
Proceedings Paper:
Manghiuc, B.-A., Peng, P. orcid.org/0000-0003-2700-5699 and Sun, H. (2020) Augmenting the algebraic connectivity of graphs. In: Grandoni, F., Herman, G. and Sanders, P., (eds.) 28th Annual European Symposium on Algorithms (ESA 2020). 28th European Symposium on Algorithms (ESA 2020), 07-10 Sep 2020, Online conference. Leibniz International Proceedings in Informatics (LIPIcs) . Schloss Dagstuhl--Leibniz-Zentrum fur Informatik , 70:1-70:22. ISBN 9783959771627

https://doi.org/10.4230/LIPIcs.ESA.2020.70

Augmenting the Algebraic Connectivity of Graphs

Bogdan-Adrian Manghiuc
School of Informatics, University of Edinburgh, UK
b.a.manghiuc@sms.ed.ac.uk

Pan Peng
Department of Computer Science, University of Sheffield, UK
p.peng@sheffield.ac.uk

He Sun
School of Informatics, University of Edinburgh, UK
h.sun@ed.ac.uk

Abstract

For any undirected graph $G = (V, E)$ and a set E_W of candidate edges with $E \cap E_W = \emptyset$, the (k, γ)-spectral augmentability problem is to find a set F of k edges from E_W with appropriate weighting, such that the algebraic connectivity of the resulting graph $H = (V, E \cup F)$ is least γ. Because of a tight connection between the algebraic connectivity and many other graph parameters, including the graph's conductance and the mixing time of random walks in a graph, maximising the resulting graph's algebraic connectivity by adding a small number of edges has been studied over the past 15 years, and has many practical applications in network optimisation.

In this work we present an approximate and efficient algorithm for the (k, γ)-spectral augmentability problem, and our algorithm runs in almost-linear time under a wide regime of parameters. Our main algorithm is based on the following two novel techniques developed in the paper, which might have applications beyond the (k, γ)-spectral augmentability problem:

- We present a fast algorithm for solving a feasibility version of an SDP for the algebraic connectivity maximisation problem from [16]. Our algorithm is based on the classic primal-dual framework for solving SDP, which in turn uses the multiplicative weight update algorithm. We present a novel approach of unifying SDP constraints of different matrix and vector variables and give a good separation oracle accordingly.
- We present an efficient algorithm for the subgraph sparsification problem, and for a wide range of parameters our algorithm runs in almost-linear time, in contrast to the previously best known algorithm running in at least $\Omega(n^2mk)$ time [22]. Our analysis shows how the randomised BSS framework can be generalised in the setting of subgraph sparsification, and how the potential functions can be applied to approximately keep track of different subspaces.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Graph sparsification, Algebraic connectivity, Semidefinite programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.70

Related Version The full version of the paper is available at `https://arxiv.org/abs/2006.14449`.

Funding *He Sun*: Support by an EPSRC Early Career Fellowship (EP/T00729X/1).

1 Introduction

Graph expansion is the metric quantifying how well vertices are connected in a graph, and has applications in many important problems of computer science: in complexity theory, graphs with good expansion are used to construct error-correcting codes [38, 42] and pseudorandom generators [18]; in network design, expander graphs have been applied in constructing super concentrators [40]; in probability theory, graph expansion is closely related to the behaviours of random walks in a graph [29, 37]. On the other side, as most graphs occurring in practice might not be expander graphs and a subset of vertices of low expansion is usually viewed as the

bottleneck of a graph, finding the set of vertices with minimum expansion has many practical applications including image segmentation [27], community detection [31, 36], ranking web pages, among many others. Because of these, both the approximation algorithms for the graph expansion problem and the computational complexity of the problem itself have been extensively studied over the past three decades.

In this paper we study the following graph expansion optimisation problem: given an undirected and weighted graph $G = (V, E, w)$, a set E_W of candidate edges, and a parameter $k \in \mathbb{N}$ as input, we are interested in (i) finding a set $F \subseteq E_W$ of k edges and their weights such that the resulting graph $H = (V, E \cup F, w')$ with weight function[1] $w' : E \cup F \to \mathbb{R}_{\geq 0}$ has good expansion, or (ii) showing that it's impossible to significantly improve the graph's expansion by adding k edges from E_W. Despite sharing many similarities with the sparest cut problem, our proposed problem has many of its own applications: for example, assume that the underlying graph G is a practical traffic or communication network and, due to physical constraints, only certain links can be used to improve the network's connectivity. For any given k and a set of feasible links, finding the best k links to optimise the network's connectivity is exactly the objective of our graph expansion optimisation problem.

To formalise the problem, we follow the work of [15, 16, 22] and define the *algebraic connectivity* of G by the second smallest eigenvalue $\lambda_2(L_G)$ of the Laplacian matrix L_G of G defined by $L_G \triangleq D_G - A_G$, where D_G is the diagonal matrix consisting of the degrees of the vertices and A_G is the adjacency matrix of G. Given an undirected and weighted graph $G = (V, E, w)$ with n vertices, $O(n)$ edges[2], a set E_W of candidate edges defined on V satisfying $E_W \cap E = \emptyset$ and a parameter k, we say that G is (k, γ)-*spectrally-augmentable with respect to* $W = (V, E_W)$, if there is $F \subseteq E_W$ with $|F| = k$ together with edge weights $\{w_e\}_{e \in F}$ such that $H = (V, E \cup F, w)$ satisfies $\lambda_2(L_H) \geq \gamma$. The main result of our work is an almost-linear time[3] algorithm that either (i) finds a set of $O(k)$ edges from E_W if G is (k, γ)-spectrally augmentable for some $\gamma \geq \Delta \cdot n^{-1/q}$, or (ii) returns "no" if G is not $(O(kq), O(\Delta \cdot n^{-2/q}))$-spectrally augmentable, where Δ is an upper bound of both the maximum degree of G and W, and $q \geq 10$ is an arbitrary integer. The formal description of our result is as follows:

▶ **Theorem 1.** *Let $G = (V, E, w)$ be a base graph with n vertices, $O(n)$ edges, and weight function $w : E \to \mathbb{R}_{\geq 0}$, and let $W = (V, E_W)$ be the candidate graph of m edges such that the maximum degrees of G and W is at most Δ. Then, there is an algorithm such that for any integer $k \geq 1$ and $q \geq 10$, the following statements hold:*

- *if G is $\left(k, \Delta \cdot n^{-1/q}\right)$-spectrally-augmentable with respect to W, then the algorithm finds a set $F \subseteq E_W$ of edges and a set of edge weights $\{w(e) : e \in F\}$ such that $|F| = O(qk)$, $\sum_{e \in F} w(e) \leq O(k)$, and the resulting graph $H = (V, E \cup F)$ satisfies that $\lambda_2(L_H) \geq c\lambda_\star^2 \Delta$, for some constant $c > 0$, where $\lambda_\star \cdot \Delta$ is the optimum solution[4].*

---

[1] We remark that the weight function w' needs to satisfy that $w'(e) = w(e)$ for any edge $e \in E$.

[2] Since a spectral sparsifier of G with $O(n)$ edges preserves the eigenvalues of the Laplacian matrix of G, we assume that G has $O(n)$ edges throughout the paper. Otherwise one can always run the algorithm in [25] to get a linear-sized spectral sparsifier of G and use this sparsifier as the input of our problem. Therefore, the number of edges in G will not be mentioned in our paper to simplify the notation.

[3] We say that a graph algorithm runs in almost-linear time if the algorithm's runtime is $O((m+n)^{1+c})$ for an arbitrary small constant c, where n and m are the number of vertices and edges of G, respectively. Similarly, we say that a graph algorithm runs in nearly-linear time if the algorithm's runtine is $O((m+n) \cdot \log^c(n))$ for some constant c.

[4] Note that since G is $(k, n^{-1/q} \cdot \Delta)$-spectrally-augmentable with respect to W, it always holds that $\lambda_\star \geq n^{-1/q}$.

- *if G is not $\left(O(kq), O(\Delta \cdot n^{-2/q})\right)$-spectrally-augmentable with respect to W, then the algorithm rejects the input G, W.*

Moreover, the algorithm runs in $\widetilde{O}\left(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\}\right)$ time. Here, the $\widetilde{O}(.)$ notation hides $\operatorname{poly}\log n$ factors, and ω is the constant for matrix multiplication.

We remark that the most typical application of our problem is the scenario in which only a low number of edges are needed such that the resulting graph enjoys good expansion, and these correspond to the regime of $k = n^{o(1)}$ and $\lambda_\star \in (n^{-1/q}, O(1))$ [34], under which our algorithm runs in almost-linear time and achieves an $\Omega(\lambda_\star)$-approximation. In particular, when it is possible to augment G to be an expander graph, i.e. $\lambda_\star = \Theta(1)$, our algorithm achieves a constant-factor approximation. Our algorithm runs much faster than the previously best-known algorithm for a similar problem that runs in *at least* $\Omega\left(n^2mk\right)$ time [22], though their algorithm solves the more general problem: for any instance G, W, k, if the optimum solution is $\lambda_\star\Delta$, i.e., G is $(k, \lambda_\star\Delta)$-spectrally-augmentable with respect to W, for *any* $\lambda_\star \in [0, 1)$, then their algorithm finds a graph $H = (V, E \cup F)$ with $\lambda_2(L_H) \geq c\lambda_\star^2\Delta$ such that $|F| = O(k)$ and the total sum of weights of edges in F is at most k. Our algorithm can only find a graph H when $\lambda_\star \in (n^{-1/q}, 1)$.

