Directed Spectral Graph Theory: Sparsification and Asymmetric Linear System Solver

Yanlin Duan
Department of Computer Science
Columbia University
yd2380@columbia.edu

Jiahui Liu
Department of Computer Science
Columbia University
jl4161@columbia.edu

Serena Liu
Department of Electrical Engineering
Columbia University
yl2904@columbia.edu

David Pu
Department of Electrical Engineering
Columbia University
sp3396@columbia.edu

May 12, 2017

Abstract

This paper is our report for the Advanced Algorithm class final project. In this report, we study fast algorithms for solving directed Laplacian linear systems. A recent body of work by Cohen *et al.* has focused on introducing a novel notion of spectral approximation for directed graphs. They provide a general framework for solving asymmetric linear systems using the notion of spectral approximation and show how to solve directed Laplacian linear systems associated with Eulerian Graphs in almost-linear-time.

In this final report, we present the general framework for solving linear systems of directed Laplacian, with focus on the key theorems and algorithms in paper [CKP+16a] and the related prior work in [CKP+16b]. We hope this project opens the door for our studies and research into directed spectral graph theory, and that this final project will serve as a brief technical report for our fellow classmates and researchers in the spectral graph research community.

1 Introduction

Spectral graph theory is the study and exploration of graphs through the eigenvalues and eigenvectors of matrices naturally associated with those graphs. Since its first emergence in the 1950s and 1960s, spectral graph theory, especially the undirected case, has been well studied and has a rich collection of algorithmic tools, most of which are rigorously analyzed. Despite the extensive and a successful line of research on the undirected case, there exists a longstanding algorithmic discrepancy between the state of algorithmic spectral graph theory in undirected and directed settings. The combinatorial properties of graph that are intrinsic to the undirected graph and are used to accelerate the solution of Laplacian linear systems have remained elusive for directed graphs. It was until 2016 that, for the first time, fast algorithm for computing the stationary distributions has been proposed [CKP+16b], which takes less time than is required to find the kernel of a general matrix.

There are three main reasons for the algorithmic performance gap between directed Laplacian linear solvers and undirected linear solvers:

Lack of efficient approach to compute the kernel/stationary distribution: The kernel of a Laplacian plays an important role in solving many problems. We can understand its importance from two perspectives: from a linear algebraic way, assume we can compute the kernel of $\mathcal{L}$ as the span of $\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_n$, then as long as we find one solution for the linear system $\mathcal{L}\vec{x} = \vec{b}$, say $\vec{x}_0$, we immediately know the general solution for $\mathcal{L}\vec{x} = \vec{b}$ is $\vec{x}_0 + a_1\vec{u}_1 + a_2\vec{u}_2 + \cdots + a_n\vec{u}_n$. (since $\mathcal{L}(\vec{x}_0 + \text{linear combination of}\{\vec{u}_1, \cdots, \vec{u}_n\}) = \mathcal{L}\vec{x}_0 + \vec{0} = \vec{b}$). Therefore, finding the kernel space of $\mathcal{L}$ helps a lot in the linear system solver.

From a Markov chain's perspective, vectors in the kernel of a Laplacian correspond to stationary distributions of the corresponding random walk (up to rescaling), and computing the stationary distribution is essential in the analysis of Markov chains, as persists for many fundamental problems associated with random walks, such as Google's famous PageRank algorithm.

Asymmetric, and non-positive semidefiniteness: The fact that directed Laplacians are asymmetric matrices makes essentially every algorithmic tool for solving undirected Laplacians unavailable in the directed setting, since we can no longer view $\mathcal{L}$ as a sum of semidefinite contributions from each of the edges. Consider a simple example of $\mathcal{L} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$, one may wish to solve the problem by considering the symmetrization of directed graph Laplacian: $\frac{1}{2}(\mathcal{L}+\mathcal{L}^\top)$. However, $\frac{1}{2}(\mathcal{L}+\mathcal{L}^\top) = \begin{bmatrix} 1 & -1/2 \\ -1/2 & 0 \end{bmatrix}$ has one eigenvalue $\frac{1-\sqrt{2}}{2} < 0$, which means this new matrix is not even positive semidefinite.

Lack of directed graph sparsification: Because of the nice properties of (cut) sparisification defined in the undirected setting, it is tempting to generalize it to the directed graphs. But it can be shown that, such sparsifier does not generally exist, even for a weaker notion of spectral sparisification. Let's consider a simple example of a complete bipartite graph G, with four vertices on each side and all edges going from one side to another, as shown in Fig 1.

If such a sparsifier does exist, it has to work for every cut in G. Now consider a family of cuts described in Fig 1. Since there is only one outgoing edge from the highlighted partition, omitting this edge will change the weight from 1 to 0, which is definitely not within a multiplicative factor of $1 \pm \epsilon$. Since all such edges need to be preserved for every cut in this family of cuts, it is not hard to see that we will need $\Omega(n^2)$ edges, which exceeds the cut sparsification requirement of $O(n \log n/2)$.

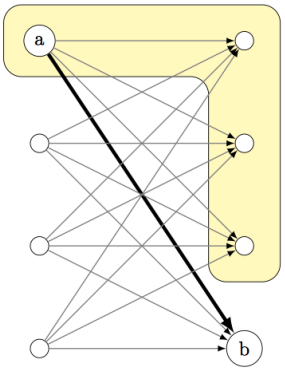

Figure 1: An example of the family of cuts described for directed bipartite graph. (Figure 1.1 from [CKP+16a])

1.1 Problem Definition and Overview

Problem definition: Given a directed Laplacian associated with a graph with n vertices and m edges, we want to find a directed Laplacian linear system solver, i.e. a solver that can (approximately) solve for $\mathcal{L}\vec{x} = \vec{b}$ in an efficient way.

In spite of the fundamental differences between the undirected and directed settings, as well as the general difficulties of applying algorithmic tools of the undirected spectral graph theory to the directed case, Cohen *et al.* present in paper [CKP+16a] the first almost-linear-time algorithms for solving directed Laplacian systems, improved upon the prior work in[CKP+16b]. See Table 1 for running time comparison.

Cohen *et al.* split the problem into three steps: reduction (reduce the problem to the special case of solving Eulerian Laplacians), sparsification (construct sparsifier for directed graph) and directed Eulerian Laplacian solver.

Cohen *et al* STOC'17[CKP+16a]	Cohen *et al* FOCS'16 [CKP+16b]	Le Gall ISSAC'14 [Le 14]
$\tilde{O}(m+n2^{O(\sqrt{\log\kappa\log\log\kappa})})$	$\tilde{O}(m^{3/4}n+mn^{2/3})$	$\Theta(\min(mn, n^{2.3728639}))$

Table 1: Time Complexity for Directed Laplacian System Solvers

1.2 Definition

For basic notations used in this report, please refer to Appendix A. Here we present important definitions used throughout the report:

Definition 1 (Directed Laplacian Matrix)**.** *A matrix $\mathcal{L} \in \mathbb{R}^{n\times n}$ is called a directed Laplacian if $\mathcal{L}_{ij} \leq 0$ for all $i \neq j$ and $\mathcal{L}_{ii} = -\sum_{j\neq i}\mathcal{L}_{ji}$ for all i.*

Definition 2 (Decomposition of Directed Laplacian Matrix)**.** *We associate a graph $G_{\mathcal{L}} = (V, E, w)$ with each directed Laplacian. We may decompose $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$, where $\mathbf{D}$ is the diagonal matrix that $\mathbf{D}_{ii} = \mathcal{L}_{ii}$ = out degree of vertex i in the associated graph $G_{\mathcal{L}}$ and $\mathbf{A}$ is weighted adjacency matrix of $G_{\mathcal{L}}$ with $\mathbf{A}_{ij} = w_{ij}$ if $(i, j) \in E$ and 0 otherwise.*

2 Reductions

As mentioned in section 1.1, the first part of the algorithms of [CKP+16b] is to reduce the problem of solving the general directed Laplacian systems to the special case of solving Eulerian Laplacians (with preferred condition number). We first introduce the definition of Eulerian directed graph.

Definition 3 (Eulerian directed graph)**.** *A directed graph is Eulerian iff every graph vertex has equal indegree and outdegree.*

The intuition behind why Eulerian Laplacian systems are much easier to solve is that, for a Eulerian Laplacian, the kernel is the all-ones vector, and $\frac{1}{2}(\mathcal{L} + \mathcal{L}^\top)$ is symmetric and positive semidefinite (PSD). More importantly, for every directed Laplacian $\mathcal{L}$ that corresponds to a strongly connected graph, there is always a vector $\vec{x} \in \mathbb{R} > 0$ such that $\mathcal{L}\vec{x} = 0$. Therefore, we can construct a new Laplacian $\mathcal{L}' = \mathcal{L}\mathbf{X}$, where $\mathbf{X}$ is a diagonal matrix associated with $\vec{x}$, Then we have $\mathcal{L}'\vec{\mathbb{1}} = 0$.

Lemma 4 (Eulerian Scaling, [CKP+16b] Lemma 1)**.** *Given a directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ whose associated graph is strongly connected, there is a positive vector $\vec{x} \in \mathbb{R}^n$ such that $\mathcal{L}\cdot\mathbf{diag}(\vec{x})$ is an Eulerian Laplacian, where $\mathbf{diag}(\vec{x})$ is the diagonal matrix with $\mathbf{diag}(\vec{x})_{ii} = \vec{x}_i$. This is called Eulerian scaling.*

Once we can approximately compute the Eulerian Scaling, then the original problem is reduced to solving Eulerian Laplacian systems. We proceed by introducing the definition of diagonal dominant matrix.

Definition 5 (Diagonal Dominance)**.** *A possibly asymmetric matrix $\mathbf{A} \in \mathcal{R}^{n\times n}$ is called α-row diagonally dominant (RDD) if $\mathbf{A}_{ii} \geq (1+\alpha)\sum_{j\neq i}|A_{ij}|$ for all $i \in [n]$, α-column diagonally dominant (CDD) if $\mathbf{A}_{ii} \geq (1+\alpha)\sum_{j\neq i}|A_{ji}|$, and α-row- or column- diagonally dominant (RCDD) if it is both α-RDD and α-CDD. If $\alpha = 0$, we say $\mathbf{A}$ is RCDD, and α-RCDD if $\alpha > 0$.*

It is not hard to see that a general directed Laplacian (may be disconnected) is a row- or column-diagonally dominant (RCDD) linear system.

