Laplacian Solvers and Graph Sparsification

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Vijay Keswani
Roll No. : 11907799

under the guidance of

Dr. Rajat Mittal

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

May, 2016

CERTIFICATE

It is certified that the work contained in the thesis titled **Laplacian Solvers and Graph Sparsification**, by **Vijay Keswani**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Rajat Mittal
Department Computer Science and Engineering
Indian Institute of Technology
Kanpur-208016

May, 2016

Abstract

Graph Laplacian system of equations figure in a variety of problems in computer science, including fundamental graph theory, computer vision and electrical circuits. Though many traditional algorithms for solving linear system of equations could be used for this problem, the time complexity of these algorithms is super quadratic in the input size. One hopes to exploit the graph sub-structure associated with the system to possibly achieve a faster approximate solver. Spielman and Teng provided a major result in this direction by exhibiting an algorithm which solves a Laplacian system in time almost-linear in the number of edges of the graph.

In this work, we survey and explore the solver by Spielman-Teng, and analyze the various sub-routines in the algorithm, including Spectral Sparsification. Spectral sparsifiers are subgraphs of the graph whose Laplacian eigenvalues are close to the eigenvalues of the original graph Laplacian. We present the Randomized Sparsification techniques of Spielman-Srivastava and the Deterministic Sparsification algorithm of Batson-Spielman-Srivastava, and analyze the underlying intuition behind the algorithms.

We further survey and analyze the Laplacian solver given by Kelner et al., and discuss how it is similar to the Spielman-Teng Laplacian solver. This solver is based on the Randomized Kaczmarz iterative method for solving linear system of equations. We provide two extensions of this iterative technique, based on dihedral angles between hyperplanes, and analyze their convergence bounds. We also show that these extensions can be used to solve Laplacian systems as well.

Acknowledgments

I wish to express my profound gratitude to my thesis advisor, Dr. Rajat Mittal, for introducing me to this wonderful topic and guiding me throughout the thesis. I will fondly remember our weekly discussions on varied topics in Theoretical Computer Science. I am thankful to him for instilling a sense of mathematical rigor in me, while encouraging me to intuitively understand the topics. Most of all, I am grateful to him for his patience, even when I was a beginner in the field. I am also indebted to Dr. Satyadev Nandakumar for his lectures and discussions on various topics.

I would also like to thank my family and friends for their continuous humour, support and motivation, throughout my academic career.

Contents

Chapter 1

Introduction

1.1 Overview

The field of *Spectral Graph Theory* aims to explore the combinatorial properties of a graph through its spectrum, that is, the set of eigenvalues of either the *Adjacency Matrix* or the *Laplacian matrix.* Several results across the years have established the fact that there exists an innate relationship between the structural properties of a graph and its spectrum. One of the earliest and famous results in this respect is the *Cheeger's Inequality*, which relates the size of the worst cut in the graph to the *spectral gap* of the transition matrix of a simple random walk on that graph.

We are interested in solving the linear system of equations $Lx = b$, where L is the Laplacian matrix of a graph. The *Laplacian system of equations* occur in many natural contexts, including finding voltages and currents in an electrical circuit, calculating maximum flow, etc. The problem of finding solutions to a *Laplacian system* quickly is a problem with far-reaching implications.

The classic *Gaussian Elimination* method of solving a linear system of n equations in n variables gives a solution in time proportional to the time taken to multiply two $n \times n$ matrices, for which the best known algorithm has complexity $\mathcal{O}(n^{2.373})$. This is perhaps too slow for our purpose, since the graph corresponding to the Laplacian system may potentially have billions of nodes. In general, it was believed that the Laplacian system can be solved in linear time.

Spielman and Teng, in their seminal paper on this subject, provided the first *nearly-linear* time algorithm for the Laplacian system, and in the process, put forward a wide range of graph-eigenvalue relations to explore. They introduced the notion of spectral sparsification, which they used as sub-routines in their *Laplacian solvers*. In a graph-theoretic sense, spectral sparsification aims to find subgraphs of the graph whose Laplacian eigenvalues are close to the Laplacian eigenvalues of the original graph. Spectral sparsifiers are useful because if H spectrally approximates G, then we can solve the problem in consideration for H rather than G, since it has fewer edges than G.

Furthermore, in the process of developing these solvers, a better understanding of the underlying random matrices was developed. Many general results in *matrix theory* were also given using the same methods. The most famous result in this regard was the affirmative resolution to *Kadison-Singer conjecture*, proven using the *method of interlacing families*.

In this work, we study, survey and explore solvers for Laplacian systems and randomized and deterministic algorithms for Spectral Sparsification.

1.2 History

Several efforts have been made in the past to provide a complete characterization of the properties of the graph using its spectrum. In particular, the work of Fiedler [Fie73][Fie10] showed the importance of *second eigenvalue* to the connectivity of the graph. Later on, Cheeger [Che70] showed how Laplacian eigenvalues can prove existence of good cuts in the graph. Further relations to *isoperimetric properties* of the graph were established in subsequent research [AM85].

Most general methods of solving linear system of equations rely on Gaussian elimination or related techniques. It was known that the Cholesky decomposition of the Laplacian matrix can lead to a nearly-linear time solver for Laplacian systems corresponding to trees [Vis13]. The result can also be generalized to Laplacians of planar graphs, where the work of Tarjan, Lipton and Gilbert [GT86] [GT79] can be used to solve such Laplacian systems in $\mathcal{O}(n^{1.5})$ time using Cholesky decomposition, where n is the number of vertices in the graph.

The first nearly-linear time solver to solve Laplacian systems was given by Spielman and Teng [ST04] [ST08], in which they also introduce the concept of spectral sparsification. They were inspired by the idea of combinatorial preconditioning, introduced by Vaidya [Vai91] in an unpublished thesis. The solver of Spielman and Teng was later improved upon by Koutis et al. [KMP11], which led to algorithms that ran in $\mathcal{O}(m \log n)$ time, where n is the number of vertices in the graph and m is the number of edges. Kolla et al. [KMST10] subsequently showed the existence of an algorithm that can run in $\mathcal{O}(m \log^{1/2} n)$ time.

The notion of graph sparsification was earlier known with respect to cuts, where optimal bounds and constructions were given by Benczur and Karger [BK96]. Spectral sparsification can be considered as a much more general notion of sparsification, and is an important subroutine in the Laplacian solver of Spielman-Teng. The first randomized algorithm for spectral sparsification was also provided by Spielman and Teng [ST11]. This was followed by another randomized spectral sparsification algorithm by Spielman and Srivastava [SS08], a result we present and discuss in this report. Later, Batson et al. [BSS09] provided a deterministic spectral sparsification algorithm, using the concepts of barrier functions and interlacing polynomial families.

Kelner et al. proposed a new approach to solving Laplacian system in [KOSZ13]. In general, solving Laplacian system corresponds to finding the voltages on the nodes in the electric circuit corresponding to the graph. Kelner et al., however suggested a solver which focuses on finding the electrical current in the circuit. The resulting solver is much simpler compared to the Laplacian solver of Spielman-Teng. Their result was extended by Lee and Sidford [LS13].

Both the solvers cited above, in their basic form, require the existence and fast construction of low-stretch spanning trees. The corresponding linear-time algorithm for finding such trees was given by Elkin et al. [EEST08].

The notion of spectral sparsification for random matrices is the following : For every set of matrices whose sum is identity, does there exist a weighted subset of small size, whose sum has eigenvalues in the range $[(1-\epsilon),(1+\epsilon)]$. The results in spectral sparsification therefore, can be extended to many different random matrix related scenarios. In a series of papers, Marcus, Spielman and Srivastava came up with a method of interlacing families of polynomials, using which they gave a positive resolution of the Kadison-Singer conjecture [MSS15], and also proved the existence of bipartite Ramanujan graphs of all degrees [MSS13]. The techniques used here intersect with those used in Deterministic Spectral Sparsification algorithm by Batson et al. [BSS09], particularly the use of *Associated Laguerre polynomials* and Barrier functions.

Very recently, the work of Kyng and Sachdeva [KS16] has provided a randomized algorithm to construct sparse Cholesky decomposition of general graphs and a corresponding unique Laplacian solver.

1.3 Outline

The primary purpose of this report is to survey the recent research in the field of Laplacian solvers and Spectral Sparsification, and study possible extensions of the underlying ideas. For this, we first look at the results in spectral sparsification and graph sparsification.

We describe the general notion of spectral sparsification, and in Chapter 2 and Chapter 3, we look at the optimal results in randomized and deterministic spectral sparsfication. In both these chapters, we refrain from starting with the graph perspective. The topic of spectral sparsification is much more general, with graph sparsification being one of its implications. In this regard, we first present the general results for random matrices and then show how they imply graph sparsification.

The randomized spectral sparsification algorithm of Spielman and Srivastava is presented in Chapter 2. The basic idea here is to sample edges with probability proportional to the *effective resistance* of the edges, and construct a weighted subgraph using the sampled edges. Using *matrix concentration inequalities*, it can be shown that this weighted subgraph is a good sparsifier of the original graph.

Chapter 3 expands upon the deterministic spectral sparsification algorithm of Batson, Spielman and Srivastava. This algorithm makes use of *barrier functions* to keep the eigenvalues of the system bounded. The application of barrier functions in this setting is a unique and novel idea, though the intuition for the same is not provided in the original papers. We use the analysis of the *Kadison-Singer problem* by Marcus, Spielman and Srivastava, to motivate the application of barrier functions to this setting, and correspondingly provide a complete analysis of the algorithm.

In Chapter 4, we show how one can use spectral sparsification to construct a nearly-linear time Laplacian solver. In this chapter, we provide the complete algorithm of Spielman-Teng and Koutis-Miller-Peng, for solving a Laplacian system and analyze the time complexity and error bounds of the algorithm.

Chapter 5 deals with another Laplacian solver, given by Kelner et al., based on an iterative method called the Randomized Kaczmarz method. This solver is significantly simpler to state and analyze compared to the spectral sparsification based Laplacian solver, but in essence, is connected to the earlier solver. We again provide the complete algorithm and its analysis in this chapter. We also look at certain disadvantages of the Randomized Kaczmarz algorithm and state extensions that possibly can tackle such problems. In the end, we prove that these extensions can also be used to solve Laplacian systems.

1.4 Preliminaries

1.4.1 Positive Semi-Definite Matrix

A symmetric $n \times n$ real matrix M is called a positive semi-definite (PSD) matrix if all the eigenvalues of M are greater than or equal to 0. An alternate characterization of the positive semi-definiteness of a matrix can be given by the quadratic form of the matrix, that is, M is PSD iff for any $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{x} \neq 0$,

$$\mathbf{x}^T M \mathbf{x} \geq 0$$

We will denote PSD matrices by the following notation $M \succeq 0$.

Furthermore, the notation $A \succeq B$ will denote that $A - B \succeq 0$, or $A - B$ is a PSD matrix.

1.4.2 Spectral Decomposition

Spectral decomposition of a real symmetric $n \times n$ matrix is a method of expressing the matrix in terms of its eigenvalues and eigenvectors.

Theorem 1.4.1 (Spectral Decomposition Theorem). Let A be a real symmetric $n \times n$ matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$. Then

$$A = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

Since B has the same action as A for all $\mathbf{u}_i$, $B = A$.

1.4.3 Pseudo-Inverse

For an $n \times n$ matrix A, B is called the *inverse* of A if $AB = \mathbf{I}$. A square matrix is not *invertible* if and only if the determinant A is zero. For a real symmetric matrix, this happens if atleast one eigenvalue is zero.

For such matrices, we define the *pseudo-inverse*, which is a generalization of the concept of inverse.

Say we have a real symmetric $n \times n$ matrix A, with spectral decomposition

$$A = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

If A is invertible, then we can write A^{-1} as

$$A^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T$$

If A is not invertible, then atleast one of λ_i's is zero. For the pseudo-inverse of such a matrix, we set coefficients λ_i''s as

$$\lambda_i' = \begin{cases} \frac{1}{\lambda_i}, & \text{if } \lambda_i \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

Then, the pseudo-inverse of A, denoted by A^+, is

$$A^+ = \sum_{i=1}^{n} \lambda_i' \mathbf{u}_i \mathbf{u}_i^T$$

The concept of pseudo-inverse can be generalised to other kinds of matrices using Singular Value Decomposition, and we refer the reader to the the survey [BH12] for more information on the topic.