To give an overview of the proof technique for Theorem 1, notice that our problem is closely linked to the *algebraic connectivity maximisation problem* studied in [16], which looks for k edges from the candidate set to maximise $\lambda_2(L_H)$ of the resulting graph H. It is known that the algebraic connectivity maximisation problem is **NP**-hard [30], and Ghosh and Boyd [16] show that this problem can be formulated as an SDP, which we call the GB-SDP. Inspired by this, we study the following P-SDP, which is the feasibility version of the GB-SDP parameterised by some γ. Here, $P_\perp$ is the projection on the space orthogonal to $\mathbf{1} \triangleq (1, \ldots, 1)^\intercal$, i.e., $P_\perp = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\intercal$.

$$\text{P-SDP}(G, W, k, \gamma) \qquad \begin{aligned} \lambda &\geq \gamma \\ L_G + \sum_{e \in E_W} w_e L_e &\succeq \lambda\Delta P_\perp \\ k - \sum_{e \in E_W} w_e &\geq 0 \\ 1 - w_e &\geq 0, \qquad \forall e \in E_W \\ w_e &\geq 0, \qquad \forall e \in E_W \\ \gamma &\geq 0. \end{aligned}$$

Notice that, if G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W, then there is a set F of k edges such that, by setting $w_e = 1$ if $e \in F$ and $w_e = 0$ otherwise, it holds that $L_G + \sum_{e \in E_W} L_e \succeq \gamma\Delta P_\perp$. Therefore, there is a feasible solution of P-SDP(G, W, k, γ). Our algorithmic result for solving the P-SDP is summarised as follows:

▶ **Theorem 2.** *Let $\delta' > 0$ be any constant. There exists an algorithm running in $\widetilde{O}((m+n)/\gamma^2)$ time that either finds a solution to P-SDP$(G, W, k, (1-\delta')\gamma)$ or certifies that there is no feasible solutions for P-SDP(G, W, k, γ).*

Since the solution to the P-SDP only guarantees that the total weights of the selected edges are at most k if G is $(k, \gamma\Delta)$-spectrally augmentable, following [22] we use a subgraph sparsification algorithm to round our SDP solution, such that there are only $O(k)$ edges selected in the end. To give a high-level overview of this rounding step, we redefine the set E_W of candidate edges, and assume that E_W consists of the edges whose weight from the P-SDP solution is non-zero. Therefore, our objective is to find $O(k)$ edges from E_W and new weights, which form an edge set F, such that the Laplacian matrix L_H of the

resulting graph $H = (V, E \cup F)$ is close to L_{G+W}. That is, the subgraph sparsification problem asks for a sparse representation of $G + W$ while keeping the entire base graph G in the resulting representation. Our improved algorithm shows that, as long as $k = n^{o(1)}$, a subgraph sparsifier can be computed in almost-linear time[5]. Our result on subgraph sparsification will be formally described in Theorem 9.

1.1 Our techniques

In this section we will explain the techniques used to design the fast algorithm for the P-SDP, and an almost-linear time algorithm for subgraph sparsification.

Faster algorithm for solving the P-SDP. Our efficient SDP solver is based on the primal-dual framework developed in [5], which has been used in many other works [19, 33]. In this primal-dual framework, we will work on both the original SDP P-SDP(G, W, k, γ) and its dual D-SDP(G, W, k, γ) that is defined as follows:

$$\begin{aligned} & Z \bullet L_G + kv + \sum_{e \in E_W} \beta_e < \gamma \\ & Z \bullet \Delta P_\perp = 1 \\ \text{D-SDP}(G, W, k, \gamma) \qquad & Z \bullet L_e \leq v + \beta_e, \qquad \forall e \in E_W \\ & Z \succeq 0 \\ & \beta_e \geq 0, \qquad \forall e \in E_W \\ & v \geq 0. \end{aligned}$$

We then apply the matrix multiplicative weight update (MWU) algorithm. Formally speaking, starting with some initial embedding $X^{(1)}$, for each $t \geq 1$ our algorithm iteratively uses a carefully constructed oracle Oracle for D-SDP(G, W, k, γ) to check whether the current embedding $X^{(t)}$ is good or not:

- If $X^{(t)}$ satisfies some condition, denoted by $\mathcal{C}\left(X^{(t)}\right)$, then the oracle fails and this implies that we can find a feasible solution from $X^{(t)}$ to D-SDP(G, W, k, γ). This implies that the primal SDP P-SDP(G, W, k, γ) has no feasible solution, which certifies that G is not (k, γ)-spectrally-augmentable with respect to W.
- If $X^{(t)}$ does not satisfy the condition $\mathcal{C}\left(X^{(t)}\right)$, then the oracle does not fail, which certifies that the current solution from $X^{(t)}$ is not feasible for D-SDP(G, W, k, γ), and will output a set of numbers $\left(\lambda^{(t)}, w^{(t)}\right)$ for updating the embedding.

The procedure above will be iterated for T times, for some T depending on the oracle and the approximate parameter $\delta' > 0$: if the oracle fails in any iteration, then P-SDP is infeasible; otherwise, the oracle does not fail for all T iterations and we find a feasible solution to P-SDP$(G, W, k, (1 - \delta')\gamma)$.

The main challenge for applying the above framework in our setting is to construct the Oracle and deal with the complicated constraints in our SDPs, which include both matrix inequality constraint and vector inequality constraints of different variables. To work with these constraints, our strategy is to unify them through a diagonal block matrix X, and through this we turn all individual constraints into a single matrix constraint. The embedding in each iteration is constructed in nearly-linear time in $n + m$ by the definition

[5] We remark that, when $k = \Theta(n)$, our problem can be solved directly by using a spectral sparsifier $\widetilde{W}$ of the graph W with $O(n)$ edges, which can be computed in nearly-linear time. This will imply that the two Laplacians $L_{G+\widetilde{W}} = L_G + L_{\widetilde{W}}$ and $L_{G+W} = L_G + L_W$ are close.

of the embedding. To construct the ORACLE, we carefully design the condition $\mathcal{C}(X)$ with the intuition that if the candidate solution corresponding to X has a relatively small dual objective value, then a re-scaling of X gives a feasible solution to D-SDP. Then we use a case analysis to show that if $\mathcal{C}(X)$ is not satisfied, we can very efficiently find updating numbers $(\lambda^{(t)}, w^{(t)})$ by distinguishing edges satisfying one constraint (in D-SDP) from those that do not satisfy it and assigning different weights $w^{(t)}$ to them accordingly.

Faster algorithm for subgraph sparsification. The second component behind proving our main result is an efficient algorithm for the subgraph sparsification problem. Our algorithm is inspired by the the original deterministic algorithm for subgraph sparsification [22] and the almost-linear time algorithm for constructing linear-sized spectral sparsifiers [26]. In particular, both algorithms follow the BSS framework, and proceed in iterations: it is shown that, with the careful choice of barrier values u_j and ℓ_j in each iteration j and the associated potential functions, one or more vectors can be selected in each iteration and the final barrier values can be used to bound the approximation ratio of the constructed sparsifier.

However, in contrast to most algorithms for linear-sized spectral sparsifiers [4, 25, 26], both the barrier values and the potential functions in [22] are employed for a slightly different purpose. In particular, instead of expecting the final constructed ellipsoid to be close to a sphere, the final constructed ellipsoid for subgraph sparsification could be still very far from being a sphere, since the total number of added edges is $O(k)$. Because of this, the two potential functions in [22] are used to quantify the contribution of the added vectors towards *two different subspaces*: one fixed k-dimensional subspace denoted by S, and one variable space defined with respect to the currently constructed matrix. Based on analysing two different subspaces for every added vector, which is computationally expensive, the algorithm in [22] ensures that the added vectors will significantly benefit the "worst subspace", the subspace in $\mathbb{R}^n$ that limits the approximation ratio of the final constructed sparsifier.

Because of these different roles of the potential functions in [22] and [6, 26], when applying the randomised BSS framework [26] for the subgraph sparsification problem, more technical issues need to take into account: (1) Since [22] crucially depends on some projection matrix denoted by P_S, of which the exact computation is expensive, to obtain an efficient algorithm for subgraph sparsification one needs to obtain some projection matrix close to P_S and such a projection matrix can be computed efficiently. (2) Since the upper and lower potential functions keep track of different subspaces whose dimensions are of different orders in most regimes, analysing the impact of multiple added vectors to the potential functions are significantly more challenging than [26].

To address the first issue, we show that the problem of computing an approximate projection close to P_S while preserving relevant proprieties can be reduced to the generalised eigenvalue problem, which in turn can be efficiently approximated by a recent algorithm [2]. For the second issue, we meticulously bound the *intrinsic dimension* of the matrix corresponding to the multiple added vectors, and by a more refined matrix analysis than [26] we show that the potential functions and the relative effective resistances decease in each iteration. We highlight that developing a fast procedure to computing all the quantities that involve a fixed projection matrix and analysing the impact of multiple added vectors with respect to two different subspaces constitute the most challenging part of the design of our algorithm.

Finally, we remark that, although the almost-linear time algorithm [26] has been improved by the subsequent paper [25], it looks more challenging to adapt the technique developed in [25] for the setting of subgraph sparsification. In particular, since the two potential functions

in [25] are used to analyse the same space $\mathbb{R}^n$, it is shown in [25] that it suffices to analyse the one-sided case through a one-sided oracle. However, the two potential functions defined in our paper are used to analyse two different subspaces, and it remains unclear whether we can reduce our problem to the one-sided case. We will leave this for future work.

1.2 Other related work

Spielman and Teng [39] present the first algorithm for constructing spectral sparsifiers: for any parameter $\varepsilon \in (0,1)$, and any undirected graph G of n vertices and m edges, they prove that a spectral sparsifier of G with $\widetilde{O}\left(n/\varepsilon^2\right)$ edges exists, and can be constructed in $\widetilde{O}\left(m/\varepsilon^2\right)$ time. Since then, there has been extensive studies on different variants of spectral sparsifiers and their efficient constructions in various settings. In addition to several results on several constructions of linear-sized spectral sparsifiers mentioned above, there are many studies on constructing spectral sparsifiers in streaming and dynamic settings [1, 20, 21]. The subgraph sparsification problem has many applications, including constructing precondtioners and nearly-optimal ultrasparsifiers [22, 35], optimal approximate matrix product [11], and some network optimisation problems [28]. Our work is also related to a sequence of research on network design, in which the goal is to find minimum cost subgraphs under some "connectivity constraints". Typical examples include constraints on vertex connectivity [8, 9, 10, 14, 23, 24], shortest path distances [12, 13], and spectral information [3, 7, 17, 32].