The first reduction There are two reductions involved in the process. The first reduction uses the following theorem from [CKP+16a] to show that we can solve such linear system by solving a small number of Laplacian systems in which the graphs are Eulerian:

Theorem 6 (Theorem 42 from [CKP+16b])**.** *Let $\mathbf{M}$ be an arbitrary $n \times n$ column- or row-diagonally-dominant matrix with diagonal $\mathbf{D}$. Let $\vec{b} \in im(\mathbf{M})$. Then for any $\epsilon \in (0, 1]$, one can compute a vector $\vec{x'}$ satisfying $||\mathbf{M}\vec{x'} - \vec{b}||_2 \leq \epsilon||\vec{b}||_2$, with high probability and in time*

$$O(\mathcal{T}_{solve}\log^2(\frac{n\cdot\kappa(\mathbf{D})\cdot\kappa(\mathbf{M})}{\epsilon}))$$

where $\mathcal{T}_{solve} = (nm^{3/4} + n^{2/3}m)(\log n)^3$.

Proof Sketch. We will only provide a proof sketch here, since the original proof of the theorem is non-trivial and the theorem is not our main focus in this report. The idea behind the proof of the theorem above is that one can first convert arbitrary CDD or RDD matrix $\mathbf{M}$ to a RCDD matrix $\hat{\mathbf{M}}$. Take CDD case as an example. $\hat{\mathbf{M}} = \alpha\mathbf{I} + \mathbf{M}\mathbf{D}^{-1}$, where $\mathbf{D}$ is the diagonal matrix. This can be viewed as first rescaling the matrix by its diagonal and then slightly increase its diagonal. This gives us an α-RCDD matrix where every off-diagonal entry is non-positive. RDD is very similar, except that we multiply the inverse of diagonal matrix from left-hand side, and that the rescaling is computed with respect to $\hat{\mathbf{M}}^\top$.

Take a $n \times n$ strictly α-RCDD matrix $\hat{\mathbf{M}}$, one can then define a $(n+1) \times (n+1)$ matrix $\mathcal{L} = \mathbf{C}\hat{\mathbf{M}}\mathbf{C}^\top$, where $\mathbf{C}$ is a $(n+1)\times$n matrix with top $n \times n$ being identity matrix $\mathbf{I}_n$, stacked with an all-negative-ones row $(-\vec{\mathbb{1}}^\top)$ at the bottom. Then clearly the top left $n \times n$ entries of $\mathcal{L}$ are a copy of $\hat{\mathbf{M}}$, and its rows and columns sum to 0 since the columns of $\mathbf{C}$ sum to 0. Also $\mathcal{L}$ has off-diagonal entries being negative and diagonal entries being positive. Therefore, it satisfies the definition of Eulerian Laplacian.

To solve for $\hat{\mathbf{M}}x = b$, it is equivalent to solve $\mathcal{L}\begin{pmatrix} x \\ 0 \end{pmatrix} = \begin{pmatrix} b \\ y \end{pmatrix}$ for some scalar y, since the upper left block is exactly $\hat{\mathbf{M}}$. A more detailed proof will show that the approximation error introduced by transforming from $\mathbf{M}$ to $\hat{\mathbf{M}}$ is not an intolerably large one, making this reduction plausible.

The second reduction There is another reduction that solves an arbitrarily ill-conditioned Eulerian Laplacian system (possibly exponential) by solving $O(\log(n\kappa))$ Eulerian Laplacian whose condition numbers are polynomial in n. The proof is shown in Appendix F of [CKP+16a]. It is not the most interesting part of the paper so we will skip it here.

3 Directed Graph Sparsification

3.1 Definition of Directed Graph Sparsifier

We have a very brief review of undirected graph sparsification in Appendix B just to give a comparison with our directed case.

If we want to directly generalize from the definition for undirected graph, both cut and spectral notions have serious issues: if using cut notion, good sparsifiers don't exists for some graphs; if using spectral notion, since $\mathcal{L}_G$ is not symmetric in the directed case, but using the quadratic form $x^\top\mathcal{L}_G x$ in the requirement $(1-\epsilon)x^\top\mathcal{L}_{G'}x \le x^\top\mathcal{L}_G x \le (1+\epsilon)x^\top\mathcal{L}_{G'}x$ symmetrizes the the thing and the form is not necessarily PSD.

To avoid these issues, Cohen *et al.* [CKP+16a] gave original definitions for asymmetric matrix approximation, directed graph approximation and Eulerian sparsifier based on properties of asymmetric matrices and Eulerian directed graphs.

Definition 7 (Asymmetric Matrix Approximation)**.** *A possibly asymmetric matrix matrix $\tilde{\mathbf{A}}$ is an $\epsilon-$approximation of $\mathbf{A}$ if: $\mathbf{U}_A$ is a symmetric PSD matrix with* $\ker(\mathbf{U}_A) \subseteq \ker(\tilde{\mathbf{A}} - \mathbf{A}) \cap \ker((\tilde{\mathbf{A}} - \mathbf{A})^\top)$ *and* $\left\|\mathbf{U}_A^{+/2}(\tilde{\mathbf{A}} - \mathbf{A})\mathbf{U}_A^{+/2}\right\|_2 \le \epsilon$*, where* $\mathbf{U}_A = \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$.

The above definition generalizes spectral approximation for both symmetric and asymmetric approximations and behaves predictably under perturbations. $\mathbf{U_A}$ and $\mathbf{U}_{\tilde{\mathbf{A}}}$ are symmetrizations of $\mathbf{A}$ and $\tilde{\mathbf{A}}$, and $\mathbf{U}_{\tilde{\mathbf{A}}}$ spectrally approximate $\mathbf{U_A}$ and so does the $\mathbf{A}$ and $\tilde{\mathbf{A}}$ pair. The kernel requirement ensures that if we write the spectral 2-norm in the form $\left\|\mathbf{U}_A^{+/2}(\tilde{\mathbf{A}} - \mathbf{A})\mathbf{U}_A^{+/2}\right\|_2 = \max_{\vec{x},\vec{y}\neq 0} \frac{\vec{x}^\top(\tilde{\mathbf{A}}-\mathbf{A})\vec{y}}{\sqrt{\vec{x}^\top\mathbf{U_A}\vec{x}\cdot\vec{y}^\top\mathbf{U_A}\vec{y}}}$, the maximization problem is bounded and thus we can have an equivalent definition using $\max_{\vec{x},\vec{y}\neq 0} \frac{\vec{x}^\top(\tilde{\mathbf{A}}-\mathbf{A})\vec{y}}{\sqrt{\vec{x}^\top\mathbf{U_A}\vec{x}\cdot\vec{y}^\top\mathbf{U_A}\vec{y}}} \le \epsilon$.

More specifically, there are several facts about the their notion of asymmetric approximation:

- It does not coincide with the standard notion in terms of symmetric matrices; In fact, it is a stronger notion.
- If $\tilde{\mathbf{A}}$ is an ϵ-approximation of $\mathbf{A}$, then $(1-\epsilon)\mathbf{U}_A \le \mathbf{U}_{\tilde{A}} \le (1+\epsilon)\mathbf{U}_A$.
- (Approximation Transitivity) If $\mathbf{C}$ is an ϵ-approximation of $\mathbf{B}$ and $\mathbf{C}$ is an ϵ-approximation of $\mathbf{B}$, then $\mathbf{C}$ is an $\epsilon(2+\epsilon)$-approximation of $\mathbf{A}$

By Lemma 4, we can scale any strongly-connected graph's Laplacian by a diagonal matrix to make it the Laplacian of an Eulerian directed graph. It is also shown in [CKP+16b] that we can find this scaling matrix efficiently.

The directed graph approximation follows that a graph approximates another graph if their Eulerian scalings are within small multiplicative factors of one another.

Definition 8 (Directed Graph Approximation)**.** *Given two Laplacians $\mathcal{L}, \tilde{\mathcal{L}}$ of directed strongly-connected graphs $G, \tilde{G}$, the Eulerian scaling $\tilde{\mathcal{L}}\tilde{\mathbf{X}}$ and $\mathcal{L}\mathbf{X}$ are given as in the definition in lemma 4. $\tilde{G}$ is an ϵ-approximation of G if:*

- $(1-\epsilon)\mathbf{X} \leq \tilde{\mathbf{X}} \leq (1+\epsilon)\mathbf{X}$
- *$\tilde{\mathcal{L}}\tilde{\mathbf{X}}$ is an ϵ-approximation of $\mathcal{L}\mathbf{X}$ by definition 7.*

Especially, if $\tilde{\mathbf{X}} = \mathbf{X}$, then we say $\tilde{G}$ is a strict ϵ-approximation of G

And the directed graph sparsifier follows from a directed graph approximation with constraint on number of nonzero entries. We denote $nnz(\mathbf{A})$ as the total number of nonzero entries in $\mathbf{A}$.

Definition 9. *The directed graph sparsifier $\tilde{G}$ is a an ϵ-approximation of a graph G and $nnz(\tilde{\mathcal{L}}) \leq \tilde{O}(n\epsilon^{-2})$, where n is the number of vertices. More importantly here, an Eulerian Laplaian $\tilde{\mathcal{L}}$ is an ϵ-sparsifier of Eulerian Laplacian $\mathcal{L}$ if $\tilde{\mathcal{L}}$ is a a strict ϵ-approximation of a graph $\mathcal{L}$ and $nnz(\tilde{\mathcal{L}}) \leq \tilde{O}(n\epsilon^{-2})$.*

3.2 Sampling Procedure

Similar to undirected graph sparsification, we first need a random sampling procedure as a subroutine for our whole sparsification algorithm.

The main tool is a concentration bound theorem which shows that by sampling the entries using a distribution built on size, row sums and column sums of a rectangular matrix, we can compute a new matrix such that the spectral norm of the differences is bounded and the row sums and column sums are approximately preserved.

The algorithm applies to any matrix, not necessarily a square matrix.