1.4.4 Interlacing polynomials

Definition 1.4.1 (Interlacing)**.** Let f be a real-rooted polynomial of degree n with roots $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$ and g be another real-rooted polynomial of degree n or $n-1$ with roots $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$.[1] Then we say g interlaces f if

$$\beta_1 \leq \alpha_1 \leq \beta_2 \leq \alpha_2 \leq \cdots \leq \beta_n \leq \alpha_n.$$

The above defined notion of interlacing can be extended to a family of polynomials where all the polynomials of the family interlace a common polynomial.

Definition 1.4.2 (Common Interlacing)**.** Let $\{f_i(x)\}_i$ be family of real-rooted polynomials of degree n, with $\alpha_1^i \leq \alpha_2^i \leq \cdots \leq \alpha_n^i$ denoting the roots of f_i. Then we say the

[1]Ignore β_1 if g has degree $n-1$.

polynomials $\{f_i(x)\}_i$ have a *common interlacing* if there exists $\gamma_1, \gamma_2, \dots, \gamma_n$ such that for all i,

$$\alpha_1^i \leq \gamma_1 \leq \alpha_2^i \leq \cdots \leq \alpha_n^i \leq \gamma_n.$$

The concept of interlacing is common in linear algebra, specifically with respect to *characteristic polynomials* of matrices. One such result is the fact the polynomial $\det(\mathbf{xI} - A)$ interlaces $\det(\mathbf{xI} - (A + \mathbf{v}\mathbf{v}^T))$. It can be derived from *Cauchy's interlacing lemma*[Hae], which we will use later in Chapter 3.

1.4.5 Laplacian Matrix of a graph

The *Laplacian* matrix of a graph G is defined as

$$L := D - A$$

where A is the adjacency matrix of G and D is the degree matrix of G, with

$$A_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

and

$$D_{i,j} = \begin{cases} degree(i), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

We will call the eigenvalues of L to be $\lambda_1, \lambda_2, \dots, \lambda_n$, with

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

It can be easily verified that the $\mathbf{1}$ vector is an eigenvector of L with eigenvalue 0.

Claim 1.4.1. For a graph $G = (V, E)$ and $\mathbf{x} \in \mathbb{R}^{|V|}$,

$$\mathbf{x}^T L \mathbf{x} = \sum_{(i,j) \in E} (x_i - x_j)^2$$

where L is the Laplacian matrix of G.

Proof. For any $\mathbf{x} \in \mathbb{R}^{|V|}$,

$$\mathbf{x}^T L\mathbf{x} = \sum_{e \in E} \mathbf{x}^T L_e \mathbf{x}$$

where L_e is the Laplacian for the graph with just the edge e. Say $e = (i, j)$. Then,

$$\mathbf{x}^T L_e \mathbf{x} = (x_i - x_j)^2$$

Therefore,

$$\mathbf{x}^T L\mathbf{x} = \sum_{(i,j) \in E} (x_i - x_j)^2$$

$\square$

The above claim also proves that the Laplacian matrix is a positive-semidefinite matrix, since $\mathbf{x}^T L\mathbf{x} \geq 0$, for all $\mathbf{x} \in \mathbb{R}^{|V|}$. Therefore the smallest eigenvalue is 0.

Further, it can be shown that the second smallest eigenvalue of the Laplacian matrix is non-zero if and only if the graph is connected.

1.4.6 Laplacian Systems and Electrical Circuits

The connection between solutions to Laplacian systems and electrical circuits provides crucial intuition to the work in this area. On a high level, it provides an abstraction on the physical importance of Laplacian solvers. We will show how voltages and current can be calculated using the solutions to Laplacian systems.

Say we are given a graph $G = (V, E)$. Give any orientation to the edges of G. We define vectors $\mathbf{b}_e \in \mathbb{R}^{|V|}$ for all $e \in E$ as

$$\mathbf{b}_e^i = \begin{cases} 1, & \text{if } i \text{ is the head of } e \\ -1, & \text{if } i \text{ is the tail of } e \\ 0, & \text{otherwise} \end{cases}$$

These vectors are called *edge-incidence vectors.* The matrix with these vectors as rows is called the *edge-incidence matrix*, and will be denoted as $B \in \mathbb{R}^{|E| \times |V|}$.[2]

It is easy to observe that $B^T B = L$.

Consider an electrical circuit based on the graph G. The edges will each have a one ohm resistance. We will denote the voltages on the nodes by $\mathbf{v} \in \mathbb{R}^{|V|}$ and current on the edges by $\mathbf{i} \in \mathbb{R}^{|E|}$.

Say $\mathbf{b} \in \mathbb{R}^{|V|}$ is the vector denoting the external input current to the nodes. By *Kirchoff's current law*, we know that for each node, the sum of incoming and outgoing current must be equal to the external input to that node. This can be expressed by the following equation,

$$B^T \mathbf{i} = b.$$

By *Ohm's law,* voltage difference across an edge must be equal to the product of current in that edge and the resistance of that edge (which is 1 ohm). Therefore,

$$B\mathbf{v} = \mathbf{i}.$$

Combining the above two equations, we get

$$B^T B\mathbf{v} = \mathbf{b} \quad \Rightarrow \quad L\mathbf{v} = \mathbf{b}$$

Hence, the voltages in the nodes of this circuit is the solution to the above Laplacian system. This gives a simple but a very crucial relationship between these two problems.

Corresponding to the external vector $\mathbf{b}$, the voltage difference across the endpoints of an edge $e = (i, j)$ is equal to

$$v_i - v_j = \mathbf{b}_e^T L^+ \mathbf{b}$$

The *effective resistance* between two nodes is defined as the voltage difference between two nodes when one unit of current is given to one node and taken out from the other. From our above characterization, we can infer that the effective resistance across the

[2]The head and tail of the edges are interchangeable.

edge e, denote by R_e is

$$R_e = \mathbf{b}_e^T L^+ \mathbf{b}_e$$

The above notions can be extended to weighted graphs. Say $W \in \mathbb{R}^{|V|\times|V|}$ is the diagonal *weight matrix*, then $L = B^T W B$, and the other definitions change correspondingly.

1.4.7 Laplacian solver

In the subsequent chapters, we will see how we can solve Laplacian systems in almost-linear time. With regard to this problem, the general theorem that we will be proving is the following:

Theorem 1.4.2. Given a graph G with m edges and n vertices, vector $\mathbf{b} \in \mathbb{R}^n$, and error parameter $\epsilon > 0$, we can find a vector $\mathbf{x}$ such that

$$||\mathbf{x} - L^+\mathbf{b}|| \leq \epsilon ||L^+\mathbf{b}||$$

in time $\tilde{\mathcal{O}}(m \log(1/\epsilon))$.

Chapter 2

Randomized Spectral Sparsification

2.1 Overview

The central purpose of Spectral Sparsification is the following : given a set of $n \times n$ real symmetric positive-semidefinite matrices $M_1, M_2, \dots, M_m$ whose sum is identity, find a weighted subset of small size whose sum has all eigenvalues close to 1. Formally, we would like to prove the following general statement:

Theorem 2.1.1. Let $\epsilon > 0$ and $M_i \in \mathbb{R}^{n \times n}$ for $i \in [1, m]$ be symmetric, positive semidefinite matrices such that

$$\sum_{i=1}^{m} M_i = \mathbf{I}$$

Then, in polynomial time, we can find scalars $s_i \geq 0$ for $i \in [1, m]$ such that

$$(1+\epsilon)^{-1}\mathbf{I} \preceq \sum_{i=1}^{m} s_i M_i \preceq (1+\epsilon)\mathbf{I}$$

and $|\{s_i | s_i \neq 0\}| = \mathcal{O}(n/\epsilon^2)$.

The original motivation for spectral sparsification was its application to graph sparsification, i.e., given a graph G, to find a subgraph H, such that the eigenvalues of the Laplacian of H are within a $(1+\epsilon)$ factor of the eigenvalues of the Laplacian of G.

The notion of spectral sparsification was introduced by Spielman and Teng [ST08] [ST11], who gave a randomized algorithm for constructing sparsifiers of size $\mathcal{O}(n \text{ polylog } n/\epsilon^2)$. They used the resultant sparsifiers to construct the first nearly linear-time Laplacian solver. Spielman and Srivastava gave another randomized algorithm for spectral sparsification, using the concept of sampling using effective resistances to obtain sparsifiers of size $\mathcal{O}(n \log n/\epsilon^2)$ [SS08] . We will describe this algorithm in a later section and analyze its complexity bounds.

A slightly looser version of spectral sparsification theorem can be proved using Matrix Chernoff bounds. The next section elaborates upon this connection.

2.2 Randomized Sparsification using Ahlswede-Winter Inequality

Matrix Chernoff bounds are Chernoff-like bounds for matrix-valued random variables. Several such bounds have been discovered, including the Rudelson inequality [Rud99] employed in the original proof of spectral sparsification theorem by Spielman and Srivastava, and the Ahlswede-Winter inequality [AW06], which can be used to get the spectral sparsification result.

Theorem 2.2.1 (Ahlswede-Winter inequality [AW06]). Let $Y \in \mathbb{R}^{n\times n}$ be a random, symmetric, positive semidefinite matrix such that $\mathbb{E}[Y] = \mu\mathbf{I}$[1]. Let $\rho := \sup_Y ||Y||$, and $Y_1, \ldots, Y_T$ be T i.i.d. copies of Y. Say $\epsilon > 0$ is a small constant. Then,

$$\Pr[||S - I|| > \epsilon] \leq 2n \cdot \exp(-\frac{T\epsilon^2\mu}{2\rho})$$

where $S := \frac{1}{\mu T}\sum_{i=1}^{T} Y_i$

[1]The statement is also valid if $\mathbb{E}[Y] = \mu \sum_{i=1}^{n'} e_i e_i^T$, where e_i's are standard basis vectors.

Proof of Theorem 2.1.1. We will prove Theorem 2.1.1 using the Ahlswede-Winter inequality by defining a suitable random process. We have matrices $M_1, M_2, \ldots, M_m$ whose sum is identity. We define the random variable Y that takes values $M_i/Tr[M_i]$ with probability $Tr[M_i]/n$. Then,

$$\mathbb{E}[Y] = \sum_{i=1}^{m} \frac{M_i}{Tr[M_i]} \frac{Tr[M_i]}{n} = \frac{1}{n} \cdot \mathbf{I}$$

Also,

$$\sup_{Y} ||Y|| = \sup_{i} \frac{||M_i||}{Tr[M_i]} \leq 1$$

Applying Ahlswede-Winter inequality on the above random process Y, we get the probability that after T iterations, S is ϵ-far from identity is less than

$$2n \cdot \exp(\frac{T\epsilon^2\mu}{2\rho}) \geq 2n \cdot \exp(\frac{T\epsilon^2}{2n})$$

For $T = \mathcal{O}(n \log n/\epsilon^2)$, the above probability becomes less than 1/2.

Therefore, we will have $\mathcal{O}(n \log n/\epsilon^2)$ samplings M_i with non-zero weights

$$s_i = \frac{1}{T \cdot Tr[M_i]}$$

If a matrix M_i is sampled multiple times, it's weight is added up. This proves Theorem 2.1.1 for $|\{s_i | s_i \neq 0\}| = \mathcal{O}(n \log n/\epsilon^2)$.

□

Corollary 2.2.1. Let $\beta > 0$ and $M_i \in \mathbb{R}^{n \times n}$ for $i \in [1, m]$ be symmetric, positive semidefinite matrices such that

$$\sum_{i=1}^{m} M_i = M'$$

Say we are given numbers q_i such that $q_i \geq Tr[M_i]$, for all $i \in [1, m]$. Let $W := \sum_{i=1}^{m} q_i$. Then we can find scalars $s_i \geq 0$ for $i \in [1, m]$ such that

$$M' \preceq 2 \sum_{i=1}^{m} s_i M_i \preceq 3M'$$

and $|\{s_i | s_i \neq 0\}| = \mathcal{O}(W \log W \log(1/\beta))$.

Proof. Using the probability distribution induced by $\{q_i\}$ in the proof of Theorem 2.1.1, and taking $\epsilon = 1/2$ will result in the above corollary. The number of samples needed to achieve the result with probability $1-\beta$ will be $\mathcal{O}(W \log W \log(1/\beta))$. $\square$

2.3 Randomized Graph Sparsification

As stated earlier, graph sparsification and its corresponding use in construction of Laplacian solvers was the main motivation for spectral sparsification. The main theorem for graph sparsification is the following :

Theorem 2.3.1 ([SS08]). Given a graph $G = (V, E)$ with n vertices and m edges, we can find a subgraph H of G with vertex set V, such that

$$(1+\epsilon)^{-1} L_G \preceq L_H \preceq (1+\epsilon) L_G$$

in time $\tilde{\mathcal{O}}(m)$, with high probability. The number of edges in H will be $\mathcal{O}(n/\epsilon^2)$.