2 A fast SDP solver

We use the primal-dual framework introduced in [5] to solve the SDP $\mathsf{P\text{-}SDP}(G, W, k, \gamma)$ and prove Theorem 2. The framework is based on the matrix multiplicative weight update (MWU) algorithm on both the primal SDP $\mathsf{P\text{-}SDP}(G, W, k, \gamma)$ and its dual $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

Notation. For any given vector β, we use $\mathbf{Diag}(\beta)$ to denote the diagonal matrix such that each diagonal entry $[\mathbf{Diag}(\beta)]_{ii} = \beta_i$. Given matrix Z, scalar v and vector β, we use $\mathbf{Diag}(Z, v, \beta)$ to denote the diagonal 3-block matrix with blocks Z, v and $\mathbf{Diag}(\beta)$. We use I_V and I_{E_W} to denote the identity matrices on vertex set V and edge set E_W with $|E_W| = m$, respectively. We further define

$$\begin{aligned} &E \triangleq \mathbf{Diag}(\Delta \cdot I_V, m, I_{E_W}), \quad \Pi \triangleq \mathbf{Diag}(P_\perp, 1, I_{E_W}), \\ &N \triangleq \mathbf{Diag}(\Delta \cdot P_\perp, m, I_{E_W}) = E^{1/2}\Pi E^{1/2}. \end{aligned} \tag{1}$$

For any given parameter λ and vector w, we define $V(\lambda, w) \triangleq \lambda$, $A(\lambda, w) \triangleq L_G + \sum_{e\in E_W} w_e L_e - \lambda\Delta P_\perp$, and $B(\lambda, w) \triangleq k - \sum_{e\in E_W} w_e$. Let $c = c(\lambda, w) \in \mathbb{R}^m$ denote the vector with $c_e = 1 - w_e$ for each $e \in E_W$, and $C = C(\lambda, w) = \mathbf{Diag}(c(\lambda, w))$ be the diagonal $m \times m$ matrix with the diagonal entry $1 - w_e$ corresponding to edge e. Then we define

$$M(\lambda, w) \triangleq \mathbf{Diag}\left(A(\lambda, w), B(\lambda, w), C(\lambda, w)\right) = \begin{bmatrix} A(\lambda, w) & 0 & 0 \\ 0 & B(\lambda, w) & 0 \\ 0 & 0 & C(\lambda, w) \end{bmatrix}. \tag{2}$$

▶ **Definition 3.** *An (ℓ, ρ)-oracle for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$ is an algorithm that on input $\langle Z, v, \beta\rangle$ with $\mathbf{Diag}(Z, v, \beta) \bullet N = 1$, either fails or outputs (λ, w) with $\lambda \geq 0$, $w \in \mathbb{R}^m_{\geq 0}$ that satisfies*

$$V(\lambda, w) \geq \gamma, \qquad A(\lambda, w) \bullet Z + B(\lambda, w) \cdot v + c(\lambda, w) \cdot \beta \geq 0, \qquad -\ell N \preceq M(\lambda, w) \preceq \rho N.$$

▶ **Fact 4.** *If an (ℓ, ρ)-oracle for D-SDP(G, W, k, γ) does not fail on input $\langle Z, v, \beta\rangle$ with $\boldsymbol{Diag}(Z, v, \beta) \bullet N = 1$, then $\langle Z, v, \beta\rangle$ is infeasible for D-SDP(G, W, k, γ).*

In order to apply the MWU algorithm, in the following we use $U_\varepsilon(A)$ to denote the matrix

$$U_\varepsilon(A) \triangleq \frac{E^{-1/2}(1-\varepsilon)^{E^{-1/2}AE^{-1/2}}E^{-1/2}}{\Pi \bullet (1-\varepsilon)^{E^{-1/2}AE^{-1/2}}},$$

where E and Π are matrices defined in (1).

2.1 The MWU algorithm

In the framework of MWU for solving our SDP, we sequentially produce candidate dual solutions $\langle Z^{(t)}, v^{(t)}, \beta^{(t)}\rangle$ such that $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)}) \bullet N = 1$ for any t. Specifically, for any given k, γ, we start with a solution $Z^{(1)} = \frac{1}{\Delta(n-1)} I$, $v^{(1)} = \frac{2}{n-1}$ and $\beta_e^{(1)} = 0$ for any $e \in E_W$. At each iteration t, we invoke a good separation oracle that takes $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$ as input, and either guarantees that $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$ is already good for dual SDP (and thus certifies infeasibility of primal SDP), or outputs $(\lambda^{(t)}, w^{(t)})$ certifying the infeasibility of $\mathbf{Diag}(Z^{(t)}, v^{(t)}, \beta^{(t)})$.

If $\left(\lambda^{(t)}, w^{(t)}\right)$ is returned by the oracle, then the algorithm updates the next candidate solution based on $X^{(t)} = U_\varepsilon\left(\frac{1}{2\rho}\sum_{s=1}^{t-1} M^{(s)}\right)$, where $M^{(s)} \triangleq M\left(\lambda^{(s)}, w^{(s)}\right)$ is as defined before and ε is a parameter of the algorithm. By definition, we have that $X^{(t)} \bullet N = 1$. Moreover, since $M^{(t)}$ can be viewed as a 3-block diagonal matrix with diagonal entries $A^{(t)}, B^{(t)}, C^{(t)}$, we have that $\exp(M^{(t)}) = \mathbf{Diag}\left(\exp\left(A^{(t)}\right), \exp\left(B^{(t)}\right), \exp\left(C^{(t)}\right)\right)$. Therefore, we can decompose $X^{(t)}$ as

$$X^{(t)} = \mathbf{Diag}\left(Z^{(t)}, v^{(t)}, \beta^{(t)}\right).$$

Note that $X^{(t)} \bullet N = 1$ is equivalent to

$$\Delta \cdot Z^{(t)} \bullet P_\perp + m \cdot v^{(t)} + \sum_{e \in E_W} \beta_e^{(t)} = 1.$$

The following theorem guarantees that, after a small number of iterations, the algorithm either finds a good enough dual solution, or a feasible solution to the primal SDP.

▶ **Theorem 5.** *Let Oracle be an (ℓ, ρ)-oracle for D-SDP(G, W, k, γ), and let $\delta > 0$. Let N, $X^{(t)}$, and $M^{(t)}$ be defined as above, for any $t \geq 1$. Let $\varepsilon = \min\{1/2, \delta/2\ell\}$. Suppose that Oracle does not fail for T rounds, where*

$$T = O\left(\frac{\rho \log n}{\delta\varepsilon}\right) \leq \max\left\{O\left(\frac{\rho \log n}{\delta}\right), O\left(\frac{\rho\ell \log n}{\delta^2}\right)\right\},$$

then $(\bar{\lambda} - 3\delta, \bar{w} - \delta)$ is a feasible solution to P-SDP$(G, W, k, \gamma - 3\delta)$, where $\bar{\lambda} \triangleq \frac{1}{T}\sum_{t=1}^{T} \lambda^{(t)}$ and $\bar{w} \triangleq \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$.

Approximate computation. By applying the Johnson-Linderstrauss (JL) dimensionality reduction to the embedding corresponding to U_ε, we can approximate $X^{(t+1)}$ by $\widetilde{X}^{(t+1)}$ while preserving the relevant properties. Specifically, let $\widetilde{U}_\varepsilon$ be a randomised approximation to U_ε from applying the JL Lemma, and we compute in nearly-linear time the matrix $\widetilde{X}^{(t+1)} = \widetilde{U}_\varepsilon\left(\frac{1}{2\rho}\sum_{s=1}^{t-1} M^{(s)}\right)$ and decompose it into 3 blocks:

$$\widetilde{X}^{(t+1)} = \mathbf{Diag}\left(\widetilde{Z}^{(t+1)}, \widetilde{v}^{(t+1)}, \mathbf{Diag}\left(\widetilde{\beta}^{(t+1)}\right)\right).$$

Moreover, $\widetilde{X}^{(t+1)} \bullet L_H$ well approximates $X^{(t+1)} \bullet L_H$ for any graph H, which suffices for our oracle. Hence, we assume that the oracle receives $\widetilde{X}^{(t+1)}$ as input instead of $X^{(t+1)}$. We defer the formal lemma we are using to the full version.

2.2 The oracle

In this subsection, we will present and analyse the oracle for our SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$, which is presented in Algorithm 1. For the simplicity of presentation, we abuse notation and use $X = \mathbf{Diag}(Z, v, \beta)$ to denote the input to the oracle, although it should be clear that the input is the approximate embedding $\widetilde{X} = \mathbf{Diag}(\widetilde{Z}, \widetilde{v}, \mathbf{Diag}(\widetilde{\beta}))$ of X.

Algorithm 1 ORACLE for SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

Require: Candidate solution $\langle Z, v, \beta \rangle$ with $\Delta \cdot Z \bullet P_\perp + m \cdot v + \sum_{e \in E_W} \beta_e = 1$, target value γ

Let $B := \{e : v + \beta_e < L_e \bullet Z\}$, $\Gamma := \sum_{e \in B}(L_e \bullet Z - v - \beta_e)$, and $T := Z \bullet \Delta P_\perp$.
Let $T_{\mathrm{tol}} := L_G \bullet Z + kv + \sum_{e \in E_W} \beta_e$.
if $\Gamma \leq T\gamma - T_{\mathrm{tol}}$ **then**
    Output "fail". ▷ In this case, $\langle Z, v, \beta \rangle$ is "good" enough
else if $T_{\mathrm{tol}} > \gamma m - \gamma \sum_{e \in E_W} Z \bullet L_e$ **then**
    return $w_e = \gamma$, and $\lambda = \gamma$.
else
    return $w_e = 1$ for $e \in B$, $w_e = 0$ for $e \in E_W \setminus B$ and $\lambda = \gamma$

To analyse the oracle, we prove the following technical lemma. First of all, we show that if the ORACLE fails, then we can find a dual feasible solution for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.

▶ **Lemma 6.** *Let $\langle Z, v, \beta \rangle$ be a candidate solution. Suppose that for $B \triangleq \{e : v + \beta_e - L_e \bullet Z < 0\}$, $T \triangleq Z \bullet \Delta P_\perp$ and $T_{tol} \triangleq L_G \bullet Z + kv + \sum_{e \in E_W} \beta_e$, then it holds that $\Gamma \triangleq \sum_{e \in B}(L_e \bullet Z - v - \beta_e) \leq T\gamma - T_{tol}$. Moreover, by setting $Z' = \frac{Z}{T}$, $v' = \frac{v}{T}$, and $\beta'_e = \frac{\beta_e}{T}$ if $e \in E_W \setminus B$ and $\beta'_e = \frac{L_e \bullet Z - v}{T}$ if $e \in B$, we have that $\langle Z', v', \beta' \rangle$ is a dual feasible for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$.*

Secondly, we show that, if ORACLE does not fail, then it returns (λ, w) that satisfies the properties of (ℓ, ρ)-oracle for $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$ for appropriate ℓ, ρ.