- $\mathbf{A} \in \mathbb{R}^{d_1 \times d_2}$ has no column/row with all zeros
- $\epsilon, p \in (0,1)$, $s = d_1 + d_2$, $\vec{r} = \mathbf{A}\mathbb{1}$(a vector of sums by rows), $\vec{c} = \mathbf{A}^\top \mathbb{1}$(a vector of sums by columns)
- $\mathcal{D}$ is a distribution of $\mathbb{R}^{d_1 \times d_2}$ matrices such that we pick a matrix $\mathbf{X}$ from $\mathcal{D}$:

$$\mathbf{X}_{ij} = \left(\frac{\mathbf{A}_{ij}}{p_{ij}}\right) \text{ with probability } p_{ij} = \frac{\mathbf{A}_{ij}}{s}\left[\frac{1}{\vec{r}_i} + \frac{1}{\vec{c}_j}\right] \text{ for all } \mathbf{A}_{ij} \neq 0$$

- We sample for k times($k \geq 128\frac{s}{\epsilon^2}\log\frac{s}{p}$) independently to get $\mathbf{A}_1, ...\mathbf{A}_k$ and we output the average $\tilde{\mathbf{A}} = \frac{1}{k}\sum_{i\in[k]} \mathbf{A}_i$

Some intuition: First, we can easily verify that the expectation of row sums and column sums of any $\mathbf{X}$ picked from the distribution are equal to the original graph's row sums and column sums, respectively. We can also observe that the entry value $\mathbf{X}_{ij}$ is in fact **not** proportional to the single value $\mathbf{A}_{ij}$ (get canceled out), but is related to the row sum and column sum for the row i and column j. Moreover, if $\mathbf{A}_{ij}$ has a relatively large value, or relatively large portion in its row and column sums, we have a larger corresponding probability p_{ij} and are more likely to pick it.

Concentration bounds: There are three concentration bounds are satisfied by this $\tilde{\mathbf{A}}$:

- $\Pr[\|\mathbf{R}^{-1/2}(\tilde{\mathbf{A}} - \mathbf{A})\mathbf{C}^{-1/2}\|_2 \geq \epsilon] \leq p$ (spectral norm is roughly preserved)
- $\Pr[\|\mathbf{R}^{-1}(\tilde{\mathbf{A}} - \mathbf{A})\mathbb{1}\|_\infty \geq \epsilon] \leq p$ (row sums are roughly preserved)
- $\Pr[\|\mathbf{R}^{-1}(\tilde{\mathbf{A}} - \mathbf{A})^\top\mathbb{1}\|_\infty \geq \epsilon] \leq p$ (column sums are roughly preserved)

3.3 Sparsification and Decomposition Algorithm

For the almost-linear time directed Laplacian system solver we will be discussing, only Eulerian graph sparisification is required. However, we have included the more general case of non-Eulerian graphs because we believe it is of independent interest and may be useful in other settings.

We can do a subgraph sparsification for a directed Laplacian $\mathcal{L}$ in time polynomial to the number of nonzero entries in $\mathcal{L}$: For the corresponding graph $\mathbf{A}$, first sample a graph according to the above sampling algorithm; then patch the sparsified graph to make sure that the row sums and column sums of the sparsified graph $\tilde{\mathbf{A}}$ are the same as original graph $\mathbf{A}$. Then output $\tilde{\mathbf{A}}$'s Laplacian $\tilde{\mathcal{L}}$. More formally, the Subgraph Sparsification algorithm is shown in 1.

The subgraph sparsification algorithm runs in time $O(nnz(\mathcal{L}) + nnz(\tilde{\mathcal{L}}))$ ([CKP+16a], Lemma 3.13). We provide a proof sketch for the correctness of this sparsification in Appendix C Lemma 17.

Algorithm 1 SpasifySubgraph

procedure SPARSIFYSUBGRAPH($\mathcal{L}, p, \epsilon$)
Input: A directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ and parameters $p, \epsilon \in (0, 1)$
Consider all the not-all-zero rows only
Compute $\vec{r} = \mathbf{A}^\top \mathbb{1}$, $\vec{c} = \mathbf{A}\mathbb{1}$, and $s = nnz(\vec{r}) + nnz(\vec{c})$ is the total number of rows and columns that are non-zero
Do the sampling procedure given by the section above using $\mathbf{A}^\top$ as original matrix for k times($k = 128 \cdot n\epsilon^2 \log(n/p)$) and compute the mean $\hat{\mathbf{A}}$
Compute a temporary matrix $(1 + \frac{\epsilon}{4})^{-1}\hat{\mathbf{A}}$ to make the rows and columns sums smaller than original $\mathbf{A}^\top$
Compute a nonnegative patching matrix $\mathbf{E}$: greedily set $O(n)$ entries to make the rows and columns sums of $(1 + \frac{\epsilon}{4})^{-1}\hat{\mathbf{A}} + \mathbf{E}$ equal to the sums of $\mathbf{A}^\top$
$\hat{\mathbf{A}} := (1 + \frac{\epsilon}{4})^{-1}\hat{\mathbf{A}} + \mathbf{E}$
Add the all-zero rows back
Return $\tilde{\mathcal{L}} = \mathbf{diag}(\tilde{\mathbf{A}}^\top \mathbb{1}) - \tilde{\mathbf{A}}$
Output: a sparsified Laplacian $\tilde{\mathcal{L}}$
end procedure

Finally, the paper showed that we can sparsify any Eulerian Laplacian by decomposing the graphs and run the subgraph sparsification algorithm on the subgraphs. The decomposition algorithm `FindDecomposition` that runs in $\tilde{O}(nnz(\mathcal{L}))$ is given in some previous works such as [ShT14].

Therefore, the general procedire of sparsifying an Eulerian Laplacian is shown in 2.

Algorithm 2 SpasifyEulerian

procedure SPARSIFYEULERIAN($\mathcal{L}, p, \epsilon$)
Input: A directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ and parameters $p, \epsilon \in (0, 1)$
Decompose the directed Laplacian $\mathcal{L}$ in to $\mathcal{L}^{(1)}, \mathcal{L}^{(2)} ... \mathcal{L}^{(k)} \in \mathbb{R}^{n \times n}$ where $\mathcal{L} = \mathcal{L}^{(1)} + \mathcal{L}^{(2)} + ... + \mathcal{L}^{(k)}$.(using the FindDecomposition algorithm)
Run subgraph sparsification algorithm on each $\mathcal{L}^{(i)}, \forall i \in [k]$, we get the subgraph Laplacian sparsifier $\tilde{\mathcal{L}^{(i)}}$.
Return the sparsifier for $\mathcal{L}$: $\tilde{\mathcal{L}} = \sum_{i=1}^{k} \tilde{\mathcal{L}}^{(i)}$
end procedure

Theorem 10. *For Eulerian Laplacian $\mathcal{L} \in \mathbf{R}^{n \times n}$ and $\epsilon, p \in (0, 1)$ with probability at least $1 - p$ the routine* `SparsifyEulerian`*$(\mathcal{L}, p, \epsilon)$ computes in $\tilde{O}(nnz(\mathcal{L}) + n\epsilon^2 \log(1/p))$ time an Eulerian Laplacian $\tilde{\mathcal{L}}$ such that $\tilde{\mathcal{L}}$ is an ϵ-sparsifier of $\mathcal{L}$ and in- and out-degrees of the graphs corresponding to $\mathcal{L}$ and $\tilde{\mathcal{L}}$ are identical.*

We defer a proof sketch of this theorem to Appendix C.

3.4 Sparsification of Squared Eulerian Laplacian

To pave the way for the later discussion on Square-Sparsification chain, we show how to sparsify implicitly represented Eulerian Laplacians like this: given a directed Eulerian Laplacian $\mathcal{L}$ for a

strongly connected graph, the goal is to sparsify $\mathcal{M} = \mathbf{D} - \mathbf{A}^\top \mathbf{D}^{-1} \mathbf{A}^\top$ in nearly linear time of $\mathcal{L}$ without constructing $\mathcal{M}$.

The idea is decomposing $\mathcal{M}$ into directed Laplacians corresponding to each vertex: $\mathcal{L}^{(i)} = \mathbf{diag}(\mathcal{A}_{i,:}) - \frac{1}{\mathbf{D}_{i,i}} \mathcal{A}_{:,i} \mathcal{A}_{i,:}^\top$, where $\mathcal{A}_{i,:}$ is row i of $\mathcal{A}$ and $\mathcal{A}_{:,i}$ is column i of $\mathcal{A}$(in- and out-edges for the vertex) and sparsify each decomposed component using a `SparsifyProduct` algorithm.

The `SparsifyProduct` algorithm is almost the same as the `SparsifySubgraph` algorithm except using a slightly different distribution for sampling.

On input vectors $\vec{x}$, $\vec{y}$ which have the same l_1 norm r, $s = nnz(\vec{x}) + nnz(\vec{y})$, the Laplacian we want to sparsify is $\mathbf{diag}(\vec{y}) - \frac{1}{r}\vec{x}\vec{y}^\top$. The distribution $\mathcal{D}$ is over $\mathbb{R}^{n \times n}$ such that $\mathbf{X} \sim \mathcal{D}$ is:

$$\mathbf{X} = \left(\frac{\vec{x_i} \vec{y_j}}{r \cdot p_{ij}} \right) \vec{1}_i \vec{1}_j, \text{ with probability } p_{ij} = \frac{1}{s} \left[\frac{\vec{x_i}}{r - \vec{x_i}} + \frac{\vec{y_j}}{r - \vec{y_j}} \right] \text{ for } i \neq j \text{ and } \vec{x_i}\vec{y_j} \neq 0$$

This is actually equivalent to the distribution used in `SparsifySubgraph`: take $\mathbf{X} = \mathbf{diag}(\vec{x})$, $\mathbf{Y} = \mathbf{diag}(\vec{y})$, $\mathbf{A}^\top = \frac{1}{r}[\vec{x}\vec{y} - \mathbf{XY}]$. It follows that $\mathbf{D} = \mathbf{Y} - \frac{1}{r}\mathbf{XY}$ and the Laplacian we sparsify is $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$. Then we let $\vec{r} = \mathbf{A}^\top \mathbb{1}$ and $\vec{c} = \mathbf{A}\mathbb{1}$. So for $i \neq j$ with $\mathbf{A}_{ij} = \vec{x_i}\vec{y_j}$ we have: $\frac{\mathbf{A}_{ij}}{s}\left[\frac{1}{\vec{r_i}} + \frac{1}{\vec{c_j}}\right] = \frac{\vec{x_i}\vec{y_j}}{r \cdot s}\left[\frac{1}{x_i - \frac{1}{r}\vec{x_i}\vec{y_j}} + \frac{1}{y_j - \frac{1}{r}\vec{x_i}\vec{y_j}}\right] = \frac{1}{s}\left[\frac{\vec{x_i}}{r - \vec{x_i}} + \frac{\vec{y_j}}{r - \vec{y_j}}\right]$. Therefore the output $\tilde{\mathcal{L}}$ is the output of SpasifySubgraph from $\mathcal{L}$.