Proof. [2] To get the above result from Theorem 2.2.1, we need to define an appropriate random process, as done in the proof of Theorem 2.1.1 from Ahlswede-Winter inequality.

Say we consider the following random process : with probability p_e, the random variable Y takes value $b_e b_e^T / p_e$. Then after T samplings $[Y_1, \ldots, Y_T]$,

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T} Y_i\right] = \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}[Y_i]$$

$$\mathbb{E}[Y] = \sum_{e \in E} p_e \cdot b_e b_e^T / p_e = L_G$$

Therefore,

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T} Y_i\right] = L_G$$

[2]The proof given here results in $E(H) = \mathcal{O}(n \log n/\epsilon^2)$. We will show the stronger bound later.

If we take our graph H to consist of edges sampled according to the distribution, and set the weight of edge e in H to be $1/(T \cdot p_e)$, then $\mathbb{E}[L_H] = L_G$.

However, we need to find a probability distribution so that L_H is spectrally close to L_G with high probability, and to make use of Ahlswede-Winter inequality in this regard, we will have to transform our random process accordingly to satisfy the assumptions.

Recall that Laplacian of G can be written as $L_G = BB^T$, where $B \in \mathbb{R}^{n \times m}$ is the edge-incidence matrix of G. We need to prove the following statement :

$$\begin{aligned}(1-\epsilon) \leq \frac{\mathbf{x}^T L_H \mathbf{x}}{\mathbf{x}^T L_G \mathbf{x}} &\leq (1+\epsilon) \\ \Rightarrow \quad \left| \frac{\mathbf{x}^T (L_H - L_G) \mathbf{x}}{\mathbf{x}^T L_G \mathbf{x}} \right| &\leq \epsilon \\ \Rightarrow \quad \left| \frac{\mathbf{x}^T (L_H - L_G) \mathbf{x}}{\mathbf{x}^T B B^T \mathbf{x}} \right| &\leq \epsilon\end{aligned}$$

Taking $\mathbf{y} = B^T \mathbf{x}$, we get $\mathbf{x} = L_G^+ B \mathbf{y}$, and

$$\left| \frac{\mathbf{y}^T B^T L_G^+ (L_H - L_G) L_G^+ B \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \right| \leq \epsilon$$

Therefore,

$$(1-\epsilon) \leq \frac{\mathbf{x}^T L_H \mathbf{x}}{\mathbf{x}^T L_G \mathbf{x}} \leq (1+\epsilon) \Rightarrow \quad \left| \frac{\mathbf{y}^T (B^T L_G^+ L_H L_G^+ B - B^T L_G^+ B) \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \right| \leq \epsilon \tag{1}$$

Note that $L_H = \sum_{e \in E(H)} w'(e) b_e b_E^T$, where w' is the weight function for H.

Using the alternate problem statement above, we can define our random process as the following : the random variable $Y = M_e / Tr[M_e]$ with probability $p_e = Tr[M_e] / \sum_e Tr[M_e]$, where $M_e = v_e v_e^T$ and $v_e = B^T L_G^+ b_e$.

Note that having fixed the random process also fixes our probability distribution. To be precise,

$$Tr[M_e] = ||v_e||^2 = b_e^T L_G^+ b_e = R_e$$

Therefore, the probability of choosing e is proportional to the effective resistance across the endpoints of edge e. We will also show that
$\sum_e Tr[M_e] = \sum_e R_e = n-1$.

All that needs to be done to effectively apply Ahlswede-Winter inequality is the analysis of $\mathbb{E}[Y]$.

$$\mathbb{E}[Y] = \sum_{e\in E} \frac{M_e}{Tr[M_e]} \frac{Tr[M_e]}{\sum_e Tr[M_e]} = \frac{1}{n-1} \cdot B^T L_G^+ B = \frac{1}{n-1}\Pi$$

Note that matrix Π is the projector onto the image of B^T. We will look at the properties of Π, and see that we can apply the concentration inequality for it.

Properties of Π

1. $\Pi^2 = B^T L_G^+ B B^T L_G^+ B = B^T L_G^+ B = \Pi$
2. Say λ is the eigenvalue of Π for eigenvector $\mathbf{x}$. Then.

   $$\Pi\mathbf{x} = \lambda\mathbf{x} \Rightarrow \quad \Pi^2\mathbf{x} = \lambda\Pi\mathbf{x} \Rightarrow \quad \Pi\mathbf{x} = \lambda^2\mathbf{x} \Rightarrow \quad \lambda\mathbf{x} = \lambda^2\mathbf{x}$$

   Therefore $\lambda = 0, 1$.
3. Π is isospectral with $BB^T L_G^+$, which is identity on the $n-1$ dimensional space orthogonal to the vector $\mathbf{1}$. Hence Π has rank $n-1$.

Since Π is unitarily equivalent to $\sum_{i=1}^{n-1} e_i e_i^T$, we can apply Ahlswede-Winter inequality for this matrix.

Say $\tilde{\Pi} = \frac{1}{T}\sum_{i=1}^{T} Y_i$. Then,

$$\tilde{\Pi} = \frac{1}{T} \sum_{e\in E(H)} \frac{M_e}{Tr[M_e]} = B^T L_G^+ L_H L_G^+ B$$

Therefore from (1), we know that

$$Pr\left[\left|\frac{\mathbf{x}^T(L_H - L_G)\mathbf{x}}{\mathbf{x}^T L_G \mathbf{x}}\right| > \epsilon\right] \leq Pr[||\tilde{\Pi} - \Pi|| > \epsilon]$$

and from Ahlswede-Winter inequality, we know that

$$Pr[||\tilde{\Pi} - \Pi|| > \epsilon] \leq 1/2$$

for $T = \mathcal{O}(n \log n/\epsilon^2)$. Hence, after T iterations, we will get a graph H such that,

$$Pr\left[\left|\frac{\mathbf{x}^T(L_H - L_G)\mathbf{x}}{\mathbf{x}^T L_G \mathbf{x}}\right| > \epsilon\right] \leq 1/2$$

Therefore, we have shown that we can obtain a good sparsified graph with $\mathcal{O}(n \log n/\epsilon^2)$ edges. □

The reason why sampling proportional to effective resistances works can be seen intuitively. If two nodes in the graph are well-connected [many paths between the nodes], then the effective resistance between these nodes is small. Correspondingly, an edge is important if there are very few paths connecting its endpoints. The effective resistance of such an edge will be relatively large. A good sparsifier will include edges which have high resistance. Therefore sampling proportional to effective resistances makes sense.

In a random walk setting, the *commute time* between two nodes is proportional to effective resistance between the nodes [CRR+96]. If the commute time between two nodes is small, a random walk from one node takes relatively few number of steps to reach the other node. Therefore, if these nodes are connected by an edge, then that edge is less important, and can be assigned lesser probability. This can be considered another justification for the sampling procedure.

Equivalent to Corollary 2.2.1, we get the following corollary of graph sparsification for weighted graphs.

Corollary 2.3.1. Given a graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, and $\beta > 0$. Say we have numbers q_e such that $q_e \geq w(e)R_e$, for all $e \in E$, where R_e is the effective resistance across endpoints of e. Let $W := \sum_{e \in E} q_e$. Then we can find a subgraph H of

G with vertex set V, such that

$$L_G \preceq 2L_H \preceq 3L_G$$

with probability $1-\beta$. The number of edges in H will be $\mathcal{O}(W \log W \log(1/\beta))$.

Proof. The proof of the corollary follows in a fashion similar to the proof of Corollary 2.2.1, that is, sampling with probability proportional to q_e, and taking $\epsilon > 0$. □

2.4 Stretch Sparsifiers

Corollary 2.4.1 (Stretch Sparsfiers). Given a graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$ and for any $\gamma < m$ and $\beta > 0$, we can compute a subgraph H of G with $n - 1 + \tilde{\mathcal{O}}((m \log^2 n/\gamma) \log \frac{1}{\beta})$ edges such that

$$L_G \preceq 2L_H \preceq 3\gamma L_G$$

with probability atleast $1-\beta$.
The algorithm takes $\tilde{\mathcal{O}}((m \log n + n \log^2 n + m \log^2 n/\gamma) \log \frac{1}{\beta})$ time.

Proof. From Corollary 2.3.1, we know that if we can find numbers q_e, then a sparsifier can be constructed. Here we will establish the proof by finding such a set $\{q_e\}$.

Say T is a spanning tree of the graph G. Then stretch of an edge e w.r.t to T, $\mathrm{str}_T(e)$, is defined as the length of the path between the endpoints of e in T. Clearly, for any T and all $e \in E$,

$$\mathrm{str}_T(e) \geq w(e) R_e$$

Therefore, we can use $\mathrm{str}_T(e)$ as q_e. Correspondingly,

$$W = \sum_{e \in E} q_e = n - 1 + \sum_{e \in E - T} \mathrm{str}_T(e) = n - 1 + \mathrm{str}_T(G)$$

where, $\mathrm{str}_T(G)$ is called the stretch of G w.r.t T.

From the results of Elkin, et al [EEST08], we know that we can find a low-stretch spanning tree T in linear time.

Theorem 2.4.1 ([EEST08]). For any undirected graph G, a spanning tree T can be constructed in $\widetilde{\mathcal{O}}(m \log n + n \log n \log \log n)$ such that the stretch corresponding to T in G is $\widetilde{\mathcal{O}}(m \log n)$.

We will correspondingly use a low-stretch spanning tree for T in further construction.

Hence, by Corollary 2.3.1, the number of samples needed to get a good sparsifer is

$$\mathcal{O}(W \log W \log(1/\beta)) = \tilde{\mathcal{O}}(m \log^2 n \log(1/\beta))$$

However, the above bound does not imply any reduction in the number of edges, and is useless for our purpose. We will have to suitably modify the distribution to prove the corollary.

To that end, construct graph G' from G, with same vertex and edge set, but with the weight function w', such that, $w'(e) = \gamma w(e)$ if $e \in T$, and $w'(e) = w(e)$ if $e \notin T$. It is easy to see that

$$L_G \preceq L'_G \preceq \gamma L_G \tag{2}$$

Also for this new graph, $\mathrm{str}_T(G') = \tilde{\mathcal{O}}(m \log^2 n/\gamma)$.

Simultaneously, redefine $q_e = \mathrm{str}_T(e)$ if $e \in T$, and $q_e = \mathrm{str}_T(e)/\gamma$ if $e \notin T$. It can be checked that they still satisfy $q_e \geq w'(e)R_e$.

Using the probability distribution induced by q_e, by Corollary 1.2, we can find a graph H such that

$$L'_G \preceq 2L_H \preceq 3L'_G$$

Applying (2), we get

$$L_G \preceq 2L_H \preceq 3\gamma L_G$$

We will analyze the number of tree edges and non-tree edges in H separately. The number of tree-edges is trivially upper bounded by $n-1$. We will use Chernoff bound to find the number of non-tree edges sampled.

Say random variable $X_i = 1$, if the i-th sampling is a non-tree edge, and 0 otherwise. Let $X = \sum_{i=1}^{T} X_i$, where $T = \mathcal{O}(W \log W \log(1/\beta))$ is the number of samples taken. Then X denotes the number of non-tree edges.

$$Pr[X_i = 1] = \frac{\sum_{e \in E-T} q_e}{W}$$

$$\mathbb{E}[X_i] = \sum_{e \in E-T} \frac{q_e}{W} = \frac{\mathrm{str}_T(G')}{W}$$

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=1}^{T} \mathbb{E}[X_i] \\
&= \frac{\mathrm{str}_T(G')}{W} \cdot W \log W \log(1/\beta) \\
&= \mathrm{str}_T(G') \log W \log(1/\beta) \\
&= \tilde{\mathcal{O}}((m \log^2 n/\gamma) \log \frac{1}{\beta})
\end{aligned}$$

By Chernoff bound,

$$\begin{aligned}
Pr[X \geq (1+\delta)\mathbb{E}[X]] &\leq e^{-\delta^2 \mathbb{E}[X]/3} \\
&\leq \frac{1}{n^{\mathcal{O}(\delta^2)}}
\end{aligned}$$

Therefore, with high probability, the number of edges in H is less $n-1+\tilde{\mathcal{O}}((m \log^2 n/\gamma) \log \frac{1}{\beta})$ and it satisfies the spectral condition as well.

