{(d+1)}(s)(x-s)^d ds.$$

For any $s \in [x, 1]$, we define

$$D(s) = \left\{z \in \mathbb{C} \ : \ |z - s| \leq s - \frac{x}{2}\right\}.$$

Since $|f(z)| \leq (x/2)^{-2}\exp(2/x)$ on $z \in D(s)$, Cauchy's estimates (Theorem 4.6) shows that

$$\begin{aligned}\left|f^{(d+1)}(s)\right| &\leq \frac{(d+1)!}{(s-\frac{x}{2})^{d+1}}\sup_{z\in B(s)}|f(z)| \\ &\leq \frac{(d+1)!}{(s-\frac{x}{2})^{d+1}}\frac{4}{x^2}\exp\left(\frac{2}{x}\right).\end{aligned}$$

Hence, we have that

$$\begin{aligned}&\left|f(x) - \sum_{k=0}^{d}\frac{1}{k!}f^{(k)}(t)(x-t)^k\right| \\ \leq&\frac{1}{d!}\left|\int_1^x \frac{(d+1)!}{(s-\frac{x}{2})^{d+1}}\frac{4}{x^2}\exp\left(\frac{2}{x}\right)(x-s)^d ds\right| \\ =&\frac{4(d+1)e^{\frac{2}{x}}}{x^2}\int_x^1\frac{(s-x)^d}{(s-\frac{x}{2})^{d+1}}ds \\ \leq&\frac{8(d+1)e^{\frac{2}{x}}}{x^3}\int_x^1(1-x)^d\,ds \\ \leq&8(d+1)\cdot e^{\frac{5}{x}-xd}.\end{aligned}$$

□

ACKNOWLEDGEMENT

The authors would like to thank Michael Cohen for helpful discussions and suggesting the ideas to improve the sparsity from $O\left(n/\varepsilon^3\right)$ to $O\left(n/\varepsilon^2\right)$.

REFERENCES

[1] Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. 2016. Using optimization to obtain a width-independent, parallel, simpler, and faster positive SDP solver. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*. 1824–1831.

[2] Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. 2015. Spectral Sparsification and Regret Minimization Beyond Multiplicative Updates. In *47th Annual ACM Symposium on Theory of Computing (STOC'15)*. 237–245.

[3] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. 2012. Twice-Ramanujan Sparsifiers. *SIAM J. Comput.* 41, 6 (2012), 1704–1721.

[4] Joshua D. Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. 2013. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM* 56, 8 (2013), 87–94.

[5] András Benczúr and David Karger. 1996. Approximating s-t Minimum Cuts in $\widetilde{O}(n^2)$ Time. In *28th Annual ACM Symposium on Theory of Computing (STOC'96)*. 47–55.

[6] L. Paul Chew. 1989. There Are Planar Graphs Almost as Good as the Complete Graph. *J. Comput. System Sci.* 39, 2 (1989), 205–219.

[7] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. 2015. Uniform sampling for matrix approximation. In *6th Innovations in Theoretical Computer Science (ITCS'15)*. 181–190.

[8] Rahul Jain and Penghui Yao. 2011. A parallel approximation algorithm for positive semidefinite programming. In *52th Annual IEEE Symposium on Foundations of Computer Science (FOCS'11)*. 463–471.

[9] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. 2016. Sparsified Cholesky and multigrid solvers for connection Laplacians. In *48th Annual ACM Symposium on Theory of Computing (STOC'16)*. 842–850.

[10] Yin Tat Lee and He Sun. 2015. Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. 250–269.

[11] Mu Li, Gary L Miller, and Richard Peng. 2013. Iterative row sampling. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS'13)*. 127–136.

[12] Jelani Nelson and Huy L Nguyên. 2013. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS'13)*. 117–126.

[13] Richard Peng and Daniel A. Spielman. 2014. An efficient parallel solver for SDD linear systems. In *46th Annual ACM Symposium on Theory of Computing (STOC'14)*. 333–342.

[14] Richard Peng and Kanat Tangwongsan. 2012. Faster and simpler width-independent parallel algorithms for positive semidefinite programming. In *24th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'12)*. 101–108.

[15] Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926.

[16] Daniel A. Spielman and Shang-Hua Teng. 2011. Spectral sparsification of graphs. *SIAM J. Comput.* 40, 4 (2011), 981–1025.