We then use this `SparsifyProduct` algorithm as a subroutine for the following SparsifySquare to obtain ϵ-sparsifier for implicitly represented Eulerian Laplacian $\mathcal{M} = \mathbf{D} - \mathcal{A}\mathbf{D}^{-1}\mathcal{A}$.

Algorithm 3 SpasifySquare

procedure SPARSIFYSQUARE($\mathcal{A}, p, \epsilon$)
Input: $\mathcal{A} \in \mathbb{R}^{n \times n}_{\geq 0}$ with $\mathcal{A}\mathbb{1} = \mathcal{A}^\top \mathbb{1}$ that implicitly represents a directed Laplacian $\mathcal{M} = \mathbf{D} - \mathcal{A}\mathbf{D}^{-1}\mathcal{A}$ where $\mathbf{D} = \mathbf{diag}(\mathcal{A}\mathbb{1})$ and parameters $p, \epsilon \in (0, 1)$
  for $i = 1 \cdots n$ **do**
    Let $\mathcal{A}_{i,:}$ be row i of $\mathcal{A}$ and $\mathcal{A}_{:,i}$ be column i of $\mathcal{A}$.
    $\tilde{\mathcal{L}}^{(i)} \leftarrow$ SparsifyProduct($\mathcal{A}_{i,:}, \mathcal{A}_{:,i}, p/2n, \epsilon/6$)
  end for
  Let $\hat{\mathcal{M}} = \sum_{i=1}^{n} \tilde{\mathcal{L}}^{(i)}$
  $\tilde{\mathcal{M}} \leftarrow$ SparsifyEulerian($\hat{\mathcal{M}}, p/2, \epsilon/3$)
  Return $\mathbf{D} - \tilde{\mathcal{M}}$
end procedure

4 Solving Directed Laplacian Systems

4.1 Overview

In this section we present an overview of the approach for using the sparsifier construction to solve the polynomially well-conditioned Eulerian Laplacian Systems in almost-linear-time.

The almost-linear-time solver can be split into two key components:

1. **preconditioned Richardson iteration** which allows us to build a pseudoinverse and to improve the quality of such approximate pseudoinverse;

2. **square-sparsification chain** which breaks down a pseudoinverse to a product expansion of a series of matrices and decrease the computational cost due to the sparse nature.

The square-sparsification chain allows us to find the pseudoinverse of a matrix by performing some matrix-vector multiplications and solving a sparse linear system. However, the problem is the approximation error from the sparsification accumulates quickly. That's where the preconditioned Richardson iterations come into play. Combining both approaches, we can formally bound the accumulated error and ensure Richardson will quickly produce a high quality approximate pseudoinverse.

4.2 Preconditioned Richardson Iteration

The solver is based on a key iterative method called **preconditioned Richardson iteration**.We interpret it here in the form of "Gradient Descent" algorithm introduced in class.

Given the following unconstrained optimization problem:

$$\underset{x}{\text{minimize}} \quad F(x) = \frac{1}{2}||\mathbf{A}x - b||_2^2$$

Since l_2 norm is convex and differentiable, we can use gradient descent to find the optimality:

$$x^{(k+1)} = x^k - \eta\nabla F(x) = x^k - \eta(\mathbf{A}^\top\mathbf{A}x^k - \mathbf{A}^\top b) = x^k + \eta(\tilde{b} - \tilde{\mathbf{A}}x^k)$$

where η is the step size, $\tilde{\mathbf{A}} = \mathbf{A}^\top\mathbf{A}$ and $\tilde{b} = \mathbf{A}^\top b$. We can then use an iterative method to solve $\tilde{\mathbf{A}}x = \tilde{b}$: start with $x^{(0)} = \mathbb{0}$, and repeatedly move toward the direction of the residual (i.e. $\tilde{b} - \tilde{\mathbf{A}}x$) for some step size η, until convergence. This is exactly the Richardson Iteration method.

As discussed in class, the quality of such iterations is highly dependent on the condition number of the matrix and thus can be unideal for our purpose in some cases. To fix this problem, we introduce a pre-conditioner $\mathbf{Z}$, to update $x^{(k+1)} = x^k + \eta\mathbf{Z}(\tilde{b} - \tilde{\mathbf{A}}x^k)$. This gives us the **preconditioned Richardson iteration**. Note that if $\mathbf{Z}$ is exactly the pseudo-inverse of $\tilde{\mathbf{A}}$, and $\eta = 1$, then we can obtain the exact solution in one iteration.

The preconditioned Richardson implements a linear operator: $x^k = \mathbf{Z}_k\vec{b}$, for some $\mathbf{Z}_k$ that depends on $\mathbf{Z}, \tilde{\mathbf{A}}, \eta, k$ only. Below is the algorithm that finds $\mathbf{Z}_k$.

Algorithm 4 Preconditioned Richardson Iteration

procedure PRECONDITIONEDRICHARDSON($\mathbf{M}, \mathbf{Z}, \eta, N$)
Input: $n \times n$ matrix $\mathbf{M}$, preconditioning linear operator $\mathbf{Z}$, step size η, iteration count N
Output: (Implicit) matrix $\mathbf{Z}_N$ that corresponds to the routine that applies a linear operator to a vector.
    for $k = 1 \cdots N$ **do**
        $\mathbf{Z}_N = \mathbf{Z}_N + (\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})^{k-1}$
    end for
    return $\eta \cdot \mathbf{Z}_N \cdot \mathbf{Z}$
end procedure

Furthermore, the preconditioned Richardson gives us the following bound on how close our result after iteration k and the optimal solution is:

Lemma 11. *(Lemma 4.2 from [CKP+16a]) Let $\vec{b} \in \mathbb{R}^n$ and $\mathbf{M}, \mathbf{Z}, \mathbf{U} \in \mathbb{R}^{n\times n}$ such that $\mathbf{U}$ is symmetric positive semidefinite, $\ker(\mathbf{U}) \subseteq \ker(\mathbf{M}) = \ker(\mathbf{M}^\top) = \ker(\mathbf{Z}) = \ker(\mathbf{Z}^\top)$, and $b \in im(\mathbf{M})$. Then $N \geq 0$ iterations of preconditioned Richardson with step size $\eta > 0$, results in a vector $\vec{x}_N = \texttt{PRECONRICHARDSON}(\boldsymbol{M}, \boldsymbol{Z}, \vec{b}, \eta, N)$ such that*

$$\left|\left|\vec{x}_N - \mathbf{M}^{+\vec{b}}\right|\right|_{\mathbf{U}} \leq ||\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M}||_{\mathbf{U}\to\mathbf{U}}^N \left|\left|\mathbf{M}^{+\vec{b}}\right|\right|_{\mathbf{U}}$$

Furthermore, preconditioned Richardson implements a linear operator, in the sense that $\vec{x}_N = \mathbf{Z}_N\vec{b}$, for some matrix $\mathbf{Z}_N$ only depending on $\mathbf{Z}, \mathbf{M}, \eta,$ and , N.

The above lemma shows that if $\eta\mathbf{Z}\tilde{\mathbf{A}}$ is sufficiently close to the identity, then preconditioned Richardson converges quickly when solving a linear system. **This highlights a precise way to quantify how good a matrix is as a preconditioner for the Richardson iteration.** Indeed, we will define approximate pseudoinverse in the exact same way:

Definition 12. *Matrix $\mathbf{Z}$ is an ϵ-approximate pseudoinverse of matrix $\mathbf{M}$ with respect to a symetric PSD matrix $\mathbf{U}$, if $\ker(\mathbf{U}) \subseteq \ker(\mathbf{M}) = \ker(\mathbf{M}^\top) = \ker(\mathbf{Z}) = \ker(\mathbf{Z}^\top)$, and*

$$||\mathbf{I}_{im(\mathbf{M})} - \mathbf{Z}\mathbf{M}||_{\mathbf{U}} \leq \epsilon$$

.

The purpose of introducing preconditioned Richardson iteration is because it can improve our pseudoinverse. Roughly speaking, this will guarantee the U-norm of the error to decrease by a factor of ϵ in each iteration:

Lemma 13 (Lemma 4.4 from [CKP+16a]). *If* $\mathbf{Z}$ *is an* ϵ*-approximate pseudoinverse of* $\mathbf{M}$ *with respect to* $\mathbf{U}$*, for* $\epsilon \in (0,1)$*, and* $N \geq 0$*, then* PRECONRICHARDSON $(\mathbf{M}, \mathbf{Z}, 1, N)$ *computes* $\mathbf{Z}_N$*, which only depends on* $\mathbf{Z}, \mathbf{M}, N$*, such that* $\mathbf{Z}_N$ *is an* ϵ*-approximate pseudoinverse of* $\mathbf{M}$ *with respect to* $\mathbf{U}$*.*

Proof. (Partial) We first examine $\vec{x}_{k+1}$ and $\vec{x}_k$ in one preconditioned Richardson iteration. Consider some vector $\vec{b} \in im(\mathbf{M})$ which we are solving $\mathbf{M}\vec{x} = \vec{b}$ for.

Since $\vec{x}_{k+1} = \vec{x}_k + \eta\mathbf{Z}(\vec{b} - \mathbf{M}\vec{x}_k)$, we know that:

$$\begin{aligned}\vec{x}_{k+1} - \mathbf{M}^+\vec{b} &= ((\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})\vec{x}_k + \eta\mathbf{Z}\vec{b}) - \mathbf{M}^+\vec{b} \\ &= (\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})\vec{x}_k + \eta\mathbf{Z}\mathbf{M}\mathbf{M}^+\vec{b} - \mathbf{M}^+\vec{b} \\ &= (\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})(\vec{x}_k - \mathbf{M}^+\vec{b})\end{aligned}$$

Then:

$$||\vec{x}_{k+1} - \mathbf{M}^+\vec{b}||_{\mathbf{U}} = ||(\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})(\vec{x}_k - \mathbf{M}^+\vec{b})||_{\mathbf{U}} \leq ||(\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})||_{\mathbf{U}}||(\vec{x}_k - \mathbf{M}^+\vec{b})||_{\mathbf{U}}$$

By induction:

$$||\vec{x}_N - \mathbf{M}^+\vec{b}||_{\mathbf{U}} \leq ||(\mathbf{I}_{im(\mathbf{M})} - \eta\mathbf{Z}\mathbf{M})||_{\mathbf{U}}^N||\mathbf{M}^+\vec{b}||_{\mathbf{U}} \leq \epsilon^N||\mathbf{M}^+\vec{b}||_{\mathbf{U}}$$

where the last inequality holds because $\mathbf{Z}$ is an ϵ-approximate pseudoinverse, and we plug the definition in.