▶ **Lemma 7.** *When ORACLE described in the algorithm does not fail, it returns a vector w and value λ such that $V(\lambda, w) \geq \gamma$, and for the matrix $M(\lambda, w) = \mathbf{Diag}(A(\lambda, w), B(\lambda, w), C(\lambda, w))$, $A(\lambda, w) \bullet Z + B(\lambda, w) \cdot v + C(\lambda, w) \cdot \beta \geq 0$. Moreover, it holds that $-N \preceq M(\lambda, w) \preceq 3N$.*

Combining these two lemmas together, we obtain the following theorem which summarise the performance of Algorithm 1.

▶ **Theorem 8.** *On input $\widetilde{X}^{(t)}$, there exists an algorithm ORACLE that runs in $\widetilde{O}(n + m)$ time and is an (ℓ, ρ)-oracle for SDP $\mathsf{D\text{-}SDP}(G, W, k, \gamma)$, where $\ell = 1$ and $\rho = 3$.*

2.3 Proof of Theorem 2

Now we are ready to prove Theorem 2.

Proof of Theorem 2. Let $\delta' > 0$ be any constant. We specify $\delta = \frac{\delta'\gamma}{3}$ in our MWU algorithm, which is described in the previous subsections. We set $\rho = 3$ and $\ell = 1$, and let

$$T \triangleq O\left(\frac{\rho \ell \log n}{\delta^2}\right) = O\left(\frac{\log n}{(\delta')^2 \gamma^2}\right) = O\left(\frac{\log n}{\gamma^2}\right).$$

In the MWU algorithm, if the ORACLE fails in the t-th iteration for some $1 \le t \le T$, then the corresponding embedding $\widetilde{X}^{(t)} = \mathbf{Diag}(\widetilde{Z}^{(t)}, \widetilde{v}^{(t)}, \mathbf{Diag}(\beta^{(t)}))$ provides a good enough solution: the precondition of Lemma 6 is satisfied, which further implies that $\widetilde{X}^{(t)}$ can be turned into a dual feasible solution with objective at most γ, i.e., we find a solution to D-SDP(G, W, k, γ). Therefore, the primal SDP P-SDP(G, W, k, γ) is infeasible. Otherwise, we know that the ORACLE does not fail for T iterations; by Theorem 5 and Lemma 7, it holds that for $\bar{\lambda} \triangleq \frac{1}{T}\sum_{t=1}^{T} \lambda^{(t)}, \bar{w} \triangleq \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$, $(\bar{\lambda} - 3\delta, \bar{w} - \delta)$ is a feasible solution for P-SDP$(G, W, k, \gamma - 3\delta)$ = P-SDP$(G, W, k, \gamma - \delta'\gamma)$.

Now we analyse the algorithm's runtime. By Theorem 8 and the approximate computation of $X^{(t+1)}$, each iteration can be implemented in $\widetilde{O}(n+m)$ time. Thus, in $\widetilde{O}((n+m)/\gamma^2)$ time, we either find a solution to our SDP with objective value at least $(1-\delta')\gamma$ for any constant $\gamma' > 0$, or we certify that the P-SDP(G, W, k, γ) is infeasible (in case the ORACLE fails). ◀

3 Algorithm for subgraph sparsification

Now we give an overview of our efficient algorithm for constructing subgraph sparsifiers. Recall that, for any $k \in \mathbb{N}$, parameter $\kappa \ge 1$, and two weighted graphs $G = (V, E)$ and $W = (V, E_W)$, the subgraph sparsification problem is to find a set $F \subseteq E_W$ of $|F| = O(k)$ edges with weights $\{w_e\}_{e\in F}$, such that the resulting graph $H = (V, E + F)$ is a κ-approximation of $G + W$, i.e.,

$$L_{G+W} \preceq L_G + \sum_{e\in F} w_e b_e b_e^{\intercal} \preceq \kappa \cdot L_{G+W}. \tag{3}$$

To construct the required edge set F, we apply the standard reduction for constructing graph sparsifiers by setting $v_e \triangleq L_{G+W}^{\dagger/2} b_e$ for every $e \in E_W$, and (3) is equivalent to

$$I_{\mathsf{im}(L_{G+W})} \preceq L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2} + \sum_{e\in F} w_e v_e v_e^{\intercal} \preceq \kappa \cdot I_{\mathsf{im}(L_{G+W})},$$

where $I_{\mathsf{im}(L_{G+W})}$ is the identity on $\mathsf{im}(L_{G+W})$. Our main result is summarised as follows:

▶ **Theorem 9.** *Let ε and q be arbitrary constants such that $\varepsilon \le 1/20$ and $q \ge 10$. Then, there is a randomised algorithm such that, for any two graphs $G = (V, E)$ and $W = (V, E_W)$ defined on the same vertex set as input, by defining $X = \left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)\Big|_{\mathrm{Im}(L_{G+W})}$ and $\overline{M} \triangleq \left(\sum_{i=1}^{m} v_i v_i^{\intercal}\right)|_{\mathrm{Im}(L_{G+W})}$ where every v_i is of the form $L_{G+W}^{\dagger/2} b_e$ for some edge $e \in E_W$, the algorithm outputs a set of non-negative coefficients $\{c_i\}_{i=1}^{m}$ with $|\{c_i \mid c_i \ne 0\}| = K$ for some $K = O\left(qk/\varepsilon^2\right)$ such that it holds for some constant C that*

$$C \cdot (1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) \cdot I \preceq X + \sum_{i=1}^{m} c_i v_i v_i^{\intercal} \preceq (1 + O(\varepsilon)) \cdot I, \tag{4}$$

where $T \triangleq \left\lceil \mathrm{tr}\left(\overline{M}\right)\right\rceil$. Moreover, if we assume that every v_i is associated with some cost denoted by $cost_i$ such that $\sum_{i=1}^{m} cost_i = 1$, then with constant probability the coefficients $\{c_i\}_{i=1}^{m}$ returned by the algorithm satisfy $\sum_{i=1}^{m} c_i \cdot cost_i \le O(1/\varepsilon^2) \cdot \min\{1, k/T\}$. The algorithm runs in time

$$\widetilde{O}\left(\min\left\{n^{\omega}, \frac{mk + nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)} \left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^{\omega}, mk + nk^2 + k^{\omega}\right\}\right) \Big/ \varepsilon^5\right).$$

Without loss of generality, we assume that $\overline{M}$ is a full-rank matrix, which can be achieved by adding n self-loops each of small weight $\gamma = \Theta(1/\mathrm{poly}(n))$, so that with constant probability these self-loops will not be sampled by the algorithm.

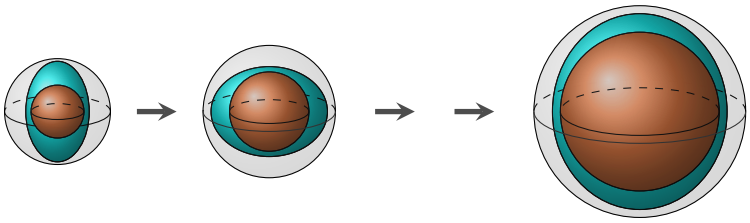

Figure 1 Illustration of the BSS framework: the light grey and orange balls in iteration j represent the spheres $u_j \cdot I$ and $\ell_j \cdot I$, and the cyan ellipsoid sandwiched between the two balls corresponds to the constructed ellipsoid in iteration j. After each iteration j, the algorithm increases the value of ℓ_j and u_j by some $\delta_{\ell,j}$ and $\delta_{u,j}$ so that the invariant (5) holds in iteration $j+1$. This process is repeated for τ iterations, so that the final constructed ellipsoid is close to be a sphere.

3.1 Overview of our algorithm

The BSS framework. At a high level, our algorithm follows the BSS framework for constructing spectral sparsifiers [6]. The BSS algorithm proceeds by iterations: in each iteration j the algorithm chooses one or more vectors, denoted by $v_{j_1}, \ldots, v_{j_k}$, and adds $\Delta_j = \sum_{i=1}^{k} c_{j_i} v_{j_i} v_{j_i}^\intercal$ to the currently constructed matrix by setting $A_j = A_{j-1} + \Delta_j$, where $c_{j_1}, \ldots, c_{j_k}$ are scaling factors, and $A_0 = \mathbf{0}$ initially. Moreover, 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. To ensure this, all the previous analysis uses a potential function $\Phi(A_j, u_j, \ell_j)$ defined by

$$\Phi\left(A_j, u_j, \ell_j\right) = \operatorname{tr}[f(u_j I - A_j)] + \operatorname{tr}[f(A_j - \ell_j I)]$$

for some function f, and a bounded value of $\Phi\left(A_j, u_j, \ell_j\right)$ implies that

$$\ell_j \cdot I \prec A_j \prec u_j \cdot I. \tag{5}$$

After each iteration, the two barrier values ℓ_j and u_j are increased properly by setting $u_{j+1} = u_j + \delta_{u,j}$ and $\ell_{j+1} = \ell_j + \delta_{\ell,j}$ for some positive values $\delta_{u,j}$ and $\delta_{\ell,j}$. The careful choice of $\delta_{u,j}$ and $\delta_{\ell,j}$ ensures that after τ iterations $\mathsf{Ellip}(A_\tau)$ is close to being a sphere, which implies that A_τ is a spectral sparsifier of $\ell_\tau \cdot I$, see Figure 1 for illustration.

The BSS framework for subgraph sparsification. The BSS framework ensures that, when starting with the zero matrix, after choosing $O(n)$ vectors, the final constructed matrix is close to I. However, applying the BSS framework to construct a subgraph sparsifier is significantly more challenging due to the following two reasons:

- Instead of starting with the zero matrix, we need to start with some *non-zero* matrix $A_0 = X$, and the number of added vectors is $K = O(k)$, which could be much smaller than n. This implies that the ellipsoid corresponding to the final constructed matrix could be still very far from being a sphere.
- Because of this and every rank-one update has different contribution towards each direction in $\mathbb{R}^n$, to "optimise" the contribution of $O(k)$ rank-one updates we have to ensure that the added vectors will significantly benefit the "worst subspace", the subspace in $\mathbb{R}^n$ that limits the approximation ratio of the final constructed sparsifier.