The reweighting of non-tree edges might seem "out-of-the-blue", but can be considered intuitive. The first step to a good sparsifier is to make sure the sparsified graph remains connected. To that end, we need to select a skeleton of the graph, which in this case is the low-stretch spanning tree T. The tree-edges are scaled higher than non-tree edges to ensure this.

Next we should add those non-tree edges which have larger separation in T, since they affect the connectivity more negatively than others. So we choose a probability distribution proportional to the separation distance in T, which is exactly the stretch of an edge w.r.t. T.

□

We will use *stretch sparsifiers* in the construction of the nearly-linear time Laplacian solver, since it gives us a resistance-independent way of finding a good sparsifier.

In general, to obtain a sparsifier using the above algorithm, one needs to be able to sample proportional to effective resistances. However, to compute effective resistances we need to solve Laplacian systems, since for any edge e, effective resistance across e is $\mathbf{b}_e^T L^+ \mathbf{b}_e$.

Spielman and Srivastava provided a way to approximately compute effective resistances. They used the fact that effective resistances are euclidean distances for a certain embedding corresponding to the vertices of the graph. They then used the *John-Lindestrauss* lemma to approximately calculate these distances. We refer the reader to [SS08] for a detailed proof of the same.

Chapter 3

Deterministic Spectral Sparsification

3.1 Overview

In the previous section, we saw that spectral sparsification is indeed possible, and looked at a randomized algorithm which achieves the sparsification goal within a factor of $\log n$.

We will now study the result of Batson, Spielman and Srivastava [BSS09], where they give a novel deterministic sparsification algorithm, which is not just important and appealing because it's deterministic, but also due to the novelty and uniqueness of the methods used.

In particular, they proved the following theorem :

Theorem 3.1.1 ([BSS09]). Given $d > 1$ and vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^n$, with

$$\sum_{i=1}^{m} \mathbf{v}_i \mathbf{v}_i^T = \mathbf{I}$$

then, in polynomial time, we can find scalars $s_i \geq 0$ for $i \in [m]$ such that

$$\left(1 - \frac{1}{\sqrt{d}}\right)^2 \mathbf{I} \preceq \sum_{i=1}^{m} s_i \mathbf{v}_i \mathbf{v}_i^T \preceq \left(1 + \frac{1}{\sqrt{d}}\right)^2 \mathbf{I}$$

and $|\{s_i | s_i \neq 0\}| \leq \lceil dn \rceil$.

Note that taking $d = 1/\epsilon^2$ gives us a spectral sparsifier of size $\mathcal{O}(n/\epsilon^2)$, which is better than the sparsifier given by the randomized algorithm.

One deterministic method of constructing a sparsifier would be finding an *appropriate* vector in each step, and adding it to our sparsifier set. The conditions that characterize the *appropriateness* of a vector is the topic that we explore in the following sections.

3.2 Rank one updates

Given a symmetric matrix A, we denote the characteristic polynomial of A as

$$\chi(A)(x) = \det(x\mathbf{I} - A).$$

The roots of this polynomial are the eigenvalues of A. The following theorem tells us that on adding $\mathbf{v}\mathbf{v}^T$ to A, the eigenvalues of the new matrix $A + \mathbf{v}\mathbf{v}^T$ interlace the eigenvalues of A.

Lemma 3.2.1 (Cauchy's Interlacing Theorem [Hae]). If the eigenvalues of A are α_i and eigenvalues of $A + \mathbf{v}\mathbf{v}^T$ are β_i, then

$$\alpha_1 \leq \beta_1 \leq \alpha_2 \leq \ldots \alpha_n \leq \beta_n.$$

In general, interlacing is a concept associated with the roots of polynomials, and so we say that $\chi(A + \mathbf{v}\mathbf{v}^T)$ interlaces $\chi(A)$.

In particular, following is the exact representation of $\chi(A + \mathbf{v}\mathbf{v}^T)$ w.r.t. $\chi(A)$.

Lemma 3.2.2 ([DZ07]). For a non-singular matrix A and a vector $\mathbf{v}$,

$$\det(A + \mathbf{v}\mathbf{v}^T) = \det(A)(1 + \mathbf{v}^T A^{-1}\mathbf{v})$$

Using the above lemma,

$$\begin{aligned}\chi(A+\mathbf{v}\mathbf{v}^T)(x) &= \det(x\mathbf{I}-A-\mathbf{v}\mathbf{v}^T)\\ &= \det(x\mathbf{I}-A)(1-\mathbf{v}^T(x\mathbf{I}-A)^{-1}\mathbf{v})\\ &= \chi(A)(x)\left(1-\sum_{i=1}^{n}\frac{\langle \mathbf{v},\mathbf{u}_i\rangle^2}{x-\lambda_i}\right)\end{aligned}$$

where $\{u_i\},\{\lambda_i\}$ are the eigenvectors and eigenvalues of A.

3.3 Interlacing polynomials

A typical deterministic algorithm for sparsification will add one vector to the solution space in each iteration. So it makes sense that we first analyze the change in matrix under rank one matrix updates.

In the above setting, at each iteration, we are given a symmetric matrix A, and vectors $\mathbf{v}_1,\ldots,\mathbf{v}_m$. The Cauchy's interlacing lemma tells us that polynomials

$$\chi(A+\mathbf{v}_1\mathbf{v}_1^T)(x),\chi(A+\mathbf{v}_2\mathbf{v}_2^T)(x),\ldots,\chi(A+\mathbf{v}_m\mathbf{v}_m^T)(x)$$

have a common interlacing. In particular, they all interlace with $\chi(A)$.

Marcus, Spielman and Srivastava, in their work on interlacing polynomial families [MSS14], shed light on the relationship between the roots of a given family of real-rooted polynomials and the roots of the expected polynomial of this family, with respect to a given distribution. They showed that under certain conditions, atleast one polynomial from a given collection of polynomials must follow the behaviour of the expected polynomial.

More formally, we have the following theorem for polynomials with a common interlacing.

Theorem 3.3.1 ([MSS13]). Say we have real-rooted, degree n polynomials $f_1,f_2,\ldots,f_m$ with positive leading coefficients. Let $\lambda_k(f_j)$ denote the k-th largest root of f_j and let u be any distribution on $[m]$. If $f_1,f_2,\ldots,f_m$ have a common interlacing, then for all

$k \in \{1, 2, \ldots, n\}$

$$\min_j \lambda_k(f_j) \le \lambda_k(\mathbb{E}_{j\sim\mu}[f_j]) \le \max_j \lambda_k(f_j)$$

Now we know that polynomials $\chi(A + \mathbf{v}_1\mathbf{v}_1^T), \ldots, \chi(A + \mathbf{v}_m\mathbf{v}_m^T)(x)$ have a common interlacing.

So using Theorem 3.3.1 we now know that for every k, there exists a j such that the k-th largest root of $\chi(A + \mathbf{v}_j\mathbf{v}_j^T)$ is atleast as large as the corresponding root of $\mathbb{E}_{j\sim\mu}[\chi(A + \mathbf{v}_j\mathbf{v}_j^T)]$. Therefore we will try and explore the properties of the expected rank-one update polynomial, under the settings of Theorem 3.1.1.

$$\begin{aligned}\mathbb{E}_j[\chi(A + \mathbf{v}_j\mathbf{v}_j^T)] &= \mathbb{E}_j[\det(x\mathbf{I} - A - \mathbf{v}_j\mathbf{v}_j^T)] \\ &= \chi(A)\left(1 - \sum_{i=1}^{n} \frac{\mathbb{E}_j[\langle \mathbf{v}_j, \mathbf{u}_i\rangle^2]}{x - \lambda_i}\right)\end{aligned}$$

Since all v_i's are equally likely, probability of choosing v_i is $1/m$.

$$\begin{aligned}\mathbb{E}_j[\langle \mathbf{v}_j, \mathbf{u}_i\rangle^2] &= \mathbf{u}_i^T(\mathbb{E}_j[\mathbf{v}_j\mathbf{v}_j^T])\mathbf{u}_i \\ &= \frac{1}{m}\mathbf{u}_i^T\mathbf{u}_i = \frac{1}{m}\end{aligned}$$

The second inequality follows because $\sum_{i=1}^m \mathbf{v}_i\mathbf{v}_i^T = \mathbf{I}$. Therefore,

$$\begin{aligned}\chi(A + \mathbf{v}_j\mathbf{v}_j^T) &= \chi(A)\left(1 - \sum_{i=1}^{n} \frac{1/m}{x - \lambda_i}\right) \\ &= \chi(A) - \frac{1}{m}\frac{d}{dx}\chi(A) \\ &= (1 - \frac{1}{m}D)\chi(A)\end{aligned}$$

where D is the $\frac{d}{dx}$ operator.

At the beginning of the algorithm, we will have $A = 0$ and $\chi(A) = x^n$. After k iterations, the expected characteristic polynomial will be

$$(1 - \frac{1}{m}D)^k x^n$$

The above polynomial belongs to the well known class of *associated Laguerre polynomials*[Kra03]. We will derive the bounds on the eigenvalues of this polynomials, and in the process try to find a deterministic algorithm which choses vectors which follow the expected behaviour.

Note that process that we have defined is the following : starting with $A = 0$, add a suitable vector v_i, for $i \in [m]$ in each iteration to A, and we want to analyze the expected behaviour of such a process. Say that $u_1, u_2, \ldots, u_k$ are the vectors chosen across k iterations. Then, by the interlacing property defined in Theorem 3.3.1, we know that with positive probability,[1]

$$\lambda_k\left(\sum_{i=1}^{k} u_i u_i^T\right) \geq \lambda_k\left(\mathbb{E}\left[\sum_{i=1}^{k} u_i u_i^T\right]\right)$$

From the analysis of the expected polynomial above, we know that

$$\mathbb{E}\left[m \cdot \sum_{i=1}^{k} u_i u_i^T\right] = (1-D)^k x^n = x^{n-k}(1-D)^n x^k$$

The second equality can be observed by expansion of the expressions and term-by-term verification.

Therefore, to bound the k-th root of $(1-D)^k x^n$, we derive a lower bound on the lowest root of $(1-D)^n x^k$, which will be the polynomial that we analyze in the next section.[2]

3.4 Analysis of the Associated Laguerre Polynomial

The idea is to study the behaviour of a polynomial f under the $(1-D)$ operation.

$$(1-D)f = f - f' = f(1 - \frac{f'}{f}) = f(1 + \Phi_f)$$

[1]We are analyzing the k-th largest root since the roots smaller than this will be zero.
[2]The process to get an upper bound also follows a similar analysis.

where $\Phi_f = -\frac{f'}{f}$

$$\begin{aligned}\Phi_f(x) = -\frac{f'(x)}{f(x)} &= -\frac{d\log f(x)}{dx} \\ &= -\frac{d}{dx}\left(\log \prod_{i=1}^{n}(x-\lambda_i)\right) \\ &= \sum_{i=1}^{n}\frac{1}{\lambda_i - x}\end{aligned}$$

λ_i's are the roots of polynomial f.

We will refer to this function as the lower barrier function. This is because if x is less than the roots of f, then Φ_f quantifies the measure of separation. If x gets close to the lowest root (or any other), Φ_f shoots to infinity. Thus, in some sense, it functions as a barrier.

Note that for x strictly less than the roots of f, $\Phi_f(x)$ is increasing, positive and convex.

For every $(1 - D)$ operation, we would like to keep the value of lower barrier function bounded above. In particular say

$$b := min\{x \in \mathbb{R} \mid \Phi_f(x) = \epsilon.\}$$

Then we will prove the following lemma on the behaviour of the lower barrier function under the $(1 - D)$ operation.