Since $\vec{x}_N = \mathbf{Z}_N\vec{b}$, we have:

$$||(\mathbf{I}_{im(\mathbf{M})} - \mathbf{Z}_N\mathbf{M})\mathbf{M}^+\vec{b}||_{\mathbf{U}} = ||\vec{x}_N - \mathbf{M}^+\vec{b}||_{\mathbf{U}} \leq \epsilon^N||\mathbf{M}^+\vec{b}||_{\mathbf{U}}$$

which holds for $\vec{b} \in im(\mathbf{M})$. Equivalently, this means that

$$||\mathbf{I}_{im(\mathbf{M})} - \mathbf{Z}_N\mathbf{M}||_{\mathbf{U}} \leq \epsilon^N$$

.

To complete the whole proof, we should also show $ker(\mathbf{U}) \subseteq \ker(\mathbf{Z}_N) = \ker(\mathbf{Z}_N^\top) = \ker(\mathbf{Z})$. This can be done using proof by contradiction (assume $\ker(\mathbf{Z})$ is a strict subset of $\ker(\mathbf{Z}_N)$; assume $\ker(\mathbf{Z})$ is a strict subset of $\ker(\mathbf{Z}_N^\top)$). We will omit here because of space reason.

4.3 Square-sparsification chain: Definition and Construction

In order to solve for $\mathcal{L}x = b$, we decompose $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top = \mathbf{D}^{1/2}(\mathbf{I} - \mathcal{A})\mathbf{D}^{1/2}$, where $\mathcal{A} = \mathbf{D}^{-1/2}\mathbf{A}^\top\mathbf{D}^{-1/2}$ and $||\mathcal{A}||_2 = 1$, and reduces the problem to solving linear systems in $\mathbf{L} = \mathbf{I} - \mathcal{A}$.

Although Peng-Spielman's solver cannot be directly applied to directed graph's case, it suggest a good algorithm for solving a system $\mathcal{L}x = b$: transform the problem to "performing a series of (nice) matrix-vector multiplication (and possibly a bit of something more)". This motivates the **square-sparsification chain**, which is formally defined as below:

Definition 14 (Square Sparsifier Chain). *We call a sequence of matrices* $\mathcal{A}_0$*,* $\mathcal{A}_1$*, ...*$\mathcal{A}_d \in \mathbb{R}^{n\times n}$ *a square-sparsifier chain of length* d *with parameter* $0 < \alpha < 1/2$ *and error* $\epsilon \leq 1/2$ *(or a (*d*,* ϵ*,* α*)-chain for short) if under the definitions* $\mathbf{L}_i = \mathbf{I} - \mathcal{A}_i$ *and* $\mathcal{A}_i^{(\alpha)} = \alpha\mathbf{I} + (1-\alpha)\mathcal{A}_i$ *for all* i *the following hold:*

1. $||\mathcal{A}_i||_2 \leq 1$ *for all* i*,*
2. $\mathbf{I} - \mathcal{A}_i$ *is an* ϵ *approximation of* $\mathbf{I} - (\mathcal{A}_{i-1}^{(\alpha)})^2$ *for all* $i \geq 1$*,*
3. $\ker(\mathbf{L}_i) = \ker(\mathbf{L}_i^\top) = \ker(\mathbf{L}_j) = \ker(\mathbf{L}_j^\top) = \ker(\mathbf{U}_{\mathbf{L}i}) = \ker(\mathbf{U}_{\mathbf{L}j})$ *for all* i*,* j*.*

Lemma 15. *(Correctness of BuildChain procedure) Let* $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$ *be an Eulerian Laplacian that is associated with a strongly connected graph, let* $\alpha, \epsilon, p \in (0,1)$*, and let* $d \geq 1$*. Then in* $\widetilde{O}(nnz(\mathcal{L}) + n\epsilon^{-2}d)$ *time the routine* BUILDCHAIN$(\mathcal{L}, d, \alpha, \epsilon, p)$ *produces* $\mathcal{A}_0, \ldots, \mathcal{A}_d \in \mathbb{R}^{n\times n}$ *that with probability* $1 - p$

Algorithm 5 BuildChain

procedure BUILDCHAIN($\mathcal{L}, d, \alpha, \epsilon, p$)
Input: Eulerian Laplacian $\mathcal{L}$, parameters $\alpha, \epsilon, p \in (0, 1)$.
Output: A series of matrix $(\mathcal{A}_0, \cdots, \mathcal{A}_d)$
  $\mathcal{L}_0 \leftarrow$ SPARSIFYEULERIAN$((\mathcal{L}, p/(d+1), 1/20))$
  $\mathcal{A}_0 = \mathbf{I} - \mathbf{D}^{-1/2}\mathcal{L}_0\mathbf{D}^{-1/2}$.
  for $i = 0, \cdots d-1$ **do**
    $\mathcal{A}_i^{(\alpha)} \leftarrow \alpha\mathbf{I} + (1-\alpha)\mathcal{A}_i$
    $\mathbf{A}_{i+1} \leftarrow$ SPARSIFYSQUARE$(\mathbf{D}^{1/2}\mathcal{A}_i^{(\alpha)}\mathbf{D}^{1/2}, p/(d+1), \epsilon)$
    $\mathcal{A}_{i+1} = \mathbf{D}^{-1/2}\mathbf{A}_{i+1}\mathbf{D}^{-1/2}$.
  end for
  return $\mathcal{A}_0, \cdots, \mathcal{A}_d$
end procedure

1. *$\mathcal{A}_0, \ldots, \mathcal{A}_d$ is a (d, α, ϵ)-chain,*
2. *$nnz(\mathcal{A}_i) = \widetilde{O}(n\epsilon^{-2})$ for all i, and*
3. *$\mathbf{I} - \mathcal{A}_0$ is a (1/20)-approximation of $\mathbf{D}^{-1/2}\mathcal{L}\mathbf{D}^{-1/2}$.*

Proof: We will skip the proof of correctness because of space reason. Please refer to Lemma 4.8 from [CKP+16a].

4.4 Square-sparsification chain: Properties and Error Bound

We will introduce a few properties of the chain we just constructed. We do not intend to prove every single one of them; Instead, we would like to prove the main theorem based on those properties.

- (Constant factor change, Lemma 4.9 from [CKP+16a]) $\kappa(\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}, \mathbf{I} - \mathbf{U}_{\mathcal{A}_{i-1}}) \leq 21, \forall i \in [d], \alpha = 1/4, d \geq 1$.
- (Triangle inequality of approximate pseudoinverse, Lemma 4.10 from [CKP+16a]) If matrix $\mathbf{Z}$ is an ϵ-approximate pseudoinverse of $\mathbf{M}$ with respect to $\mathbf{U}$, and $\tilde{\mathbf{M}}^+$ is an ϵ'-approximate pseudoinverse of $\mathbf{Z}^+$ with respect to $\mathbf{U}$, and has the same left and right kernels as $\mathbf{M}$ and $\mathbf{Z}$, then $\tilde{\mathbf{M}}^+$ produces an $(\epsilon + \epsilon' + \epsilon\epsilon')$-approximate pseudoinverse for $\mathbf{M}$ with respect to $\mathbf{U}$.
- (Pseudoinverse of approximation is good approximate pseudoinverse of original, Lemma 4.11 from [CKP+16a]) Let $\mathbf{M}$ be any matrix with $\mathbf{U}_\mathbf{M}$ positive semidefinite, such that $\ker(\mathbf{M}) = \ker(\mathbf{M}^\top) = \ker(\mathbf{U}_\mathbf{M})$. Suppose that $\tilde{\mathbf{M}}$ is $\mathbf{M}$'s $\epsilon-$approximation, then $\tilde{\mathbf{M}}^+$ is an $2\epsilon-$ approximate pseudoinverse for $\mathbf{M}$ with respect to $\mathbf{U}_\mathbf{M}$
- (Preserved under right multiplication, Lemma 4.12 part 1 from [CKP+16a]) Let $C \in \mathbb{R}^{n\times n}$ such that both $\mathbf{C}$ and $\mathbf{C}^\top$ are invariant on $\ker(\mathbf{M})$, then $\mathbf{Z}$ is an ϵ-approximate pseudoinverse for $\mathbf{CM}$ with respect to $\mathbf{U}$ if and only if $\mathbf{ZC}$ is an ϵ-approximate pseudoinverse for $\mathbf{M}$ with respect to $\mathbf{U}$.
- (Approximately preserved under norm change, Lemma 4.12 part 2 from [CKP+16a]) If $\mathbf{Z}$ is an ϵ-approximate pseudoinverse for $\mathbf{M}$ with respect to $\mathbf{U}$, then for any symmetric PSD matrix $\tilde{\mathbf{U}}$, such that $\ker(\tilde{\mathbf{U}}) = \ker(\mathbf{U})$, $\mathbf{Z}$ is an $(\epsilon \cdot \sqrt{\kappa(\tilde{\mathbf{U}}, \mathbf{U})})$-approximate pseudoinverse of $\mathbf{M}$ with respect to $\tilde{\mathbf{U}}$

Equipped with the above property, we can then prove the lemma below which formally bound the amount of error accumulated:

Lemma 16. *(Lemma 4.13 from [CKP+16a]) Let the sequence $\mathcal{A}_0, \mathcal{A}_1, \cdots, \mathcal{A}_d$ be a (d, ϵ, α)-chain as specified in Definition 14, with $\epsilon \leq 1/2$ and $\alpha = 1/4$. Using the notation from Definition 14, consider the matrix*

$$\bar{\mathbf{Z}}_{i,i+\Delta} = (1-\alpha)^\Delta(\mathbf{I} - \mathcal{A}_{i+\Delta})^+(\mathbf{I} + \mathcal{A}_{i+\Delta-1}^{(\alpha)}) \cdots \mathbf{I} + \mathcal{A}_i^{(\alpha)},$$

for any $i, \Delta \geq 0$. Then $\bar{\mathbf{Z}}_{i,i+\Delta}$ is an $(\exp(5\Delta) \cdot \epsilon)$-approximate pseudoinverse of $IIbf - \mathcal{A}_i$ with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$.