To address these two challenges, in the celebrated paper Kolla et al. [22] propose to keep track of the algorithm's progress with respect to two subspaces, each of which is measured by some potential function. Specifically, in each iteration j they define $A_j \triangleq X + \sum_i c_i v_i v_i^\intercal$,

where $\sum_i c_i v_i v_i^\intercal$ is the sum of currently picked rank-one matrices after reweighting during the first j iterations. For the upper barrier value u_j in iteration j, they define the upper potential function

$$\Phi^{u_j}(A_j) \triangleq \operatorname{tr}\left(P_{L(A_j)}\left(u_j I - A_j\right) P_{L(A_j)}\right)^\dagger,$$

where $L(A_j)$ is the T-dimensional subspace of A_j spanned by the T largest eigenvectors of A_j and $P_{L(A_j)}$ is the projection onto that subspace. Notice that $\Phi^{u_j}(A_j)$ is defined with respect to a variable space $L(A_j)$ that *changes after every rank-one update*, in order to upper bound the maximum eigenvalue of the final constructed matrix in the entire space. Similarly, for the same matrix A_j and lower barrier ℓ_j in iteration j, they define the lower potential function by

$$\Phi_{\ell_j}(B_j) \triangleq \operatorname{tr}\left(P_S(B_j - \ell_j I)P_S\right)^\dagger,$$

where P_S is the orthogonal projection onto S, the subspace generated by the bottom k eigenvectors of X, and the matrix B_j is defined by $B_j = Z(A_j - X)Z$, for $Z = \left(P_S(I - X)P_S\right)^{\dagger/2}$. Since the total number of chosen vectors is $K = O(k)$, instead of expecting the final constructed matrix A_τ approximating the identity matrix, the objective of the subgraph sparsification is to find coefficients $\{c_i\}$ with $K = O(k)$ non-zeros such that the following two conditions hold for some positive constants $\theta_{\min}, \theta_{\max}$:

- it holds that $X + \sum_{i=1}^m c_i v_i v_i^\intercal \preceq \theta_{\max} I$, and
- it holds that $\sum_{i=1}^m c_i Z v_i v_i^\intercal Z \succeq \theta_{\min} P_S$.

Informally, the first condition above states that the length of any axis of $\mathsf{Ellip}(A_j)$ is upper bounded, and the second condition ensures that the final matrix A_τ has significant contribution towards the bottom k eigenspace X. In other words, instead of ensuring $\ell_j \cdot I \prec A_j \prec u_j \cdot I$, $\Phi^{u_j}(A_j)$ and $\Phi_{\ell_j}(B_j)$ are used to "quantify" the shapes of the two ellipsoids with different dimensions:

- The function $\Phi^{u_j}(A_j)$ studies the ellipsoid A_j projected onto its own top eigenspaces, the subspace that *changes* after each iteration;
- The function $\Phi_{\ell_j}(B_j)$ studies $A_j - X$ projected onto the bottom k eigenspace of X, the subspace that remains *fixed* during the entire BSS process.

Proving the existence of some vector in each iteration so that the algorithm will make progress is much more involved, and constitutes one of the key lemmas used in [22] for constructing a subgraph sparsifier. However, the subgraph sparsification algorithm in [22] requires the computation of the projection matrices $P_{L(A_j)}$ in each iteration. Because of this, the algorithm presented in [22] runs in $\Omega\left(n^2 mk\right)$ time.

Our approach. At a very high level, our algorithm and its analysis can be viewed as a neat combination of the algorithm presented in [26] and the algorithm presented in [22]. Specifically, for any iteration j with the constructed matrix A_j, we set $B_j \triangleq Z(A_j - X)Z$, where $Z \triangleq \left(P_{\mathcal{V}}(I - X)P_{\mathcal{V}}\right)^{\dagger/2}$, and define the two potential functions by

$$\Phi^{u_j}(A_j) \triangleq \operatorname{tr}\left(P_{L(A_j)}\left(u_j I - A_j\right) P_{L(A_j)}\right)^{\dagger q},$$

and

$$\Phi_{\ell_j}(B_j) \triangleq \operatorname{tr}\left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^{\dagger q}$$

for some fixed projection matrix $P_{\mathcal{V}}$, projecting on a k-dimensional subspace S'. Similar with [26], with the help of the q-th power in the definition of $\Phi^{u_j}(A_j)$ and $\Phi_{\ell_j}(B_j)$ we show that the eigenvalues of our constructed matrices A_j and B_j are never very close to the

two barrier values u_j and ℓ_j. Moreover, although the top T-eigenspace of the currently constructed matrix A_j changes after every rank-one update, multiple vectors can still be selected according to some probability distribution in each iteration.

However, when combining the randomised BSS framework [26] with the algorithm presented in [22], we have to take many challenging technical issues into account. In particular, we need to address the following issues: (1) Both the algorithm and its analysis in [22] crucially depend on the projection matrix P_S, of which the exact computation is expensive. Therefore, in order to obtain an efficient algorithm for subgraph sparsification, one needs to obtain some projection matrix close to P_S and such projection matrix can be computed efficiently. (2) As indicated by our definition of $\Phi_{\ell_j}(B_j)$ above, developing a fast subgraph sparsification algorithm would require efficient approximation of polynomials of the matrix $(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}})^q$. In comparison with [26], the fixed projection matrix $P_{\mathcal{V}}$ sandwiched between two consecutive $(B_j - \ell_j I)$ makes computing the required quantities much more challenging.

To address these issues, we prove that there is a k-dimensional subspace S' close to S, and all of our required quantities that involve the projection onto S', denoted by $P_{\mathcal{V}}$, can be computed efficiently. Moreover, we prove that the quality of our constructed subgraph sparsifer based on the "approximate projection" $P_{\mathcal{V}}$ is the same as the one constructed by [22], in which the "optimal projection" P_S is needed. Our result regarding the approximate subspace S' is summarised as follows:

▶ **Lemma 10.** *There is an algorithm that computes a matrix $\mathbb{V} = L^{-1/2}V$ for matrix V in $t_{10} \triangleq \min\left\{O(n^\omega), \widetilde{O}\left(\frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right)\right\}$ time, such that with constant probability the following two properties hold: (1) $P_{\mathcal{V}} = VV^\intercal$ is a projection matrix on a k-dimensional subspace S' of $\mathbb{R}^n$; (2) For any $u \in \mathbb{R}^n$ satisfying $u^\intercal V = 0$, we have that*

$$\frac{u^\intercal X u}{u^\intercal u} \geq \frac{\lambda_{k+1}(X)}{2}.$$

We highlight that, in comparison to [26], in our setting the upper and lower potential functions *keep track of two different subspaces whose dimensions are of different orders in most regimes*, i.e., k versus T, and this makes our analysis much more involved than [26]. On the other side, we also show that the algorithm in [26] can be viewed as a special case of our algorithm, and from this aspect our algorithm presents a general framework for constructing spectral sparsifiers and subgraph sparsifiers.

3.2 Description of our algorithm

Our algorithm proceeds in iterations in which multiple vectors are sampled with different probabilities. In each iteration j, A_j is updated by setting $A_{j+1} = A_j + \Delta_j$, where Δ_j is the sum of the sampled rank-one matrices with reweighting. To compensate for this change, the two barriers u_j and ℓ_j are increased by $\delta_{u,j}$ and $\delta_{\ell,j}$. The algorithm terminates when the difference of the barriers is greater than α, defined by $\alpha \triangleq 4k/\Lambda$. Specifically, in the initialisation step, the algorithm sets $A_0 \triangleq X$,$u_0 \triangleq 2 + \lambda_{\max}(X)$, $\ell_0 \triangleq -2k/\Lambda$, where $\Lambda \triangleq \max\{k, T\}$. In each iteration j, the algorithm keeps track of the currently constructed matrix A_j and hence, also of the matrix $B_j \triangleq Z(A_j - X)Z$, where $Z \triangleq (P_{\mathcal{V}}(I - X)P_{\mathcal{V}})^{\dagger/2}$ for some fixed projection matrix $P_{\mathcal{V}}$. Intuitively, the projection matrix $P_{\mathcal{V}}$ used here is close to P_S, but can be approximated more efficiently than computing P_S precisely. In each iteration j, the algorithm starts by computing the *relative effective resistances*, which are defined as

$$R_i(A_j, B_j, u_j, \ell_j) \triangleq v_i^\intercal \left(u_j I - A_j\right)^{-1} v_i + v_i^\intercal Z \left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^\dagger Z v_i,$$

for all vectors v_i. Then, the algorithm computes the number of vectors N_j that will be sampled, which can be written as

$$N_j \triangleq \left(\frac{\varepsilon}{4\rho_j} \cdot \lambda_{\min}\left[(u_j I - A_j)^{-1}\overline{M}\right] \cdot \frac{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}{\operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right]}\right)^{2\varepsilon/q} \cdot \rho_j \cdot \min\left\{\frac{1}{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}, \frac{1}{\lambda_{\max}\left(P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right)^\dagger}\right\},$$

where

$$\rho_j \triangleq \sum_{t=1}^{m} R_t(A_j, B_j, u_j, \ell_j) = \operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right] + \operatorname{tr}\left[P_{\mathcal{V}}(B_j - \ell_j I)P_{\mathcal{V}}\right]^\dagger.$$

Next, the algorithm samples N_j vectors such that every v_i is sampled with probability proportional to $R_i(A_j, B_j, u_j, \ell_j)$, i.e., the sampling probability of every v_i is defined by

$$p(v_i) \triangleq \frac{R_i(A_j, B_j, u_j, \ell_j)}{\sum_{t=1}^{m} R_t(A_j, B_j, u_j, \ell_j)}.$$

For every sampled v_i, the algorithm scales it to

$$w_i \triangleq \sqrt{\frac{\varepsilon}{q \cdot R_i(A, B, u, \ell)}} \cdot v_i,$$

and gradually adds $w_i w_i^\intercal$ to A_j. After each rank-one update, the algorithm increases the barrier values by the average increases

$$\overline{\delta}_{u,j} \triangleq \frac{(1+3\varepsilon)\cdot\varepsilon}{q\cdot\rho_j} \qquad \text{and} \qquad \overline{\delta}_{\ell,j} \triangleq \frac{(1-3\varepsilon)\cdot\varepsilon}{q\cdot\rho_j},$$

and checks whether the terminating condition of the algorithm is satisfied. Note that between two consecutive iterations j and $j+1$ of the algorithm, the two barriers u_j and ℓ_j are increased by $\delta_{u,j} \triangleq N_j \cdot \overline{\delta}_{u,j}$ and $\delta_{\ell,j} \triangleq N_j \cdot \overline{\delta}_{\ell,j}$, respectively.