Lemma 3.4.1. Given that $\Phi_f(b) = \epsilon$, then for all $\delta \leq \frac{1}{1+\epsilon}$,

$$\Phi_{(1-D)f}(b+\delta) \leq \epsilon$$

Proof.

$$\begin{aligned}\Phi_{(1-D)f} = -\frac{((1-D)f)'}{(1-D)f} &= -\frac{(f(1+\Phi_f))'}{f(1+\Phi_f)} \\ &= \Phi_f - \frac{\Phi_f'}{1+\Phi_f}\end{aligned}$$

$$\Rightarrow \quad \Phi_{(1-D)f}(b+\delta) = \Phi_f(b+\delta) - \frac{\Phi'_f(b+\delta)}{1+\Phi_f(b+\delta)}$$

We want $\Phi_{(1-D)f}(b+\delta) \leq \epsilon = \Phi_f(b)$. This is equivalent to saying that

$$\begin{aligned} & \Phi_f(b+\delta) - \frac{\Phi'_f(b+\delta)}{1+\Phi_f(b+\delta)} \leq \Phi_f(b) \\ \Rightarrow \quad & \Phi_f(b+\delta) - \Phi_f(b) \leq \frac{\Phi'_f(b+\delta)}{1+\Phi_f(b+\delta)} \\ \Rightarrow \quad & \frac{\Phi'_f(b+\delta)}{\Phi_f(b+\delta)-\Phi_f(b)} - \Phi_f(b+\delta) \geq 1 \end{aligned} \tag{2}$$

Expanding Φ'_f and Φ_f as a sum of terms and applying Cauchy-Schwartz tells us that

$$\frac{\Phi'_f(b+\delta)}{\Phi_f(b+\delta)-\Phi_f(b)} - \Phi_f(b+\delta) \geq \frac{1}{\delta} - \Phi_f(b) \tag{3}$$

Therefore, to ensure 2, we need

$$\frac{1}{\delta} - \Phi_f(b) \geq 1 \;\Rightarrow\quad \delta \leq \frac{1}{1+\epsilon}.$$

□

The above lemma also tell us that for $\delta \leq (1+\epsilon)^{-1}$, the roots of $(1-D)f$ are lower bounded by $b+\delta$.

Consider the following inverse function of Φ_f :

$$\mathrm{smin}_\epsilon(f) := \min\{x \in \mathbb{R} \mid \Phi_f(x) = \epsilon\}$$

One can see that $\mathrm{smin}_\epsilon(f) = b$ and also that it defines a lower bound on the roots of f, such that ϵ functions as a sensitivity parameter to this measure.

Under the operation $(1-D)$, we would like to observe how does the lower bound change. The above lemma yields the following corollary in this regard.

Corollary 3.4.1. $\mathrm{smin}_\epsilon((1-D)f) \geq \mathrm{smin}_\epsilon(f) + (1+\epsilon)^{-1}$

Applying the corollary k times gives us the following bound on the lowest root of $(1-D)^n x^k$.

$$\begin{aligned}\lambda_{min}((1-D)^n x^k) &\geq \mathrm{smin}_\epsilon((1-D)^n x^k)) \\ &\geq \mathrm{smin}_\epsilon(x^k) + \frac{n}{1+\epsilon} \\ &= -\frac{k}{\epsilon} + \frac{n}{1+\epsilon}\end{aligned}$$

Taking $\epsilon = \frac{\sqrt{k}}{\sqrt{n}-\sqrt{k}}$ gives us

$$\lambda_{min}((1-D)^n x^k) \geq n\left(1-\sqrt{\frac{k}{n}}\right)^2$$

In particular, the lemma and the corollary tells us that to keep the value of the lower barrier function bounded across all operations of $(1-D)$, we should increase the lower barrier by atmost $(1+\epsilon)^{-1}$.

We can similarly also define an upper barrier function $\Phi_f = f'/f$, and the corresponding inverse function $\mathrm{smax}_\epsilon(f)$ and in a similar fashion, derive the following lemma:

Lemma 3.4.2. $\mathrm{smax}_\epsilon((1-D)f) \leq \mathrm{smax}_\epsilon(f) + \frac{1}{1-\epsilon}$

Again, the lemma tells us that to keep the upper barrier function (and simultaneously the maximum root) bounded, we must increase the upper barrier by atleast $(1-\epsilon)^{-1}$.

Taking $\epsilon = \frac{\sqrt{k}}{\sqrt{n}+\sqrt{k}}$ gives us

$$\lambda_{max}((1-D)^n x^k) \leq n\left(1+\sqrt{\frac{k}{n}}\right)^2$$

Therefore, we have an upper and lower bound on the roots of the expected characteristic polynomial after k iterations.

In particular, for $k = dn$ iterations, as required in the statement of Theorem 3.1.1, the ratio of the largest and smallest roots of the expected characteristic polynomial will be

$$\frac{d + 2\sqrt{d} + 1}{d - 2\sqrt{d} + 1}$$

This is the exact bound that we require in Theorem 3.1.1. Infact the above explanation, along with the interlacing property of Theorem 3.3.1, intuitively tells us that in each iteration, there exists a vector whose addition to the existing system should follow the behaviour expected on addition of the average vector.

3.5 Deterministic Sparsification Algorithm using Barrier Functions

The deterministic algorithm given by Batson, Spielman and Srivastava [BSS09] made use of barrier functions on eigenvalues of the system, to keep them bounded across all iterations.

Intuitively speaking, we want to choose a vector in each iteration which simulates the behaviour in the expected case. From the analysis of associated Laguerre polynomial, we know that using the barrier function Φ, we can effectively capture the behaviour of the $(1 - D)$ operator on the extremal eigenvalues. We will make use of similar barrier function to keep the eigenvalues of system bounded across all iterations in the deterministic algorithm.

Formally, say we want to keep the eigenvalues of the current $n \times n$ matrix A between u and l. Then the upper barrier function will be

$$\Phi_u(A) := \sum_{i=1}^{n} \frac{1}{u - \lambda_i} = \mathrm{Tr}(u\mathbf{I} - A)^{-1}$$

and the lower barrier function will be

$$\Phi_l(A) := \sum_{i=1}^{n} \frac{1}{\lambda_i - l} = \mathrm{Tr}(A - l\mathbf{I})^{-1}.$$

As before, the physical importance of barrier functions lie in the fact that if the eigenvalues get close to the boundaries, the value of barrier functions shoot to infinity. Therefore, keeping the value of barrier functions finite, along with other necessary conditions, should ensure that the eigenvalues remain bounded.

The algorithm involves starting with an empty matrix, say A_0, and adding an appropriate rank-one matrix $s\mathbf{v}\mathbf{v}^T$ in each iteration, for $T = dn$ iterations. Say we construct the sequence of matrices $A_0(=0), A_1, \ldots A_T$, with A_i corresponding to the matrix constructed at the i-th iteration.

In each iteration, adding a positive rank-one matrix will increase the eigenvalues of the current matrix. Correspondingly, we will have to increase the upper and lower bounds by a small amount, to ensure that we keep the eigenvalues bounded.

In particular, we will fix parameters $\epsilon_u, \epsilon_l, \delta_u, \delta_l, u_0, l_0$, such that in each of $i = 1$ to T iterations we can find a vector v and scalar s which satisfies all the conditions below:

1. $\Phi_{u_0}(A_0) = \frac{n}{u_0} = \epsilon_u$
2. $\Phi_{l_0}(A_0) = \frac{n}{l_0} = \epsilon_l$
3. $\Phi_{u+\delta_u}(A_{i+1}) \leq \Phi_u(A_i) \leq \epsilon_u$, where $u = u_0 + i\delta_u$
4. $\Phi_{l+\delta_l}(A_{i+1}) \leq \Phi_l(A_i) \leq \epsilon_l$, where $l = l_0 + i\delta_l$
5. $\lambda_{max}(A_{i+1}) \leq u + \delta_u$ and $\lambda_{min}(A_{i+1}) \geq l + \delta_l$

We will find the equivalent conditions on the feasible vectors (which are also computable in polynomial time), such that the above five are sastisfied. In particular the following lemma should hold:

Theorem 3.5.1. $\Phi_{l'}(A + s\mathbf{v}\mathbf{v}^T) \leq \Phi_l(A)$ if and only if $s\mathbf{v}^T L_A \mathbf{v} \geq 1$, where

$$L_A = \frac{(A - l'\mathbf{I})^{-2}}{\Phi_{l'}(A) - \Phi_l(A)} - (A - l'\mathbf{I})^{-1}, \text{ and}$$

$$l' = l + \delta_l$$

To prove the above theorem, we will use the Sherman-Morrison formula, on the behaviour of inverse of a matrix under rank-one updates.

Lemma 3.5.1 (Sherman-Morrison formula [SM50]). Given a non-singular matrix A and a vector v

$$(A + \mathbf{v}\mathbf{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{v}\mathbf{v}^T A^{-1}}{1 + \mathbf{v}^T A^{-1}\mathbf{v}}$$

Proof of Theorem 3.5.1.

$$\begin{aligned}
\Phi_{l'}(A + s\mathbf{v}\mathbf{v}^T) &= \mathrm{Tr}(A + s\mathbf{v}\mathbf{v}^T - l'\mathbf{I})^{-1} \\
&= \mathrm{Tr}(A - l'\mathbf{I})^{-1} - \mathrm{Tr}(\frac{s(A - l'\mathbf{I})^{-1}\mathbf{v}\mathbf{v}^T(A - l'\mathbf{I})^{-1}}{1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v}}) \\
&= \mathrm{Tr}(A - l'\mathbf{I})^{-1} - \frac{s\mathbf{v}^T(A - l'\mathbf{I})^{-2}\mathbf{v}}{1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v}} \\
&= \Phi_{l'}(A) - \frac{s\mathbf{v}^T(A - l'\mathbf{I})^{-2}\mathbf{v}}{1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v}}
\end{aligned}$$

We want

$$\begin{aligned}
& \Phi_{l'}(A + s\mathbf{v}\mathbf{v}^T) \leq \Phi_l(A) \\
\Leftrightarrow\ & \Phi_{l'}(A) - \frac{s\mathbf{v}^T(A - l'\mathbf{I})^{-2}\mathbf{v}}{1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v}} \leq \Phi_l(A) \\
\Leftrightarrow\ & \frac{s\mathbf{v}^T(A - l'\mathbf{I})^{-2}\mathbf{v}}{1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v}} \geq \Phi_{l'}(A) - \Phi_l(A) \\
\Leftrightarrow\ & \frac{s\mathbf{v}^T(A - l'\mathbf{I})^{-2}\mathbf{v}}{\Phi_{l'}(A) - \Phi_l(A)} \geq 1 + s\mathbf{v}^T(A - l'\mathbf{I})^{-1}\mathbf{v} \\
\Leftrightarrow\ & s\mathbf{v}^T\left(\frac{(A - l'\mathbf{I})^{-2}}{\Phi_{l'}(A) - \Phi_l(A)} - (A - l'\mathbf{I})^{-1}\right)\mathbf{v} \geq 1 \\
\Leftrightarrow\ & s\mathbf{v}^T L_A \mathbf{v} \geq 1
\end{aligned}$$

One could also prove this theorem using the proof of Lemma 3.4.1. In particular, note that the condition (2) exactly corresponds to the condition $s\mathbf{v}^T L_A\mathbf{v} \geq 1$.

Therefore one can see that we want each iteration in our algorithm to follow behaviour similar to the application of the $(1 - D)$ operator. □

Corollary 3.5.1. If Theorem 3.3 holds then $\lambda_{min}(A + s\mathbf{v}\mathbf{v}^T) \geq l + \delta_l$.

Proof. Say $\lambda_{min}(A+s\mathbf{v}\mathbf{v}^T) < l+\delta_l$, then there exists $s' < s$ such that $\lambda_{min}(A+s'\mathbf{v}\mathbf{v}^T) = l+\delta_l$. In that case, $\Phi_{l'}(A+s'\mathbf{v}\mathbf{v}^T) = \infty$. But by Theorem 1.3, we know $\Phi_{l'}(A+s'\mathbf{v}\mathbf{v}^T)$ must be finite.

Therefore, $\lambda_{min}(A+s\mathbf{v}\mathbf{v}^T) \geq l+\delta_l$. □

Similarly, we can prove the following theorem on the bound on lower barrier function.

Theorem 3.5.2 ([BSS09]). $\Phi_{u'}(A+s\mathbf{v}\mathbf{v}^T) \leq \Phi_u(A)$ if and only if $s\mathbf{v}^T U_A \mathbf{v} \leq 1$, where

$$U_A = \frac{(u'\mathbf{I}-A)^{-2}}{\Phi_u(A)-\Phi_{u'}(A)} + (u'\mathbf{I}-A)^{-1}, \text{ and}$$

$$u' = u+\delta_u$$

Corollary 3.5.2. If Theorem 3.4 holds then $\lambda_{max}(A+s\mathbf{v}\mathbf{v}^T) \leq u+\delta_u$.

Hence, if we prove the existence of s and $\mathbf{v}$ such that $s\mathbf{v}^T U_A\mathbf{v} \leq 1$ and $s\mathbf{v}^T L_A\mathbf{v} \geq 1$, then conditions 1-5 will all be satisfied.