Proof. Let's prove by induction on Δ.

Base Case: $\Delta = 0$, it's trivial since $\bar{\mathbf{Z}}_{i,i+0} = (\mathbf{I} - \mathcal{A}_i^+)$, so it is 0-approximation.

Induction Hypothesis: Assume that the claim is true for $\Delta - 1$, i.e.

$$\bar{\mathbf{Z}}_{i+1,i+\Delta} = (1-\alpha)^{\Delta-1}(\mathbf{I} - \mathcal{A}_{i+\Delta})^+(\mathbf{I} + \mathcal{A}_{i+\Delta-1}^{(\alpha)}) \cdots (\mathbf{I} + \mathcal{A}_{i+1}^{(\alpha)})$$

is an $(\exp(5(\Delta-1))\epsilon)$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_{i+1}$ with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_{i+1}}$.

Inductive Step: By "constant factor change" property, we know that when $\alpha = 1/4$, $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$ and $\mathbf{I} - \mathbf{U}_{\mathcal{A}_{i-1}}$ will not differ too much – actually, the relative condition number is less than 21. Then following "Approximately preserved under norm change" property, $\bar{\mathbf{Z}}_{i+1,i+\Delta}$ is also a $(\sqrt{21}\exp(5(\Delta-1))\epsilon)$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_{i+1}$ with respect to the new norm, $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$.

Then, since $\mathbf{I} - \mathcal{A}_{i+1}$ is an ϵ-approximation of $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2$, using the property "Pseudoinverse of approximation is good approximate pseudoinverse of original", the pseudoinverse of $\mathbf{I} - \mathcal{A}_{i+1}$, $(\mathbf{I} - \mathcal{A}_{i+1})^+$, is a 2ϵ-approximate pseudoinverse of $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2$ with respect to $\mathbf{I} - \mathbf{U}_{(\mathcal{A}_i^{(\alpha)})^2}$.

Now, to make the norm the same, we need to change the norm again and find the relative condition number between old norm $\mathbf{I} - \mathbf{U}_{(\mathcal{A}_i^{(\alpha)})^2}$ and expected new norm $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$. Some linear algebra tricks tell us :

$$\kappa(\mathbf{I} - \mathbf{U}_{(\mathcal{A}_i^{(\alpha)})^2}, \mathbf{I} - \mathbf{U}_{\mathcal{A}_i}) \leq \frac{28}{3} \leq 7$$

so we know that $(\mathbf{I} - \mathcal{A}_{i+1})^+$ is a 7ϵ-approximation pseudoinverse of $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2$ with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$.

Now: $(\mathbf{I} - \mathcal{A}_{i+1})^+$ (Denote as $\mathbf{Z}$) is a 7ϵ-approximation pseudoinverse of $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2$(Denote as $\mathbf{M}$), $\bar{\mathbf{Z}}_{i+1,i+\Delta}$(Denote as $\tilde{\mathbf{M}}^+$) is a $(\sqrt{21}\exp(5(\Delta-1))\epsilon)$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_{i+1}$(Denote as $\mathbf{Z}^+$), both with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$, we can use the "Triangle inequality of approximate pseudoinverse" to conclude that $\bar{\mathbf{Z}}_{i+1,i+\Delta}$ is an ϵ'-approximate pseudoinverse of $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2$ with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$ where

$$\epsilon' = (\sqrt{21}\exp(5(\Delta-1))\epsilon) + 7\epsilon + (\sqrt{21}\exp(5(\Delta-1))\epsilon) \cdot 7\epsilon \leq 50\exp(5(\Delta-1))\epsilon$$

.

Finally, note that $\mathbf{I} - (\mathcal{A}_i^{(\alpha)})^2 = (1-\alpha)(\mathbf{I} + \mathcal{A}_i^{(\alpha)}) \cdot (\mathbf{I} - \mathcal{A}_i)$. Then by "Preserved under right multiplication", we move $(1-\alpha)(\mathbf{I} + \mathcal{A}_i^{(\alpha)})$ to the right hand side of $\bar{\mathbf{Z}}_{i+1,i+\Delta}$ and conclude that

$$\bar{\mathbf{Z}}_{i+1,i+\Delta} \cdot (1-\alpha) \cdot (\mathbf{I} + \mathcal{A}_i^{(\alpha)}) = \bar{\mathbf{Z}}_{i,i+\Delta}$$

is a $50\exp(5(\Delta-1))\epsilon \leq \exp(5\Delta)\epsilon$-approximate pseudoinverse of $\mathbf{I} - \mathcal{A}_i$ with respect to $\mathbf{I} - \mathbf{U}_{\mathcal{A}_i}$, completing the proof. □

4.5 Combining Richardson Iteration with Square-Sparsification Chain

In this subsection, we finally combine the preconditioned Richardson iteration and the square-sparsification chain to obtain an almost-linear time solver for Eulerian Laplacians. As discussed above, blindly apply square-sparsification will accumulate error too fast. Therefore, we need to reduce the error somehow using preconditioned Richardson iteration. To achieve so, we break the d iterations to Δ-size chunks.

Let's expand the product expansion from i to $i + \Delta$:

$$(\mathbf{I} - \mathcal{A}_i)^+ \approx (1-\alpha)^{\Delta} \underbrace{(\mathbf{I} - \mathcal{A}_{i+\Delta}^{(\alpha)})^+}_{\text{Recursive call to find approx. pseudoinverse with } \epsilon_{lo}} \overbrace{(\mathbf{I} + \mathcal{A}_{i+\Delta-1}^{(\alpha)})(\mathbf{I} + \mathcal{A}_{i+\Delta-2}^{(\alpha)})...(\mathbf{I} + \mathcal{A}_i^{(\alpha)})}^{\Delta \text{ items, each has \# of nnz at most } \tilde{O}(n/\epsilon^2)}$$

We utilize the last lemma in the previous section to turn an approximate pseudoinverse of $(\mathbf{I} - \mathcal{A}_{i+\Delta}^{(\alpha)})^+$ into an approximate pseudoinverse for $(\mathbf{I} - \mathcal{A}_i^{(\alpha)})^+$ (with larger error). The error accumulated in this process is then removed via preconditioned Richardson iteration. Our base case would be $(\mathbf{I} - \mathcal{A}_d^{(\alpha)})^+$, which is well-conditioned, so we can approximately apply its pseudoinverse using a standard iterative method.

Below we provide the pseudocode for the "core" of our recursive solver: the *Solve* function and the complete algorithm:

Algorithm 6 Solve

procedure SOLVE($(\mathcal{A}_i \cdots \mathcal{A}_d), \hat{\lambda}, \epsilon$)

Input: Matrices $\mathcal{A}_i \cdots \mathcal{A}_d$ forming a subsequence of a $(d, \hat{\epsilon}, 1/4)$-chain, a lower bound $\hat{\lambda}$ on $\lambda_*(\mathbf{D}^{-1/2}\mathcal{L}\mathbf{D}^{-1/2})$, accuracy ϵ

Output: (Implicit) matrix that is an ϵ-approximate pseudoinverse of $\mathbf{I}-\mathcal{A}_i$ with respect to $\mathbf{I}-\mathbf{U}_{\mathcal{A}_i}$

  if $i = d$ **then**

    $l \leftarrow \min(1/4, 1.125^d \cdot 0.9 \cdot \hat{\lambda})$

    return PRECONRICHARDSON($\mathbf{I} - \mathcal{A}_d, \frac{l}{4}\mathbf{I}_{im(\mathbf{I}-\mathcal{A}_d)}, 1, \frac{8}{l^2}\log 1/\epsilon$)

  end if

  $\Delta \leftarrow \min(\sqrt{d \log d}, d - i)$

  $\tilde{\mathbf{Z}}_i \leftarrow (1-\alpha)^{\Delta} \cdot$ SOLVE($(\mathcal{A}_{i+\Delta} \cdots \mathcal{A}_d), \hat{\lambda}, \exp(-5\Delta)/30) \cdot (\mathbf{I} + \mathcal{A}_{i+\Delta-1}^{(1/4)}) \cdots (\mathbf{I} + \mathcal{A}_i^{(1/4)})$

  PRECONRICHARDSON($\mathbf{I} - \mathcal{A}_i, \tilde{\mathbf{Z}}_i, 1, \log(1/\epsilon)$)

end procedure

Algorithm 7 SolveEulerian

procedure SOLVEEULERIAN($\mathcal{L}, \epsilon, \vec{b}$)

Input: Eulerian Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$, accuracy ϵ, vector $\vec{b} \perp \mathbb{1}$

Output: An approximate solution x to $\mathcal{L}x = b$ in the sense that $||x - \mathcal{L}^+ b||_{\mathbf{U}_\mathcal{L}} \leq \epsilon ||\mathcal{L}^+ b||_{\mathbf{U}_\mathcal{L}}$.

  Computer estimate $\hat{\lambda}$ of the second eigenvalue of $\mathbf{U}_{\mathbf{D}^{-1/2}\mathcal{L}\mathbf{D}^{-1/2}}$

  $d \leftarrow 6\log(1/\hat{\lambda})$

  $\hat{\epsilon} = \exp(-5\sqrt{d \log d})/30$

  $(\mathcal{A}_0, \cdots, \mathcal{A}_d) \leftarrow$ BUILDCHAIN($\mathcal{L}, d, 1/4, \hat{\epsilon}, 1/n^2$)

  $\mathbf{A}_{\text{tmp}} \leftarrow$ SOLVE($(\mathcal{A}_0, \cdots, \mathcal{A}_d), \hat{\lambda}, \hat{\epsilon}$)

  $\tilde{\mathbf{Z}} \leftarrow$ PRECONRICHARDSON($\mathbf{D}^{-1/2}\mathcal{L}\mathbf{D}^{-1/2}, \mathbf{A}_{\text{tmp}}1, 10\log(1/\epsilon)$)

  return $\mathbf{D}^{-1/2}\tilde{\mathbf{Z}}\mathbf{D}^{-1/2}\vec{b}$

end procedure

Runtime: Following the Lemma 15, it first takes

$$\tilde{O}(m + nd \cdot e^{O(\sqrt{d \log d})})$$

to build the square-sparsification chain of length $d = O(\log \kappa)$.