The formal description of our algorithm is presented in Algorithm 2. We remark that, in contrast to the algorithm for constructing a spectral sparsifier [26], the total number of vectors needed in the final iteration j could be much smaller than $O(N_j)$. This is why our algorithm performs a sanity check in Line 15 after every rank-1 update $w_i w_i^\intercal$.

3.3 Proof sketch of Theorem 9

In this subsection we give an overview of the main techniques used for proving Theorem 9. We refer the reader to the full version of our paper for a more detailed discussion.

Approximation Guarantee

Firstly, we will focus on showing that, at the end of Algorithm 2, (4) holds. The result is summarised in the following lemma, whose proof will be left for the end of this subsection.

Algorithm 2 Algorithm for constructing subgraph spectral sparsifiers.

Require: $\varepsilon \leq 1/20, q \geq 10$

$u_0 = 2 + \lambda_{\max}(X)$, $\ell_0 = -2k/\Lambda$ ▷ Here u_0 and ℓ_0 are the initial barrier values

$\widehat{u} = u_0$ and $\widehat{\ell} = \ell_0$ ▷ Here $\widehat{u}$ and $\widehat{\ell}$ are the current barrier values

$j = 0$ ▷ j will be the index of the current iteration

$A_0 = X$, $B_0 = \mathbf{0}$

while $\widehat{u} - \widehat{\ell} > \alpha + u_0 - \ell_0$ **do** ▷ Start of iteration j

  Compute $R_t(A_j, B_j, \ell_j, u_j)$ and hence $p(v_t)$ for all vectors v_t

  Compute N_j

  Sample N_j vectors $v_1, \dots v_{N_j}$ according to p

  Set $W_j \leftarrow 0$

  for every subphase $i = 1 \dots N_j$ **do** ▷ Start of subphase i

    $w_i \leftarrow \sqrt{\frac{\varepsilon}{q \cdot R_i(A_j, B_j, u_j, \ell_j)}} \cdot v_i$

    $W_j \leftarrow W_j + w_i w_i^\intercal$

    $\widehat{u} \leftarrow \widehat{u} + \overline{\delta}_{u,j}$

    $\widehat{\ell} \leftarrow \widehat{\ell} + \overline{\delta}_{\ell,j}$

    if $\widehat{u} - \widehat{\ell} > \alpha + u_0 - \ell_0$ **then**

      Stop at the current subphase ▷ End of subphase i

  $A_{j+1} \leftarrow A_j + W_j$

  $B_{j+1} \leftarrow Z(A_{j+1} - X)Z$

  $j = j + 1$ ▷ End of iteration j

Return $M = A_j$

▶ **Lemma 11.** *The condition number of the returned matrix A_τ after τ iterations is at most $1 + O(\varepsilon) \cdot \max\{1, T/k\}$. Moreover, it holds that*

$$\lambda_{\min}(A_\tau) \geq c \cdot (1 - O(\varepsilon)) \cdot \lambda_{k+1}(X) \min\{1, k/T\},$$

for some constant c.

The above result is based on the following technical lemma:

▶ **Lemma 12.** *For all iterations j, the following invariant is preserved by Algorithm 2*

$$A_j \prec (1 - \eta) u_j \cdot I \qquad \text{and} \qquad P_{\mathcal{V}} B_j P_{\mathcal{V}} \succeq (\ell_j + |\ell_j|\eta) P_{\mathcal{V}},$$

for some parameter $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$.

In order to prove Lemma 12, we need to develop a sequence of results. For the moment, we fix an iteration j. Recall that in this iteration the algorithm samples N_j vectors independently from $\{v_i\}_{i=1}^m$ such that each sampled vector v_i is further scaled to w_i. Moreover, the algorithm keeps track of the matrix

$$W_j \triangleq \sum_{i=1}^{N_j} w_i w_i^\intercal.$$

Our choice of N_j ensures that, with high probability, the matrix W_j has bounded eigenvalues with respect to the matrix A_j.

▶ **Lemma 13.** *Assume that the number of samples satisfies*

$$N_j \leq \left(\frac{\varepsilon}{4\rho_j} \cdot \lambda_{\min}\left[(u_j I - A_j)^{-1}\overline{M}\right] \cdot \frac{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}{\operatorname{tr}\left[(u_j I - A_j)^{-1}\overline{M}\right]} \right)^{2\varepsilon/q} \cdot \rho_j \cdot \frac{1}{\lambda_{\max}\left((u_j I - A_j)^{-1}\overline{M}\right)}.$$

Then it holds that

$$\mathbf{P}\left[\mathbf{0} \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \geq 1 - \frac{\varepsilon}{2n}.$$

Notice that if $W_j \preceq \frac{1}{2}(u_j I - A_j)$, W_j's contribution towards the direction of A_j's eigenvector associating with its largest eigenvalue is upper bounded. Thus, conditioned on this event, we can control better the eigenvalues of the resulting matrix $A_{j+1} = A_j + W_j$. Formally, we show the following result:

▶ **Lemma 14.** *It holds that*

$$\mathbf{E}\left[\Phi^{u_{j+1}}(A_{j+1}) \middle| 0 \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \leq \Phi^{u_j}(A_j) \quad \textit{and}$$
$$\mathbf{E}\left[\Phi_{\ell_{j+1}}(B_{j+1}) \middle| 0 \preceq W_j \preceq \frac{1}{2}(u_j I - A_j)\right] \leq \Phi_{\ell_j}(B_j).$$

Finally, we show that the careful choice of N_j ensures that a sufficiently large number of vectors are sampled in each iteration. This implies that the total number of iterations executed by the algorithm cannot be too large. The result is summarised below:

▶ **Lemma 15.** *With probability at least* $4/5$*, Algorithm 2 finishes in at most*

$$\tau \leq \frac{80q}{3\varepsilon^2} \cdot \frac{1}{c_N} \cdot \Lambda^{(1+2\varepsilon)/q}$$

iterations, where $c_N = \Omega\left((1/(\operatorname{poly}(n))^{2\varepsilon/q}\right)$.

We are now ready to prove the main technical lemma.

Proof of Lemma 12. By Lemma 13, Lemma 15 and the union bound, with probability at least $3/4$, all matrices picked in

$$\tau \leq \frac{80q}{3\varepsilon^2} \cdot \frac{1}{c_N} \cdot \Lambda^{(1+2\varepsilon)/q} \leq \frac{80q}{3\varepsilon^2} \cdot n^{c/q}$$

iterations, for some small constant $c < q$, satisfy $W_j \preceq \frac{1}{2}(u_j I - A_j)$ for all iterations j. Therefore, by Lemma 14 and conditioning on the event that $\forall i : W_i \preceq 1/2 \cdot (u_i I - A_i)$ we have that

$$\mathbf{E}\left[\Phi^{u_j}(A_j) \middle| \forall i : W_i \preceq (1/2) \cdot (u_i I - A_i)\right] \leq \Phi^{u_0}(A_0) \leq \frac{T}{2^q}$$

and

$$\mathbf{E}\left[\Phi_{\ell_j}(B_j) \middle| \forall i : W_i \preceq (1/2) \cdot (u_i I - A_i)\right] \leq \Phi_{\ell_0}(B_0) \leq k \cdot \left(\frac{\Lambda}{2k}\right)^q.$$

By Markov's inequality, it holds with high probability that

$$(\Phi^{u_j}(A_j))^{1/q} = O\left(T^{1/q} \cdot \tau^{1/q}\right) \quad \text{and} \quad \left(\Phi_{\ell_j}(B_j)\right)^{1/q} = O\left(k^{1/q} \cdot \frac{\Lambda}{k} \cdot \tau^{1/q}\right).$$

For any eigenvalue of A_j, say λ_i, we have

$$(u_j - \lambda_i)^{-q} \leq (u_j - \lambda_{\max}(A_j))^{-q} < \sum_{t=n-T+1}^{n} (u_j - \lambda_t(A_j))^{-q} = \Phi^{u_j}(A_j).$$

Therefore, it holds that

$$\lambda_i < u_j - (\Phi^{u_j}(A_j))^{-1/q} \leq u_j - O\left(\frac{1}{T^{1/q}} \cdot \frac{1}{\tau^{1/q}}\right) \leq u_j - O\left(\frac{2}{T^{1/q}} \cdot \left(\frac{\varepsilon^2}{qn^{c/q}}\right)^{1/q}\right).$$

Since u_j is $O(1/\varepsilon^2)$ and $T \leq n$, we can choose $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$ such that $A_j \prec (1-\eta)u_j I$.