Note that if we instead prove that $\mathbb{E}_v[\mathbf{v}^T U_A\mathbf{v}] \leq \mathbb{E}_v[\mathbf{v}^T L_A\mathbf{v}]$, then we can find a $\mathbf{v}$ which satisfies $\mathbf{v}^T U_A\mathbf{v} \leq \mathbf{v}^T L_A\mathbf{v}$ and set s such that 1 lies between the quantities.

Lemma 3.5.2. If $\Phi_l(A) \leq \epsilon_u$, then

$$\mathbb{E}_v[\mathbf{v}^T L_A\mathbf{v}] \geq \frac{1}{\delta_l} + \epsilon_l$$

Proof.

$$\begin{aligned}
\mathbb{E}_v[\mathbf{v}^T L_A \mathbf{v}] &= \mathbb{E}[\mathrm{Tr}(L_A \mathbf{v}\mathbf{v}^T)] \\
&= \mathrm{Tr}(\mathbb{E}[L_A \mathbf{v}\mathbf{v}^T]) \\
&= \mathrm{Tr}(L_A \cdot \mathbb{E}[\mathbf{v}\mathbf{v}^T]) \\
&= \mathrm{Tr}(L_A) \\
&= \frac{\mathrm{Tr}(u'\mathbf{I} - A)^{-2}}{\Phi_u(A) - \Phi_{u'}(A)} + \mathrm{Tr}(u'\mathbf{I} - A)^{-1} \\
&\geq \frac{1}{\delta_l} - \Phi_{l'}(A) \qquad (4) \\
&\geq \frac{1}{\delta_l} - \epsilon_l
\end{aligned}$$

To prove the inequality at 4 one can either expand L_A as a sum of terms and apply Cauchy-Schwartz or directly infer the result from the equivalent step 3 in the analysis of the expected case.

□

Note that this lemma is equivalent to the step (3) in the analysis of the expected case. Particularly, it is ensured that the each iteration follows the expected behaviour of $(1 - D)$ operation.

The matrix U_A also satisfies a similar lemma.

Lemma 3.5.3. If $\Phi_u(A) \leq \epsilon_u$, then

$$\mathbb{E}_v[\mathbf{v}^T U_A \mathbf{v}] \leq \frac{1}{\delta_u} + \epsilon_u$$

The proof is along the lines of the proof of Lemma 3.4.2.

We will set the values of $\delta_u, \delta_l, \epsilon_u, \epsilon_l$ such that they satisfy

$$\frac{1}{\delta_u} + \epsilon_u \leq \frac{1}{\delta_l} - \epsilon_l \qquad (5)$$

and this will ensure that

$$\mathbb{E}_v[\mathbf{v}^T U_A \mathbf{v}] \leq \mathbb{E}_v[\mathbf{v}^T L_A \mathbf{v}]$$

In particular, the values of the parameters are the following

$$\epsilon_u = \frac{\sqrt{d}-1}{d+\sqrt{d}}$$

$$\epsilon_l = \frac{1}{\sqrt{d}}$$

Lemma 3.4.1 and 3.4.2 tells us the increment in the lower and upper barrier to ensure that the values of the lower and upper barrier functions remain bounded. Using those expressions, we can set

$$\delta_u = \frac{\sqrt{d}+1}{\sqrt{d}-1}$$

$$\delta_l = 1$$

One can verify that these values satisfy (5). Therefore, using these values for the parameters, we know we can find a vector $\mathbf{v}$ in each iteration such that $\mathbf{v}^T U_A \mathbf{v} \leq \mathbf{v}^T L_A \mathbf{v}$.

Also after $T = dn$ steps,

$$\begin{aligned} \frac{\lambda_{max}(A_T)}{\lambda_{min}(A_T)} &\leq \frac{n/\epsilon_u + dn\delta_u}{-n/\epsilon_l + dn\delta_l} \\ &\leq \frac{(\sqrt{d}+1)^2}{(\sqrt{d}-1)^2} \end{aligned}$$

The following algorithm summarises the process represented and proved above.

```
Data: Vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$
Result: Sparsification values $s_i$ for all $i \in [m]$
$A \leftarrow 0$  $j \leftarrow 0$  $s_i \leftarrow 0$ for all $i \in [m]$  while $j < dn$ do
    Compute matrices $U_A$ and $L_A$  for $i \in [m]$ do
        if $\mathbf{v}_i^T U_A \mathbf{v}_i \leq \mathbf{v}_i^T L_A \mathbf{v}_i$ then
            $k \leftarrow i$
        end
    end
    $s_k \leftarrow 1/(\mathbf{v}_k^T L_A \mathbf{v}_k)$  $A \leftarrow A + s\mathbf{v}_k\mathbf{v}_k^T$
end
```

To computationally find such a **v** in each iteration, we need to compute U_A and L_A in each iteration. This can be done in $\mathcal{O}(n^3)$ time. Then we need to calculate $\mathbf{v}_i^T U_A \mathbf{v}_i$ and $\mathbf{v}_i^T L_A \mathbf{v}_i$ for all $i \in [m]$, which can be done in $\mathcal{O}(n^2 m)$ time. The total number of iterations is dn. Hence the total time taken by this algorithm is $\mathcal{O}(dn^3 m)$.

The important feature of the algorithm is that at every iteration, we try to extend the expected behaviour. The analysis of the expected scenario sheds light on the bound that can be achieved and the means to do the same, by the application of barrier functions. Therefore, the algorithm in itself, can be considered an extension of the analysis of the associated Laguerre polynomial, ensuring that the properties derived there, are satisfied while choosing an appropriate vector in each iteration.

Chapter 4

Laplacian Solver using Preconditioners

4.1 Overview

The line of work leading upto the Laplacian solver and the corresponding genealogy of the research based on the techniques used in finding a nearly-linear time Laplacian solver has led to significant developments in spectral graph theory and it's associated fields. One such development was the concept of spectral sparsification, and we will exhibit the importance of such sparsifiers in this section.

We will look at the construction of the Laplacian solver proposed by Spielman and Teng [ST08], and further improved upon and simplified by Koutis, Miller and Peng [KMP11].

4.2 Preconditioners

One of the most general techniques of solving a linear system of equations $A\mathbf{x} = \mathbf{b}$, where A is a symmetric positive-definite matrix, is the **Conjugate Gradient method**. The following is the formal statement the algorithm :

Theorem 4.2.1 (Conjugate Gradient Method [Vis13]). For a symmetric positive definite matrix A, a vector $\mathbf{b}$ and $\epsilon > 0$, after $t = \mathcal{O}(\sqrt{\kappa(A)} \log \frac{1}{\epsilon})$ iterations, the Conjugate Gradient algorithm finds a vector $\mathbf{x}$, such that

$$||\mathbf{x_t} - A^{+}\mathbf{b}||_A \leq \epsilon ||\mathbf{x}_0 - A^{+}\mathbf{b}||_A$$

where $A\mathbf{x}^* = \mathbf{b}$ and $\kappa(A)$ is the condition number of A.

Each iteration of the algorithm takes $\mathcal{O}(t_A)$ time, where t_A is the time taken to multiply a vector to A.

The Conjugate Gradient method was first published in [HS52] and for a detailed analysis of the algorithm, we refer the reader to the following book [BV04].

4.2.1 Condition number of a matrix

The condition number of a matrix is defined as

$$\kappa(A) = ||A|| \cdot ||A^{+}||$$

Given a system $A\mathbf{x} = \mathbf{b}$, condition number is a quantitative measure of the maximum relative error in $\mathbf{x}$ with respect to the relative error in $\mathbf{b}$.

The above definition of condition number does depend on the particular matrix norm used. However, since most such norms are equivalent to each other within a constant, the condition number changes by atmost a constant on changing the norm.

If A is *normal*, then one can verify that

$$\kappa(A) = \frac{|\lambda_{max}(A)|}{|\lambda_{min}(A)|}$$

The complexity of the Conjugate gradient method depends crucially on the condition number of the system, and for an ill-conditioned matrix, directly applying the algorithm does not work.

Instead if we can find a matrix B such that the condition number of B^+A is small and B is easy to invert, then the Conjugate Gradient method can be used to efficiently solve the system $B^+A\mathbf{x} = B^+\mathbf{b}$. Such a matrix B is called a **preconditioner**.

4.2.2 Laplacian Preconditioner

There is no predefined method for finding a good preconditioner, it always depends on the system in consideration. For a Laplacian system however, we saw how to spectrally sparsify a graph G, to get a subgraph H such that

$$(1+\epsilon)^{-1}L_G \preceq L_H \preceq (1+\epsilon)L_G$$

L_H works as good preconditioner for the system $L_G\mathbf{x} = \mathbf{b}$ since

$$(1+\epsilon)^{-1}\mathbf{I} \preceq L_H^+ L_G \preceq (1+\epsilon)\mathbf{I}$$

We will use the sparsifier given by Corollary 2.4.1 of the main sparsification result. Note that we cannot use the main theorem because it requires us to solve Laplacian systems to get the probability distribution.

Theorem 4.2.2 (Stretch sparsifiers). Given a graph G with positive edge weights and for any $\gamma < m$ and $\beta > 0$, we can compute a subgraph H of G with
$n - 1 + \tilde{\mathcal{O}}(\frac{m \log^2 n}{\gamma})$ [1]edges such that

$$L_G \preceq 2L_H \preceq 3\gamma L_G$$

with probability atleast $1 - \beta$.
The algorithm takes $\tilde{\mathcal{O}}((m \log n + n \log^2 n + m \log^2 n/\gamma) \log \frac{1}{\beta})$ time.

Using graph H from the above theorem gives us a suitable preconditioner for our problem. Note that $\kappa(L_H^+ L_G) = \mathcal{O}(\gamma)$.

[1]A factor of $\log(1/\beta)$ will be implicit in these quantities. We hide it to make the proof tidier.

While applying Conjugate Gradient method to the preconditioned system $L_H^+ L_G \mathbf{x} = L_H^+ \mathbf{b}$, we will have to make $\mathcal{O}(\sqrt{\kappa(A)} \log \frac{1}{\epsilon}) = \mathcal{O}(\sqrt{\gamma} \log \frac{1}{\epsilon})$ queries of the form $L_H^+ L_G \mathbf{x} = \mathbf{b}'$. $L_G \mathbf{x}$ can be computed in $\mathcal{O}(m)$ time. So the problem reduces to solving $L_H^+ \mathbf{x}' = \mathbf{b}'$ for multiple $\mathbf{b}'$.

Note that the size of H is much smaller compared to G. We will further reduce the number of edges in H, and then apply the same process recursively on the resulting graph.

4.3 Greedy Elimination using Cholesky Decomposition

4.3.1 Cholesky Decomposition

A symmetric $n \times n$ PSD matrix A can be decomposed into the following form : $A = PP^T$, where P is a lower triangular matrix. This is known as the *Cholesky Decomposition* of A.

If we can find the Cholesky decomposition of A, then solving $A\mathbf{x} = \mathbf{b}$ becomes relatively easier. This is because given the Cholesky decomposition, $A = PP^T$, we can first solve for z in $Pz = b$ and then for x in $P^T x = z$.

To solve for $P\mathbf{z} = \mathbf{b}$, we exploit the property that P is lower triangular.

$$P\mathbf{z} = \begin{bmatrix} p_{11} & 0 & 0 & \dots & 0 \\ p_{21} & p_{22} & 0 & \dots & 0 \\ . & & & & \\ . & & & & \\ p_{n1} & p_{n2} & p_{n3} & \dots & p_{nn} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ z_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}$$

This is equivalent to the system

$$\begin{aligned} p_{11}z_1 &= b_1 \\ p_{21}z_1 + p_{22}z_2 &= b_2 \\ &\cdots \\ p_{n1}z_1 + p_{n2}z_2 \cdots + p_{nn}z_n &= b_n \end{aligned}$$

Solving the first equation gives us z_1. Using the value of z_1 in second equation gives us z_2 and so on. This way we can easily compute the vector $\mathbf{z}$. We further solve $P^T\mathbf{x} = \mathbf{z}$ similarly. The overall time taken to compute $\mathbf{x}$ is proportional to the number of non-zero entries in P.

4.3.2 Decomposition using Schur's complement

A. George [Geo73] showed that Cholesky decomposition of a PSD matrix can be used to solve Laplacian systems for certain special graphs, including square grids and trees. Inspired from their construction, we will give a brief idea about how one can use Schur's decomposition to effectively remove redundant vertices from our current sparsified graph H.