Then we examine the recursion part. We know that in each iteration, we first use the square sparsification chain to obtain the approximate pseudoinverse (with high error accumulated) in time:

$$\mathcal{T}_{i,\epsilon_{hi}} = \mathcal{T}_{i+\Delta,\epsilon_{lo}} + \tilde{O}(\Delta n \epsilon_{\text{spar}}^{-2})$$

Since it comprises of finding low-error approximate pseudoinverse for $i + \Delta$, plus Δ sparse matrix vector multiplication which has at most $n\epsilon_{\text{spar}}^{-2}$ non-zero entries each.

The second part is to run preconditioned Richardson iteration to lower the error. We know that preconditioned Richardson will guarantee the U-norm of the error to decrease by a factor of ϵ in each iteration. To decrease from ϵ_{high} to ϵ_{lo}, we will then need $O(\frac{\log \epsilon_{lo}}{\log \epsilon_{hi}})$ iterations. Since we need to apply this error lowering technique once in each iteration, it takes time:

$$\mathcal{T}_{i,\epsilon_{lo}} = O(\frac{\log \epsilon_{lo}}{\log \epsilon_{hi}})\mathcal{T}_{i,\epsilon_{hi}} = O(\frac{\log \epsilon_{lo}}{\log \epsilon_{hi}})(\mathcal{T}_{i+\Delta,\epsilon_{lo}} + \tilde{O}(\Delta n \epsilon_{\text{spar}}^{-2}))$$

Setting $\epsilon_{lo} = 1/10, \epsilon_{\text{spar}} + \epsilon_{lo} = 2^{-\Omega(\Delta)}, \epsilon_{\text{spar}} = \epsilon_{lo} = 2^{-\Theta(\Delta)}$, we can simplify the above equation to:

$$\mathcal{T}_{i,\epsilon_{lo}} = O(\Delta)\mathcal{T}_{i+\Delta,\epsilon_{lo}} + \tilde{O}(n2^{\Theta(\Delta)})$$

.

Note that the time used to find base case $\mathbf{I} - \mathcal{A}_d^{(\alpha)}$ using a standard iterative method is less than $\tilde{O}(n2^{\Theta(\Delta)})$ which can be absorbed into the additive term, so it does not affect the time bound much.

After getting the recurrence equation, we need to get the depth of the recursion tree. We can think of the whole algorithm as producing a recursion tree with $O(\Delta)^{\lceil d/\Delta \rceil}$ nodes in total, and at each we add $\tilde{O}(n2^{\Theta(\Delta)})$ (from the recurrence relations), and set $\Delta = \sqrt{d \log d} = \sqrt{\log \kappa \log\log \kappa}$, we get the final bound:

$$\begin{aligned}
\tilde{O}(m + nd \cdot e^{O(\sqrt{d \log d})}) + \mathcal{T}_{0,\epsilon_{lo}} &= \tilde{O}(m + nd \cdot e^{O(\sqrt{d \log d})}) + O(\Delta)^{\lceil d/\Delta \rceil} \tilde{O}(n2^{\Theta(\Delta)}) \\
&= \tilde{O}(m + nd \cdot e^{O(\sqrt{d \log d})}) + nO(\Delta)^{O(d/\Delta)} 2^{O(\Delta)} \\
&= \tilde{O}(m + nd \cdot e^{O(\sqrt{d \log d})}) + n2^{O(\Delta + \frac{d \log \Delta}{\Delta})} \\
&= \tilde{O}(m) + n2^{O(\sqrt{\log \kappa \log\log \kappa})}
\end{aligned}$$

which is exactly the almost-linear-time bound we are looking for.

5 Open Questions and Conclusions

We conclude this report with analysis of the bottlenecks of the algorithms and tentatively propose alternative methods as open questions for future research.

Bottlenecks:

- From lemma 15, we know that it takes $\widetilde{O}(\text{nnz}(\mathcal{L}) + n\epsilon^{-2}d)$ time to `BUILDCHAIN`$(\mathcal{L}, d, \alpha, \epsilon, p)$ which produces $\mathcal{A}_0, \ldots, \mathcal{A}_d \in \mathbb{R}^{n \times n}$ that with probability $1 - p$. So the running time here is dependent on the number of nonzero entries in the graph sparsifier $\tilde{\mathcal{L}}$. But according to Definition 9, $nnz(\tilde{\mathcal{L}})$ is upper bounded by $\tilde{\mathcal{O}}(n\epsilon^{-2})$. That means the sparsity of the the Eulerian sparsifier is the bottleneck of the running time.

- For the reduction part, while the technique the author uses is sufficient to improve the running time bounds, there are discussions that better reduction can be done. More specifically, the second reduction we did not go into detail (as it is not the central of the paper) removes the $\exp(\log \kappa)$ dependencies in the algorithms and replaces with terms in n, which performs $O(\log n\kappa)$ calls to approximate Eulerian Laplacian solver and involves $O(m \log(n\kappa))$ additional work. However, the author believes that better reduction can be done to lower the overhead for reducing condition number to be as low as $O(1)$, since the different scales would only have minimal overlap, and processing could occur in parallel. More discussion is available in Appendix F of [CKP+16a].

Open Questions:

- The key success of [CKP+16a] is because instead of figuring out how to properly neglect the directed structure like previous papers do, this new solver tries to **intrinsically work with asymmetric (directed) objects, and build up the whole framework to properly capture those characteristics.** Then it is tempting to ask: are there any other properties of asymmetric objects that we can utilize to come up with a better solver?

- The cut sparisification, though very intuitive, does not work well for directed graph case. Do we have a 'unified' definition of sparsification that is natural and intuitive for both undirected and directed graph?

- A method of sampling edges is sampling by effective resistance by [SS11], which have many fewer edges than the sparsifier in [ST08]. Can we modify this procedure to do sampling on directed Eulerian Laplacians? (there may exist difficulties since the spectral sparsifiers in [SS11] are similar to cut sparsifiers.)

- To speed up sparsification, we normally use decomposition and local clusterings. In the local clustering problem, one is given a vertex and cluster size as input, and one tries to find a cluster of low conductance near that vertex of size at most the target size, in time proportional to the target cluster size. Besides decomposition which has already been used, since the local clustering coefficient for vertices in directed networks is well-defined, is it possible to apply local clustering to directed Eulerian graphs?

- Besides the parallel solver by [PS14] this paper mostly resembles in method, there exist many other linear system solvers(for both symmetric and asymmetric), is it possible to construct a different framework inspired by other methods?
- Systematically developing strong versions of "condition number reducing reductions" is an important and relevant direction. Can we have better ways to do so? E.g: As highly ill-conditioned systems often arise under floating point arithmetic (due to the exponent), can we develop reductions that are robust to floating point arithmetics?

References

[BK96] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}$(n2) time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 47–55, New York, NY, USA, 1996. ACM.

[BSST13] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, August 2013.

[CKP+16a] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. *CoRR*, abs/1611.00755, 2016.

[CKP+16b] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. *CoRR*, abs/1608.03270, 2016.

[Le 14] F. Le Gall. Powers of Tensors and Fast Matrix Multiplication. *ArXiv e-prints*, January 2014.

[PS14] Richard Peng and Daniel A. Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 333–342, New York, NY, USA, 2014. ACM.

[ShT14] Daniel A. Spielman and Shang hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems, 2014.

[SS11] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, December 2011.

[ST08] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.

A Notation

Matrices and Vectors: We use bold to denote matrices and $\mathbf{I}$ to denote identity matrix. We use the vector notation when we want to highlight that a certain variable is a vector. Furthermore, we use $\vec{\mathbb{0}}, \vec{\mathbb{1}}$ to represent all zeros and ones vectors, respectively. $\mathbf{I}_n = \mathbf{I} \in \mathbb{R}^{n \times n}$. For a matrix $\mathbf{A}$, we use $nnz(\mathbf{A})$ to denote the number of non-zero entries in it. We use $im(\mathbf{A})$ and $ker(\mathbf{A})$ to denote its image space and kernel respectively. We let $\mathbf{A}^+$ denote the (Moore-Penrose) pseudoinverse of $\mathbf{A}$. For a symmteirc positive semidefinite matrix $\mathbf{B}$, let $\mathbf{B}^{1/2}$ denote the square root of $\mathbf{B}$. We use $\kappa(\mathbf{A}) = ||\mathbf{A}||_2 \cdot ||\mathbf{A}^+||_2$ to represent its condition number and $\kappa(\mathbf{A}, \mathbf{B}) = \kappa(\mathbf{A}^{+/2}\mathbf{B}\mathbf{A}^{+/2})$ for their relative condition number. Note that if $\mathbf{X} \in \mathbb{R}^{n \times n}$ is a nonzero diagonal matrix, then $\kappa(\mathbf{X}) = \frac{\max_{i \in [n]} |\mathbf{X}_{ii}}{\min_{i \in [n]} |\mathbf{X}_{ii}}$. Note that if $\alpha\mathbf{B} \preceq \mathbf{A} \preceq \beta\mathbf{B}$, then $\kappa(\mathbf{A}, \mathbf{B}) \leq \beta/\alpha$. Let $\lambda_*(\mathbf{A})$ denote the smallest non-zero eigenvalue of $\mathbf{A}$.

Positive Semidefinite Ordering: For symmetric matrices $\mathbf{A}$, $\mathbf{B} \in \mathbb{R}^{n \times n}$, we use $\mathbf{A} \preceq \mathbf{B}$ to denote the condition that $x^\top \mathbf{A} x \leq x^\top \mathbf{B} x$, for all x. We define $\succeq$, $\prec$, and $\succ$ analogously. We call a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ positive semidefinite (PSD) if $\mathbf{A} \succeq \mathbf{0}$. For vectors x, we let $||x||_{\mathbf{A}} \stackrel{def}{=} \sqrt{x^\top \mathbf{A} x}$. For asymmetric $\mathbf{A} \in \mathbb{R}^{n \times n}$, we let $\mathbf{U}_{\mathbf{A}} \stackrel{def}{=} \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$ and note that $x^\top \mathbf{A} x = x^\top \mathbf{A}^\top x = x^\top \mathbf{U}_{\mathbf{A}} x$ for all $x \in \mathbb{R}^{n \times n}$.