The second statement can be shown in a similar way, i.e., we show that for any nonzero eigenvalue λ_i of B_j, it holds that $\lambda_i \geq \ell_j + \left(\Phi_{\ell_j}(B_j)\right)^{-1/q}$. Hence

$$\lambda_i \geq \ell_j + \Omega\left(\frac{1}{k^{1/q}} \cdot \frac{k}{\Lambda} \cdot \left(\frac{\varepsilon^2}{qn^{c/q}}\right)^{1/q}\right).$$

Since $|\ell_j| = O\left(\frac{k}{\Lambda} \cdot 1/\varepsilon\right)$ and $k \leq n$, we can choose $\eta = O\left(\frac{\varepsilon^{2+2/q}}{n^{2/q}}\right)$ such that $P_{\mathcal{V}} B_j P_{\mathcal{V}} \succeq (\ell_j + |\ell_j|\eta) P_{\mathcal{V}}$. ◀

Proof sketch of Lemma 11. Notice that it holds for any iteration j that

$$\frac{\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}}{\overline{\delta}_{u,j}} = \frac{6\varepsilon}{1+3\varepsilon},$$

which implies that

$$\overline{\delta}_{u,j} = \frac{1+3\varepsilon}{6\varepsilon}\left(\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}\right) \geq \frac{1}{6\varepsilon}\left(\overline{\delta}_{u,j} - \overline{\delta}_{\ell,j}\right).$$

Let u_τ and ℓ_τ be the barrier values when the algorithm terminates, and our goal is to show that

$$\frac{u_\tau}{\ell_\tau} = \left(1 - \frac{u_\tau - \ell_\tau}{u_\tau}\right)^{-1} = 1 + O(\varepsilon) \cdot \max\{1, T/k\},$$

which suffices to prove that

$$\frac{u_\tau - \ell_\tau}{u_\tau} = O(\varepsilon) \cdot \max\{1, T/k\}.$$

By definition, we know that

$$\frac{u_\tau - \ell_\tau}{u_\tau} \leq \frac{u_0 - \ell_0 + \alpha}{u_0 + (6\varepsilon)^{-1}\alpha} \leq \frac{3 + 6k/\Lambda}{2 + (6\varepsilon)^{-1}4k/\Lambda} \leq O(\varepsilon) \cdot \max\{1, T/k\},$$

where the last inequality holds by the definition of Λ. Similar with [22], in the full version of the paper we prove that

$$\lambda_{\min}(A_K) \geq \frac{\theta_{\min}\lambda_{k+1}(X)/2}{\left((\lambda_{k+1}(X)/2)^{1/2} + \theta_{\min}^{1/2} + \theta_{\max}^{1/2}\right)^2}.$$

assuming that $\lambda_{\max}(A_\tau) \leq \theta_{\max}$ and $\lambda_{\min}\left(B_K\big|_{S'}\right) \geq \theta_{\min}$. By setting $\theta_{\min} = \ell_\tau$ and $\theta_{\max} = u_\tau$, the inequality above implies the second statement of the lemma. ◀

The results for the total number of edges (vectors) sampled as well as the assigned costs are summarised below. Due to space constraints, we defer the proofs to the full version of the paper.

▶ **Lemma 16.** *With probability at least* $3/4$*, Algorithm 2 terminates after choosing at most*

$$K = \frac{20 \cdot q\ k}{3 \cdot \varepsilon^2}$$

vectors.

▶ **Lemma 17.** *It holds with constant probability that* $\sum_{i=1}^{m} c_i \cdot cost_i = O(1/\varepsilon^2) \cdot \min\{1, k/T\}$.

Runtime analysis

Now we discuss a fast approximation of the quantities needed for our subgraph sparsification algorithm. We fix an arbitrary iteration j, and drop this subscript for simplicity. A careful inspection tells us that the efficiency of our algorithm is based on the fast approximation of the following quantities:

1. $v_i^\intercal Z \left(P_{\mathcal{V}} \left(B - \ell I\right) P_{\mathcal{V}}\right)^\dagger Z v_i$
2. $\lambda_{\max} \left(P_{\mathcal{V}}(B - \ell I)P_{\mathcal{V}}\right)^\dagger$
3. $\lambda_{\min} \left[(uI - A)^{-1}\overline{M}\right]$
4. $\lambda_{\max} \left[(uI - A)^{-1}\overline{M}\right]$
5. $\operatorname{tr} \left[(uI - A)^{-1}\overline{M}\right]$
6. $v_i^\intercal \left(uI - A\right)^{-1} v_i$

These are precisely the nontrivial quantities required to compute the values N and $R_i(A, B, u, \ell)$ for all vectors v_i. As previously mentioned, for the first two items we use the approximate projection $P_{\mathcal{V}}$ instead of the actual projection P_S on the bottom k eigenspace of X. This is done in order to overcome the expensive exact computation of P_S. We also remark that, while $P_{\mathcal{V}} = VV^\intercal$ for some unitary matrix V is used in our previous analysis, we do not need to compute the matrices V or $P_{\mathcal{V}}$ explicitly. Instead, it suffices to compute the matrix $\mathbb{V} \triangleq L_{G+W}^{-1/2} V$ in order to approximate our required quantities (1), (2). The fast computation of $\mathbb{V}$ builds upon the work of [2] and our result is summarised in Lemma 10.

Once we have access to the matrix $\mathbb{V}$, we can efficiently approximate the above quantities (1)–(6). The techniques we used are inspired from the previous work [4, 26]. However, the presence of the matrix $\overline{M}$ as well as the projection $P_{\mathcal{V}}$ make the computations nontrivial and require extra work. We summarise our results bellow and refer the reader to the full version of the paper for the details of each individual approximation.

▶ **Lemma 18.** *Let j be an arbitrary iteration of Algorithm 2. We can approximately compute the quantities (1)–(6), for all vectors v_i in time*

$$t_{\text{iteration}} = \widetilde{O}\left(\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^3\right).$$

The running time of Algorithm 2 is analysed in the next lemma.

▶ **Lemma 19.** *Assuming Algorithm 2 finishes in $\tau = O\left(q \cdot n^{O(1/q)}/\varepsilon^2\right)$ iterations, then the total running time is*

$$t_{\text{alg}} = \widetilde{O}\left(\min\left\{n^\omega, \frac{mk + nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)}\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^5\right).$$

Proof. By Lemma 10, we can compute the matrix $\mathbb{V}$ in time $t_{10} = \min\left\{O(n^\omega), \widetilde{O}\left(\frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right)\right\}$. This is computed only once and will be used throughout every iteration.

By Lemma 18, the running time in each iteration is

$$t_{\text{iteration}} = \widetilde{O}\left(\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^3\right).$$

Thus, the algorithm's overall running time is

$$\begin{aligned} t_{\text{alg}} &\triangleq t_{10} + \tau \cdot t_{\text{iteration}} \\ &= \widetilde{O}\left(\min\left\{n^\omega, \frac{mk+nk^2}{\sqrt{\lambda_{k+1}(X)}}\right\} + q \cdot n^{O(1/q)}\left(\frac{mn^{2/q}}{\varepsilon^{2+2/q}} + \min\left\{n^\omega, mk + nk^2 + k^\omega\right\}\right)/\varepsilon^5\right). \end{aligned}$$

◀

4 Proof of the main theorem

Finally we apply our fast SDP solver and the subgraph sparsification algorithm to design an algorithm for the spectral-augmentability problem, and prove Theorem 1. We first give an overview of the main algorithm: for any input $G = (V, E)$, the set E_W of candidate edges, and parameter k, our algorithm applies the doubling technique to enumerate all the possible γ under which the input instance is (k, γ)-spectrally augmentable: starting with the initial γ, which is set to be $1/n^{1/q}$ and increases by a factor of 2 each time, the algorithm runs the SDP solver, a subgraph sparsification algorithm, and a Laplacian solver to verify the algebraic connectivity of the output of our subgraph sparsification algorithm. The algorithm terminates if the algebraic connectivity is greater than some threshold at some iteration, or it is below the initial threshold. See Algorithm 3 for formal description.

The following lemma will be used in our analysis.

▶ **Lemma 20.** *Let $\gamma > 0$. If $G = (V, E)$ is $(k, \gamma\Delta)$-spectrally-augmentable with respect to $W = (V, E_W)$, then the SDP solver finds a feasible solution $(\widehat{\lambda}, w)$ to P-SDP$(G, W, k, (1-\delta')\gamma)$, and the subgraph sparsification algorithm with input $G, E_W, k, \varepsilon, q$ and weights $\{w_e : e \in E \cup E_W\}$ will find a graph $H = (V, E \cup F)$ with $F \subseteq E_W$, $\lambda_2(L_H) \geq c_1\gamma^2 \cdot \Delta$, $|F| \leq O(qk/\varepsilon^2)$ and total new weights of edges in F at most $O(k/\varepsilon^2)$.*

Proof. If G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W, then there exists a feasible solution to P-SDP(G, W, k, γ) and our SDP solver will find a solution $(\widehat{\lambda}, w)$ to P-SDP$(G, W, k, (1-\delta')\gamma)$, for any constant $\delta' > 0$. Note that $\widehat{\lambda} \geq (1-\delta')\gamma$. Now we use the subgraph sparsification algorithm to sparsify the SDP solution.

We apply Theorem 9 to graphs G, W, by setting $V = \text{im}(L_{G+W}) = \ker(L_{G+W})^\perp$, $X = \left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)_{|V}$ and $Y_e = w_e\left(L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2}\right)_{|V}$, and $\overline{M} = \sum_{e\in E_W} Y_e$, $K = O(qk/\varepsilon^2)$, $\lambda^* = \lambda_{k+1}(X)$ and $cost_e = \frac{w_e}{\sum_{f\in E_W} w_f}$. Note that $T = \lceil \text{tr}(\overline{M}) \rceil \leq k$. This is true since $\sum_{e\in E_W} w_e \leq k$, $L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2} \preceq I$, $L_{G+W}^{\dagger/2} L_e L_{G+W}^{\dagger/2}$ is a rank one matrix and has trace at most 1. We get a set of coefficients $\{c_e\}$ supported on at most K edges, such that

$$\begin{aligned} C(1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) &\leq \lambda_{\min}\left(X + \sum_{e\in E_W} c_e Y_e\right) \\ &\leq \lambda_{\max}\left(X + \sum_{e\in E_W} c_e Y_e\right) \leq 1 + O(\varepsilon). \end{aligned}$$

Algorithm 3 Algorithm for augmenting the algebraic connectivity.

Require: the base graph $G = (V, E)$, and the set E_W of m edges defined on V, and $k \in \mathbb{Z}^+$.

$\gamma_0 \leftarrow 1/n^{\frac{1}{q}}$;
$\gamma \leftarrow \gamma_0$;
$\alpha \leftarrow 0$;
$F \leftarrow \emptyset$; ▷ the set of edges added to G
while $\gamma < 1$ **do**
  $\gamma \leftarrow 2 \cdot \gamma$, and run the SDP solver from Theorem 2 for P-SDP(G, W, k, γ)
  if the solver certifies that P-SDP(G, W, k, γ) is infeasible **then**
    if $\alpha = 0$ **then**
      Abort and output **Reject**.
    else
      return graph $H = (V, E(G) \cup F)$. ▷ $\lambda_2(L_H) \geq c_1\alpha^2\Delta$
  else the solver finds a feasible solution for P-SDP$(G, W, k, 0.9\gamma)$ with weights $\{w_e\}_{e\in E_W}$
    $\alpha \leftarrow \gamma$
    Let $H = (V, E(G) \cup F)$ be the output of our subgraph sparsification algorithm with edge weights $\{w_e\}_{e\in E_W}$, q, k and a sufficiently small constant ε.
    $\eta_2 \leftarrow$ a 1.1-approximation of $\lambda_2(L_H)$ ▷ apply the Laplacian solver to compute η_2
    if $\eta_2 \leq O\left(\Delta \cdot n^{-2/q}\right)$ **then**
      Abort and output **Reject**.