Theorem 4.3.1 (Schur's lemma). Say we are given an $n \times n$ real symmetric matrix A. We can write A as

$$A = \begin{bmatrix} d & u^T \\ u & B \end{bmatrix}$$

Then, $A \succ 0$ if and only if $d > 0$ and $B - uu^t/d \succ 0$.

Using the above representation, we can write A as,

$$A = \begin{bmatrix} d & u^T \\ u & B \end{bmatrix} = \begin{bmatrix} 1 & 0^T \\ u/d & I_{n-1} \end{bmatrix} \begin{bmatrix} d & 0^T \\ 0 & B - uu^T/d \end{bmatrix} \begin{bmatrix} 1 & u^T/d \\ 0 & I_{n-1} \end{bmatrix}$$

Let us denote $A_1 := \begin{bmatrix} d & 0^T \\ 0 & B - uu^T/d \end{bmatrix}$

From Theorem 4.3.1, we know that the matrix $B_1 := B - uu^T/d$ of size $n-1 \times n-1$ is also positive definite. Hence can recursively compute the decomposition of this matrix as well, in a similar fashion as above, till we get a 2 x 2 matrix, for which the above representation is a valid decomposition.

Now say we have the decomposition of $B_1 = P'DP'^T$ (recursively computed). Then we can write A_1 as

$$A_1 = \begin{bmatrix} 1 & 0^T \\ 0 & P' \end{bmatrix} \begin{bmatrix} d & 0^T \\ 0 & D \end{bmatrix} \begin{bmatrix} 1 & 0^T \\ 0 & P'^T \end{bmatrix}$$

And correspondingly A becomes,

$$A = \begin{bmatrix} 1 & 0^T \\ u/d & P' \end{bmatrix} \begin{bmatrix} d & 0^T \\ 0 & D \end{bmatrix} \begin{bmatrix} 1 & u^T/d \\ 0 & P'^T \end{bmatrix}$$

Hence we can recursively compute the Cholesky decomposition of A.

At each step if there exists such a row u which has one or two (or a small constant number of) non-zero entries, then the final triangular matrix P will have only $\mathcal{O}(n)$ non-zero entries. In such a case, the solution to $PDP^T\mathbf{x} = \mathbf{b}$ can be found in $\mathcal{O}(n)$ time.

4.3.3 Decomposing the Laplacian system

Attempting to employ the decomposition to Laplacian system reduces the problem to finding a *vertex elimination ordering* $v_1, \ldots, v_n$, such that for all i, v_i has very few neighbours amongst v_{i+1} to v_n [preferably one or two]. The problem of finding the most optimal ordering has been proven to be NP-hard [Yan80].

However, we can use the above method to remove degree 1 and degree 2 vertices from our graph, by doing a partial decompostion. Lets say we have p degree 1 and q degree 2 vertices, then we can execute the above recursive procedure till we reach a point where we have no such vertices present in our graph. We will call this new graph G'.

Note that H has $n-1+\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$ edges. For simplicity, lets call
$k := \tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$

Now in G', every vertex has degree atleast 3 and total number of edges is $n-1+k-(p+q)$ [2]. Therefore,

$$3(n-p-q) \leq 2(n-1+k-p-q) \Rightarrow \quad (n-p-q) \leq 2(k-1)$$

Hence, the number of vertices in G' is $2(k-1)$ and correspondingly number of edges is $3(k-1)$.

If $\mathbf{x} \in \mathbb{R}^{|V(G)|}$ is the solution to $L_G\mathbf{x} = \mathbf{b}$, then $\mathbf{x}$ can be computed as $[\mathbf{z}\ \mathbf{y}]$, where $\mathbf{z}$ corresponds to the eliminated vertices, and can be computed using the partial decomposition above [3]; and $\mathbf{y}, \mathbf{c} \in \mathbb{R}^{|V(G')|}$ is the solution to the system $L_{G'}\mathbf{y} = \mathbf{c}$. Therefore we have now reduced our problem to this new system with $\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$ vertices and $\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$ edges.

The entire elimination procedure involves removing degree 1 and 2 vertices and computation of their corresponding solutions, and can be done in
$\mathcal{O}(m+n)$ time.

4.4 Recursive Preconditioning and Complexity Analysis

After Cholesky decomposition, the new graph G' has $\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$ vertices and $\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})$ edges. We will recursively apply the preconditioning process and the greedy elimination on G' and continue till the number of vertices goes beyond certain threshold value n_t. The following figure gives a visual representation of the process.

[2]Removing degree 2 vertex removes 2 edges and adds 1, because of the term $B - uu^T/d$
[3]i-the step of the decompostion gives us x_i.

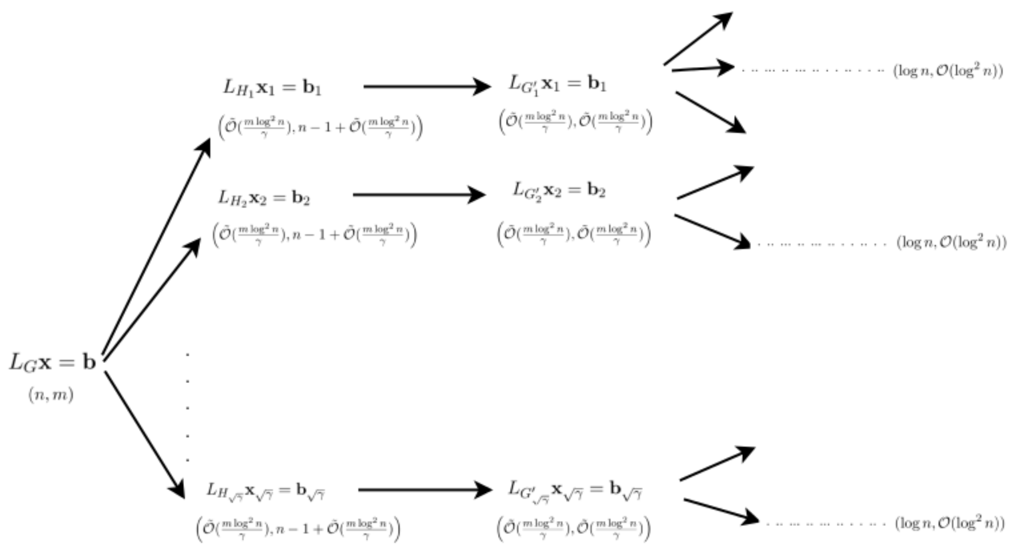
$L_G\mathbf{x} = \mathbf{b}$
(n, m)
$L_{H_1}\mathbf{x}_1 = \mathbf{b}_1$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), n-1+\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$L_{G'_1}\mathbf{x}_1 = \mathbf{b}_1$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), \tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$(\log n, \mathcal{O}(\log^2 n))$
$L_{H_2}\mathbf{x}_2 = \mathbf{b}_2$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), n-1+\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$L_{G'_2}\mathbf{x}_2 = \mathbf{b}_2$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), \tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$(\log n, \mathcal{O}(\log^2 n))$
$L_{H_{\sqrt{\gamma}}}\mathbf{x}_{\sqrt{\gamma}} = \mathbf{b}_{\sqrt{\gamma}}$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), n-1+\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$L_{G'_{\sqrt{\gamma}}}\mathbf{x}_{\sqrt{\gamma}} = \mathbf{b}_{\sqrt{\gamma}}$
$\big(\tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma}), \tilde{\mathcal{O}}(\frac{m\log^2 n}{\gamma})\big)$
$(\log n, \mathcal{O}(\log^2 n))$

FIGURE 4.1: Recursion tree of the Laplacian solver. The quantity in brackets denote the number of verties and edges.

The recursive procedure can be visualized as a tree with the original Laplacian system being the root node. Each internal node makes $\mathcal{O}(\sqrt{\gamma})$ queries of the form $L_{G'}\mathbf{x} = \mathbf{b}$. Say the height of the tree is d. The graph corresponding to the leaf nodes will have n_t nodes. We will use the Gaussian elimination method to solve the Laplacian system for these graphs. Each such system will take $\mathcal{O}(n_t^3)$ time.

Now that the general algorithm has been described, we can calculate the time required in the process of finding the solution to the system and show that setting $\gamma = \mathcal{O}(log^4 n)$ and $n_t = \log n$ gives us a nearly-linear time solver.

Since Gaussian elimination is used to solve the systems corresponding to the leaf nodes, time required at the leaf nodes will be $\mathcal{O}((\sqrt{\gamma})^d n_t^3)$. For $\gamma = \mathcal{O}(log^4 n)$ and $n_t = \log n$,

$$d = \frac{\log n}{\log \log n}$$

At each level, we will have to find the spectral sparsifier of the graph corresponding to the previous level. The total time taken in finding all sparsifiers is

$$\sum_{i=1}^{d} \tilde{\mathcal{O}}((m_i \log n_i + n_i \log^2 n_i + m_i \log^2 n_i/\gamma)$$

where (n_i, m_i) are number of vertices and edges at i-th level.

Taking $\gamma = \mathcal{O}(log^4 n)$, it can be seen that m_i decreases geometrically w.r.t to i and so the above runtime will converge to $\mathcal{O}(m \log n + n \log^2 n)$.

Furthermore, at each internal node, we will have to perform the greedy elimination step. This will take time $\mathcal{O}(m_i + n_i)$. The total time taken for all internal nodes will be

$$\sum_{i=1}^{d} (\sqrt{\gamma})^i \mathcal{O}(m_i + n_i)$$

Again for $\gamma = \mathcal{O}(log^4 n)$, this sum converges to $\mathcal{O}(m)$.

Therefore the total time taken to get ϵ-close to the solution of the Laplacian system is $\tilde{\mathcal{O}}(m \log \frac{1}{\epsilon})$.

Chapter 5

Laplacian Solver using Randomized Kaczmarz Method

5.1 Overview

In 2013, Kelner et. al. [KOSZ13] showed that the general iterative technique of Randomized Kaczmarz can be used to construct a nearly-linear time Laplacian solver, which is much simpler compared to the solver based on spectral sparsification.

This solver aims to compute the electric current in the circuit corresponding to the Laplacian system $L\mathbf{x} = \mathbf{b}$, and use it to find the voltage vector $\mathbf{x}$. In contrast, sparsification based solvers used iterative techniques like Conjugate Gradient method to find $\mathbf{x}$ directly.

In this chapter we give the details of this Laplacian solver based on Randomized Kaczmarz method, outlining the algorithm used and the complexity achieved. We then suggest certain extensions of the method employed and their application in solving the Laplacian system.

5.2 Kaczmarz Method

Kaczmarz method and its variants are a set of general iterative techniques for solving a system of linear equations. They were first discovered by S. Kaczmarz [Kac37]. We describe the Kaczmarz method below and present its analysis and application to the Laplacian problem in the later sections.

Given a system of linear equations $Ax = b$, we try to find the solution of the system by finding the point in the solution space which satisfies $\forall i, \langle a_i, x\rangle = b_i$, where a_i's are the rows of A and b_i are the elements of vector b. The notation h_i will denote the hyperplane $\langle a_i, x\rangle = b_i$.

In Kaczmarz method, we start with some suitably chosen point x_0, and at each iteration, try to satisfy one of the hyperplanes. Say x_t is the point we have at the end of t-th iteration, and it does not satisfy some hyperplane $h_j :: \langle a_j, x\rangle = b_j$. Then we take the orthogonal projection of x_t on h_j to get x_{t+1}.

In the case of simple Kaczmarz method, the hyperplane chosen in t-th iteration is $\langle a_k, x\rangle = b_k$, where $k = t \pmod m + 1$. However apart from some special cases, the convergence bound of simple Kaczmarz method for general systems is not known.

Another variant is the Randomized Kaczmarz method, developed by Strohmer and Vershayin [SV09a], where we associate a probability distribution with the set of hyperplanes, and sample from this distribution at each iteration of the algorithm.

5.3 Randomized Kaczmarz Method

In the Randomized Kaczmarz method, at each iteration we sample a hyperplane from the given probability distribution on the set of hyperplanes. The considered probability distribution is the following :

Each hyperplane is chosen with probability proportional to the square of the norm of its coefficient vector, i.e., hyperplane $h_i : \langle a_i, x\rangle = b_i$ is chosen with probability proportional to $||a_i||^2$.

We will prove the following convergence theorem for the above described Randomized Kaczmarz method.