Operator Norms: For any norm $||.||$ defined on vectors in $\mathbb{R}^n$, we define the *seminorm* it induces on $\mathbb{R}^{n\times n}$ by $||\mathbf{A}|| = \max_{x\neq 0} \frac{||\mathbf{A}x||}{||x||}$ for all $\mathbf{A} \in \mathbb{R}^{n\times n}$. When we wish to make clear that we are considering such a ratio we use $\rightarrow$ symbol; e.g., $||\mathbf{A}||_{\mathbf{H}\rightarrow\mathbf{H}} = \max_{x\neq 0} \frac{||\mathbf{A}x||_{\mathbf{H}}}{||x||_{\mathbf{H}}}$. But we may also simply write $||\mathbf{A}||_{\mathbf{H}} \stackrel{def}{=} ||\mathbf{A}||_{\mathbf{H}\rightarrow\mathbf{H}}$ in this case. For symmetric positive definite $\mathbf{H}$ we have that $||\mathbf{A}||_{\mathbf{H}\rightarrow\mathbf{H}}$ can be equivalently expressed in terms of $\|.\|_2$ as $||\mathbf{A}||_{\mathbf{H}\rightarrow\mathbf{H}} = ||\mathbf{H}^{1/2}\mathbf{A}\mathbf{H}^{-1/2}||_2$. Also note that $||\mathbf{A}||_1$ is the maximum l_1 norm of a column of $\mathbf{A}$ and $||\mathbf{A}||_\infty$ is the maximum l_1 norm of a row of $\mathbf{A}$.

Misc: The definition of $\tilde{O}$ notation in this report is a bit different to what we discussed in class: here we suppress terms that are polylogarithmic in n, the natural condition number of the problem κ, and the desired accuracy ϵ.

B Review of Undirected Graph Sparsification

Graph sparsification is the approximation of an arbitrary graph by a sparse graph. The sparsifier for a graph has fewer edges(degrees) but also has similarities in properties to the original graph. The definitions of "sparse" and "similar" matter a lot: they are given according to what properties we are looking for about the graphs and probably depend on how we use them in the algorithms.

Notions of Similarities

- Cut similarity [BK96]: When considering weighted graphs, if we want to develop fast algorithms for the minimum cut and maximum flow problems, then we will be interested in the sum of the weights of edges that are cut when one divides the vertices of a graph into two pieces. Two weighted graphs on the same vertex set are cut-similar if the sum of the weights of the edges cut is approximately the same in each such division. Given any weighted undirected graph G, Benczur and Karger showed that one could construct a new graph H with at most $O(n \log n/2)$ edges such that the value of every cut in G is within a multiplicative factor of $1 \pm \epsilon$ of its value in H.

- Spectral similarity [BSST13]: Given a weighted graph $G = (V, w)$, the Laplacian quadratic form of G is defined as the function Q_G from $\mathbb{R}^v$ to $\mathbb{R}$: $Q_G(x) = \sum_{(u,v)\in E} w_{(u,v)}(x(u)-x(v))^2$. If S is a set of vertices and x is the characteristic vector of S (1 inside S and 0 outside). We say two graphs $G = (V, w)$ and $G' = (V', w')$ are σ-spectrally similar if $Q_{G'}(x) \leq Q_G(x) \leq \sigma \cdot Q_{G'}(x)$. This definition of spectral similarity is in fact a special case of cut similarity.

  For the problem of solving systems of linear equations in Laplacian matrices, if two graphs are spectrally similar, then through the technique of preconditioning one can use solutions to linear equations in the Laplacian matrix of one graph to solve systems of linear equations in the Laplacian of the other.

- Spectral similarity of matrices: For two symmetric matrices A, B are σ-spectrally similar if $B/\sigma \preceq A \preceq \sigma \cdot B$ where $A \preceq B$ is defined as $x^\top Ax \leq x^\top Bx$ for all $x \in \mathbb{R}^n$. It implies that the two matrices have similar eigenvalues. By Courant-Fisher Theorem that

$$\lambda_i(A) = max_{S:dim(S)=i} min_{x\in S} \frac{x^\top Ax}{x^\top x}$$

  And therefore we can deduce from the above property: $L_{G'}/\sigma \preceq L_G \preceq \sigma \cdot L_{G'}$, i.e. two graphs are σ-spectrally similar if their Laplacian matrices are.

- Distance Similarity: An alternative to cut- and spectral-similarity if we assign a length to every edge in a graph.

Undirected Sparsifier For undirected graph G, its (d, σ)-sparsifier G' is defined to satisfy:

- G' is σ-spectrally similar to G
- The edges of G' consist of reweighted edges of G
- G' has at most $d|V|$ edges

Sampling and Decomposition Random sparse graphs are usually good expanders (see Bollobás9 and Friedman17). Therefore, we can obtain a good spectral sparsifier of the complete graph by random sampling. Spielman and Teng in [ST08] gave the sampling procedure:

- Assign a probability $p_{u,v}$ to each edge $(u,v) \in G$
- select edge (u,v) to be in the graph G' with probability $p_{u,v}$
- if edge (u,v) is chosen to be in the graph, multiply its weight by $1/p_{u,v}$

This sampling method guarantees $E[\mathcal{L}_{G'}] = \mathcal{L}_G$. The most important thing is balancing between a smaller $p_{u,v}$ to generate a sparser graph and a larger $p_{u,v}$ for better approximation.

And [ST08] also gave a theorem that says every graph G has an $\Omega(\log^{-2} n)$-spectral decomposition with a boundary size of at most $|E|/2$ edges, where boundary of a decomposition is the set of edges between different vertex sets in the partition.

C Analysis of Sparsification Algorithm

Here we give a analysis sketch of the SparsifyEulerian algorithm.

After the FindDecomposition algorithm which runs in time $\tilde{O}(nnz(\mathcal{L}))$, we apply SparsifySubgraph to each $\mathcal{L}^{(i)}$.

For a directed Laplacian $\mathcal{L} = \mathbf{D} - \mathbf{A}^\top$, denote its graph symmetrization is $\mathbf{S}_\mathcal{L}$ is $\mathbf{S}_\mathcal{L} = diag(\mathbf{U}_A \cdot \mathbb{1}) - \mathbf{U}_A$. And $\mathbf{diag}(\mathbf{S}_\mathcal{L})$ denotes the diagonal matrix having same diagonal as $\mathbf{S}_\mathcal{L}$.

Lemma 17 (Subgraph Sparsification, [CKP+16a] **Lemma 3.13**). *Given a directed Laplacian $\mathcal{L}$ and undirected Laplacian $\mathbf{U}$ with spectral gap at least α, v nonzero rows. $\mathbf{diag}(\mathbf{S}_\mathcal{L}) \preceq \mathbf{diag}(\mathbf{U})$. For $\delta \leq \alpha\epsilon/2$, SparsifySugraph$(\mathcal{L}, p, \delta)$ can compute a Laplacian $\tilde{\mathcal{L}}$ such that $\|\mathbf{U}^{+/2}(\mathcal{L} - \tilde{\mathcal{L}})\mathbf{U}^{+/2}\|_2 \leq \epsilon$ and $nnz(\tilde{\mathcal{L}}) = O(v\delta^{-2}\log(v/p))$, in time $O(nnz(\mathcal{L}) + v\delta^{-2}\log(v/p))$.*

Proof Sketch. The above lemma follows from the sampling procedure that gives $\|\mathbf{R}^{-1/2}(\tilde{\mathbf{A}} - \mathbf{A})\mathbf{C}^{-1/2}\|_2 \leq \epsilon$ with probability at least $1-p$ and the definition of asymmetric matrix approximation. We can obtain $\|\mathbf{D}_{in}^{-1/2}(\mathcal{L} - \tilde{\mathcal{L}})\mathbf{D}_{out}^{-1/2}\|_2 \leq \epsilon$ where $\mathbf{D}_{in} = \mathbf{diag}(\mathbf{A}^\top\mathbb{1})$ and $\mathbf{D}_{out} = \mathbf{diag}(\mathbf{A}\mathbb{1})$.

So here we can have $\|\mathbf{D}_{in}^{-1/2}(\mathcal{L} - \tilde{\mathcal{L}})\mathbf{D}_{out}^{-1/2}\|_2 = \|\mathbf{D}_{in}^{-1/2}(\mathbf{A} - \tilde{\mathbf{A}})\mathbf{D}_{out}^{-1/2}\|_2 \leq \delta$ with probability at least $1-p$. By the equivalent form of asymmetric matrix approximation ([CKP+16a] Lemma 3.5), we can deduce $\|\mathbf{U}^{+/2}(\mathcal{L} - \tilde{\mathcal{L}})\mathbf{U}^{+/2}\|_2 = \frac{\vec{x}'(\mathcal{L}-\tilde{\mathcal{L}})\vec{y}'}{\sqrt{(\vec{x}')^\top\mathbf{U}\vec{x}'\cdot(\vec{y})^\top\mathbf{U}\vec{y}'}} \leq \frac{1}{\alpha}\frac{\vec{x}'(\mathcal{L}-\tilde{\mathcal{L}})\vec{y}'}{\sqrt{(\vec{x}')^\top\mathbf{D}\vec{x}'\cdot(\vec{y})^\top\mathbf{D}\vec{y}'}} \leq \frac{2}{\alpha}\|\mathbf{A} - \tilde{\mathbf{A}})\mathbf{D}_{out}^{-1/2}\|_2 \leq \frac{2}{\alpha}\delta$ since $\mathbf{D}_{in}, \mathbf{D}_{out} \preceq 2\mathbf{D}$ and using $\vec{x}', \vec{y}'$ that give $\max_{\vec{x},\vec{y}\neq 0}\frac{\vec{x}(\mathcal{L}-\tilde{\mathcal{L}})\vec{y}}{\sqrt{(\vec{x})^\top\mathbf{U}\vec{x}\cdot(\vec{y})^\top\mathbf{U}\vec{y}}}$.

The running time is easy to see form the steps of SparsifySubgraph. □

Theorem 10 hence follows from the above lemma by taking $v = O(n)$, $\alpha = \tilde{O}(1)$ (which is also a requirement for the FindDecomposition routine) and $\delta = \epsilon$.