From the above and the fact that $T \leq k$, $K = O(qk/\varepsilon^2)$, we have that

$$\begin{aligned}\lambda_2\left(L_G + \sum_e c_e w_e L_e\right) &\geq C(1 - O(\varepsilon)) \cdot \min\{1, K/T\} \cdot \lambda_{k+1}(X) \cdot \lambda_2\left(L_G + \sum_e w_e L_e\right)\\ &\geq C' \cdot \frac{\lambda_{k+2}(L_G)}{4\Delta} \cdot \widehat{\lambda} \cdot \Delta\\ &= \frac{C'}{4} \cdot \widehat{\lambda} \cdot \lambda_{k+2}(L_G)\end{aligned}$$

for some constant $C' > 0$, where the last inequality follows from the fact that

$$\lambda_i(X) = \lambda_i\left(\left(L_{G+W}^{\dagger/2} L_G L_{G+W}^{\dagger/2}\right)\Big|_V\right) \geq \frac{\lambda_{i+1}(L_G)}{4D},$$

for any $i \geq 1$.

▷ Claim 21. It holds that $\lambda_{k+2}(L_G) \geq \lambda_{\mathrm{OPT}}$, the maximum algebraic connectivity of adding a subset set of k edges from E_W to G.

Proof. Let L_R be the Laplacian matrix of the graph which is formed by the optimum solution R. Then $\dim\ker(L_R) \geq n-k$ as $\mathrm{rank}(L_R) \leq |E| \leq k$. Consider the space S spanned by all the eigenvectors of L_G corresponding to $\lambda_2(L_G), \ldots, \lambda_{k+2}(L_G)$. Since $\dim(S) + \dim\ker(L_R) > n$, there exists a unit vector $v \in \ker(L_R) \cap \dim(S)$ such that $v \perp \mathbf{1}$, and

$$v(L_G + L_R)v^\intercal \leq \lambda_{k+2}(L_G) + 0 = \lambda_{k+2}(L_G).$$

This further implies that $\lambda_{\mathrm{OPT}} = \lambda_2(L_G + L_R) \leq \lambda_{k+2}(L_G)$. ◁

Therefore, if we let $F = \{e : e \in E_W, c_e > 0\}$ and set the edge weights to be $\{c_e \cdot w_e : e \in F\}$, then the resulting graph $H = (V, E + F)$ with the corresponding weights satisfies that

$$\lambda_2(L_H) = \lambda_2\left(L_G + \sum_e c_e w_e L_e\right) \geq c \cdot \gamma \cdot \lambda_{\mathrm{OPT}} \geq c \cdot \gamma^2 \Delta$$

for some constant $c > 0$, where the last inequality follows from the assumption G is $(k, \gamma\Delta)$-spectrally-augmentable with respect to W and thus $\lambda_{\mathrm{OPT}} \geq \gamma\Delta$. Since

$$\sum_{e \in E_W} cost_e \cdot c_e \leq O(1/\varepsilon^2) \min\{1, K/T\} = O(1/\varepsilon^2),$$

the total weights of added edges become

$$\sum_{e \in E_W} c_e w_e = \left(\sum_{e \in E_W} w_e\right) \cdot \left(\sum_{e \in E_W} cost_e \cdot c_e\right) O(1/\varepsilon^2) \cdot k = O(k/\varepsilon^2). \qquad \blacktriangleleft$$

Finally, we are ready to prove the main theorem of the paper.

Proof of Theorem 1. Let G and W be the input to Algorithm 3. Note that the algorithm only returns a subgraph H with $\lambda_2(L_H) \geq c_1\gamma_0^2\Delta$, and H contains at most $K = O(kq)$ edges from E_W. Hence, if G is not $(O(kq), c_1\gamma_0^2\Delta)$-spectrally-augmentable with respect to W, then the algorithm will reject the input instance.

Without loss of generality, in the following analysis we assume that G is $(k, \lambda_\star\Delta)$-augmentable for some $\lambda_\star > \gamma_0$, where $\lambda_\star\Delta$ is the optimum solution. In this case, by the geometric search over γ in the algorithm, when $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$, the SDP solver will find a feasible solution for P-SDP$(G, W, k, 0.9\gamma)$ and the graph H returned by the subgraph sparsification algorithm with input G, W, q, k and constant ε satisfies that $\lambda_2(L_H) \geq c_1\gamma^2\Delta \geq c_1'\lambda_\star^2\Delta$. If $\gamma \geq \lambda_\star$, then the algorithm will either return the graph H that we constructed corresponding to the value $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$, or finds a graph H with $\lambda_2(L_H) \geq c_1\gamma^2\Delta \geq c_1'\lambda_\star^2\Delta$. By Lemma 20, the number of added edges and the total sum of their weights are $O(qk)$ and $O(k)$, respectively.

Furthermore, since $\lambda_\star \geq \gamma_0$, it only takes $O(\log n)$ iterations to reach γ with $\gamma \in (\frac{\lambda_\star}{2}, \lambda_\star)$. In each iteration, by Theorem 2, the running time for solving P-SDP$(G, W, k, 0.9\gamma)$ for $\gamma \geq \gamma_0$ is $\widetilde{O}(m+n)/\gamma^2) = \widetilde{O}((m+n)n^{O(1/q)})$; by Theorem 9, the time for applying subgraph sparsification with input G, W and constant ε is $\widetilde{O}(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\})$. For the latter, we note that whenever we apply the subgraph sparsification from Theorem 9, the corresponding matrix X satisfies that

$$\lambda_{k+1}(X) \geq \frac{\lambda_{k+2}(L_G)}{4\Delta} \geq \frac{\lambda_{\mathrm{OPT}}}{4\Delta} = \frac{\lambda_\star\Delta}{4\Delta} \geq \frac{\gamma_0\Delta}{4\Delta} = \Omega\left(n^{-1/q}\right)$$

and thus we obtain the claimed runtime. Furthermore, we can compute an estimate η_2 of $\lambda_2(L_H)$ by the algorithm given in [41], which takes $\widetilde{O}(|E(H)|+n) = \widetilde{O}(n+k)$ time. Thus, the total running time is $\widetilde{O}\left(\min\left\{qn^{\omega+O(1/q)}, q(m+n)n^{O(1/q)}k^2\right\}\right)$. This completes the proof of the theorem. $\blacktriangleleft$

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS'16)*, pages 335–344, 2016.

2 Zeyuan Allen-Zhu and Yuanzhi Li. Doubly accelerated methods for faster CCA and generalized eigendecomposition. In *34th International Conference on Machine Learning (ICML'17)*, pages 98–106, 2017.

3 Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. Near-optimal design of experiments via regret minimization. In *34th International Conference on Machine Learning (ICML'17)*, pages 126–135, 2017.

4 Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond multiplicative updates. In *47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 237–245, 2015.

5 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM*, 63(2):12, 2016.

6 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

7 Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing Markov chain on a graph. *SIAM review*, 46(4):667–689, 2004.

8 Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 167–176, 2008.

9 Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

10 Julia Chuzhoy and Sanjeev Khanna. An $O(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*, pages 437–441, 2009.

11 Michael B. Cohen, Jelani Nelson, and David P. Woodruff. Optimal approximate matrix product in terms of stable rank. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP'16)*, pages 1–14, 2016.

12 Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, pages 821–840, 2016.

13 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 750–759, 1999.

14 Jittat Fakcharoenphol and Bundit Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. *SIAM Journal on Computing*, 41(5):1095–1109, 2012.

15 Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

16 Arpita Ghosh and Stephen Boyd. Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6605–6611, 2006.

17 Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM review*, 50(1):37–66, 2008.

18 Russell Impagliazzo, Noam Nisan, and Avi Wigderson. Pseudorandomness for network algorithms. In *26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 356–364, 1994.

19 Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP= PSPACE. *Journal of the ACM (JACM)*, 58(6):30, 2011.

20 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017.

21 Jonathan A Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262, 2013.

22 Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *42nd Annual ACM Symposium on Theory of Computing (STOC'10)*, pages 57–66, 2010.

23 Guy Kortsarz, Robert Krauthgamer, and James R Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

24 Bundit Laekhanukit. Parameters of two-prover-one-round game and the hardness of connectivity problems. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1626–1643, 2014.

25 Yin Tat Lee and He Sun. An SDP-based algorithm for linear-sized spectral sparsification. In *49th Annual ACM Symposium on Theory of Computing (STOC'17)*, pages 678–687, 2017.

26 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. *SIAM Journal on Computing*, 47(6):2315–2336, 2018.

27 Marina Meila and Jianbo Shi. Learning segmentation by random walks. In *Advances in Neural Information Processing Systems*, pages 873–879, 2001.

28 William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. *Proceedings of the 17th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

29 Milena Mihail. Conductance and convergence of Markov chains-a combinatorial treatment of expanders. In *30th Annual IEEE Symposium on Foundations of Computer Science (FOCS'89)*, pages 526–531, 1989.

30 Damon Mosk-Aoyama. Maximum algebraic connectivity augmentation is NP-hard. *Operations Research Letters*, 36(6):677–679, 2008.

31 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

32 Aleksandar Nikolov, Mohit Singh, and Uthaipon Tao Tantipongpipat. Proportional volume sampling and approximation algorithms for a-optimal design. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 1369–1386, 2019.

33 Lorenzo Orecchia and Nisheeth K Vishnoi. Towards an sdp-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition. In *22th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*, pages 532–545, 2011.

34 Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1256–1266, 2014. doi:10.1137/1.9781611973402.93.

35 Richard Peng. Algorithm Design Using Spectral Graph Theory. *PhD Thesis*, 2013. doi:10.1184/R1/6714635.v1.

36 Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *SIAM Journal on Computing*, 46(2):710–743, 2017.

37 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

38 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Information Theory*, 42(6):1710–1722, 1996.

39 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

40 Leslie G. Valiant. Graph-theoretic properties in computational complexity. *J. Comput. Syst. Sci.*, 13(3):278–285, 1976.

41 Nisheeth K. Vishnoi. $Lx = b$. *Foundations and Trends in Theoretical Computer Science*, 8(1-2):1–141, 2013.

42 Gilles Zémor. On expander codes. *IEEE Trans. Information Theory*, 47(2):835–837, 2001.