Theorem 5.3.1 ([Vis13]). For the given system of equations $Ax = b$ with solution x^*, we can get

$$||x_t - x^*|| \leq \epsilon ||x_0 - x^*||$$

for $t = \mathcal{O}(\kappa^2(A) \log \frac{1}{\epsilon})$ with probability 0.9, using Randomized Kaczmarz method.

The Conjugate Gradient method, in comparison, converges in $\sqrt{\kappa(A)}$ iterations, with each iteration taking time proportional to time taken to multiply a vector to A. Randomized Kaczmarz method takes more number of iterations to converge, but the time taken per iteration is the time taken to compute the projection of a vector on a hyperplane. We will see how this characterization is useful in our system.

To prove the above theorem we will first prove the following lemma. The proof of the theorem will follow from the lemma by a simple application of Markov's inequality.

Lemma 5.3.1. In the $t + 1$-th step of the Randomized Kaczmarz method,

$$\mathbb{E}||x_{t+1} - x^*||^2 \leq \left(1 - \frac{1}{\kappa^2(A)}\right) ||x_t - x^*||^2$$

where expectation is taken over the random choice in the $t + 1$-th iteration.

Proof. Say at the $(t + 1)$-th iteration, we randomly chose the hyperplane $\langle a_i, x\rangle = b_i$. The unit normal to this hyperplane is $\frac{a_i}{||a_i||}$. Then for some c,

$$x_{t+1} = x_t + c \cdot \frac{a_i}{||a_i||}$$

$$\begin{aligned} \Rightarrow \quad \langle a_i, x^*\rangle &= \langle a_i, x_{t+1}\rangle \\ &= \langle a_i, x_t\rangle + c \cdot \frac{\langle a_i, a_i\rangle}{||a_i||} \\ &= \langle a_i, x_t\rangle + c \cdot ||a_i|| \end{aligned}$$

Therefore,

$$c = \frac{\langle a_i, x^* - x_t \rangle}{||a_i||}$$

and,

$$x_{t+1} - x_t = \frac{\langle a_i, x^* - x_t \rangle}{||a_i||^2}.a_i \tag{5.1}$$

$$\begin{aligned}
||x_{t+1} - x^*||^2 &= ||x_t - x^* + \frac{\langle a_i, x^* - x_t \rangle}{||a_i||^2}.a_i||^2 \\
&= ||x_t - x^*||^2 - \frac{\langle a_i, x^* - x_t \rangle^2}{||a_i||^4}||a_i||^2 \\
&= ||x_t - x^*||^2 - \frac{\langle a_i, x^* - x_t \rangle^2}{||a_i||^2}
\end{aligned}$$

We now look at the expected value of the LHS with respect to the choice of hyperplane in the $(t+1)$-th iteration. Note that since the iterations upto t are fixed, $||x_t - x^*||^2$ is not affected by taking expectation.

$$\begin{aligned}
\mathbb{E}||x_{t+1} - x^*||^2 &= ||x_t - x^*||^2 - \sum_{i=1}^{m} \frac{||a_i||^2}{\sum_i ||a_i||^2} \frac{\langle a_i, x^* - x_t \rangle^2}{||a_i||^2} \\
&= ||x_t - x^*||^2 - \sum_{i=1}^{m} \frac{\langle a_i, x^* - x_t \rangle^2}{||A||_F^2} \\
&= ||x_t - x^*||^2 (1 - \frac{1}{||A||_F^2} \sum_{i=1}^{m} \frac{\langle a_i, x^* - x_t \rangle^2}{||x^* - x_t||^2}) \\
&= ||x_t - x^*||^2 (1 - \frac{1}{||A||_F^2} \frac{||A(x^* - x_t)||^2}{||x^* - x_t||^2}) \\
&\leq ||x_t - x^*||^2 (1 - \frac{1}{||A||_F^2 ||A^+||^2}) \\
&\leq (1 - \frac{1}{\kappa^2(A)})||x_t - x^*||^2
\end{aligned}$$

□

Proof of Theorem 5.3.1. Using Lemma 5.3.1, after $T = \kappa^2(A)\log\frac{10}{\epsilon}$ iterations, we can say

$$\begin{aligned}\mathbb{E}||x_T - x^*||^2 &\leq \left(1 - \frac{1}{\kappa^2(A)}\right)^T ||x_0 - x^*||^2 \\ &\leq \frac{\epsilon}{10}||x_0 - x^*||^2\end{aligned}$$

Let Y be the random variable $||x_T - x^*||^2$. By Markov's inequality, the probability that Y is greater than $\epsilon^2||x_0 - x^*||^2$ is

$$\begin{aligned}Pr[Y \geq \epsilon^2||x_0 - x^*||^2] &\leq \frac{\mathbb{E}[Y]}{\epsilon^2||x_0 - x^*||^2} \\ &\leq \frac{1}{10}\end{aligned}$$

Therefore,

$$Pr[Y \leq \epsilon^2||x_0 - x^*||^2] \geq 0.9$$

□

5.4 Equivalent formulation of Laplacian system

In 2013, Kelner et al. [KOSZ13] developed a combinatorial algorithm to solve Laplacian systems, which also had a geometric interpretation in terms of Randomized Kaczmarz method. We state and analyze the algorithm using Randomized Kaczmarz method in this section.

We will derive an equivalent formulation of the Laplacian system of equations, and apply Randomized Kaczmarz method on it to get an approximate solution to our problem.

Given the system $Lx = b$ and the corresponding graph $G = (V, E)$, we look at the electrical circuit with the same structure as this graph and resistance of 1 ohm on the edges. Fix an arbitrary orientation of the edges in E, which will denote the flow of current in these edges. Let the matrix $B \in \mathbb{R}^{m\times n}$ be the orientation matrix for this circuit, with $B[e, i] = +1$, if vertex i is the tail of edge e and -1, if it is the head.

Then it is easy to see that $B^T B = L$.

Let $i \in \mathbb{R}^m$ be the vector of current in edges and $v \in \mathbb{R}^n$ be the vector of voltages in the vertices. Vector b will denote the external current in the vertices. By the Kirchoff's law of current conservation, $B^T i = b$.

Also by Ohm's law, $Bv = i$.

$$\Rightarrow \quad B^T BV = B^T i$$

$$\Rightarrow \quad Lv = b$$

Therefore, finding the voltage in this circuit will give us the solution to the Laplacian system of equations, and vice-versa.

On careful consideration, one can also realise that just finding the current in each edge of this circuit also suffices. This is because for each edge (p, q), $v_q - v_p = i_{q,p}$. Given the current in each of the edges, one can take some vertex to have voltage 0, and, relative to that, calculate the voltages in other vertices using the given set of linear equations in two variables of voltages, in $\mathcal{O}(m)$ time. So we will focus on finding the current vector.

For a vector $f \in \mathbb{R}^m$ to be a valid flow, it has to satisfy Kirchoff's conservation law, i.e., $B^T f = b$.

Also it must satisfy the Ohm's law. One way to express this constraint would be to say that for all cycles, potential drop across a cycle must be zero. Let $\mathbf{1}_C \in \mathbb{R}^m$ be the indicator vector for cycle C. Then f must satisfy $\langle \mathbf{1}_C, f \rangle = 0$, for all cycles C in G.

Since there can exponential number of cycles in the graph, we need to capture the second constraint by some subset of cycles. To this end, we consider the vector space spanned by the indicator vectors of the cycles of the graph (reffered to as cycle space) taking the addition operation to be the component-wise sum of vectors, mod 2. If we can efficiently find a small basis for this vector space, then we can use it for the new system of equations.

5.4.1 Basis of cycle space of graph G

Take a spanning tree T of graph G. It contains $n-1$ edges. Adding one non-tree eked to this tree will result in a cycle. Let $\mathbf{1}_{C_e} \in \mathbb{R}^m$ be the indicator vector for the cycle formed when e is inserted in T. We will show that $\mathbf{1}_{C_e}$'s form a cycle space.

Lemma 5.4.1. For all $e \in E/T$, the cycle formed by adding e to T, $\mathbf{1}_{C_e}$, is unique and the set of all such cycles $\{\mathbf{1}_{C_e}\}_{e \in E/T}$ is a basis for the cycle space.

Proof. Before proving the above statement, we will first show that linear combination of two cycle vectors results in a vector which again represents a cycle.

Take cycle vectors $\mathbf{1}_{C_a}$ and $\mathbf{1}_{C_b}$. If C_a and C_b have any common edge e, the corresponding components' addition mod 2 will be 0. Then $\mathbf{1}_{C_a} + \mathbf{1}_{C_b}$ will have 0 for the component corresponding to e and 1 for the components corresponding to edges neighbouring to e in $\mathbf{1}_{C_a}$ and $\mathbf{1}_{C_b}$. So the sum will again be a cycle, albeit a different one.

Next we will prove that for any cycle C which does not belong to the set, $\{\mathbf{1}_{C_e} : e \in E - T\}$, $\mathbf{1}_C$ is a linear combination of some elements in this set.

Say C contains non-tree edges $e_1, e_2, ...e_k$. Then

$$\mathbf{1}_C + \mathbf{1}_{C_{e_1}} + ... + \mathbf{1}_{C_{e_k}}$$

will have 0 in all components corresponding to non-tree edges. Also, by our earlier proof, this sum will correspond to some cycle in the graph. So the sum corresponds to a cycle composed only of tree edges. There is only one such cycle, which is the empty cycle. Therefore,

$$\mathbf{1}_C + \mathbf{1}_{C_{e_1}} + ... + \mathbf{1}_{C_{e_k}} = \mathbf{0}$$

Hence, $\mathbf{1}_C$ is a linear combination of $\mathbf{1}_{C_{e_1}}, ..., \mathbf{1}_{C_{e_k}}$.

So $\{\mathbf{1}_{C_e} : e \in E - T\}$ is a valid basis for the cycle space. $\square$

From the above analysis, we now have the following system of equations whose solution is the current vector in the circuit formed from graph G.

$$\langle \mathbf{1}_{C_e}, f \rangle = 0 \ \forall e \in E/T, \text{ such that } B^T f = b \tag{1}$$

Given the above system of equations, we solve it using the Randomized Kaczmarz method stated earlier. However, we need to start with an appropriate f_0 which satisfies $Bf_0 = b$. This is the necessary and sufficient condition to ensure $Bf_t = b$ for all iterations t, because

$$\begin{aligned} & f_{t+1} = f_t + c \cdot \mathbf{1}_{C_e} \\ \Rightarrow \quad & Bf_{t+1} = Bf_t + c \cdot B\mathbf{1}_{C_e} \end{aligned}$$

But we know that potential drop across a cycle is zero, i.e., $B\mathbf{1}_{C_e} = 0$. So if $Bf_0 = b$, then $Bf_t = b$ for all iterations t.

5.4.2 Choice of f_0

We need a vector which is a valid flow for the given circuit. To find this, we take a spanning tree of the graph, and starting from the leaf edges, we assign a flow to each edge so as to satisfy the flow conservation on the parent node, moving from bottom to top. Since the overall external flow is zero, flow conservation will be satisfied on all nodes till the root.

We can now apply Randomized Kaczmarz method on 1 using the above specified f_0, and obtain an approximate current vector.

The only thing left to analyze is the number of iterations, or in particular, the condition number of this system of equations.

5.4.3 Condition number of the system

Let A denote the matrix with $\mathbf{1}_{C_e}$ as its rows. We know that $\kappa(A)^2 = ||A^+||^2||A||_F^2$.

Lemma 5.4.2. $||Af|| \geq ||f||$ if $Bf = 0$

Proof. A is an $m-n+1 \times m$ matrix. Assume w.l.o.g., that the first $n-1$ columns of A correspond to tree-edges. Arrange the non-tree edges part of A to form an identity matrix. This again can be done without affecting the system. So A looks like,

$$\begin{bmatrix} A_T & I_N \end{bmatrix}$$

with A_T and I_N corresponding to tree and non-tree edges respectively.

Similarly we divide f into two vectors f_T and f_N, corresponding to tree and non-tree edges respectively. We will first prove that $A_T^T f_N = f_T$.

This is true because flow in a non-tree edge e is inherent to the cycle C_e, since e is not present in any other cycle. This is the flow which is present because of circulation in cycle C_e. So to find the flow in a tree edge, one just needs to sum up the flows in each of the cycles in which this tree edge is present, which corresponds to the sum of flows in the corresponding non-tree edges which form this cycle. This is exactly what the above equality represents.

Therefore,

$$||Af||^2 = ||A_T f_T + I_N f_N||^2 = ||A_T f_T||^2 + ||f_N||^2 + 2f_T^T A_T^T f_N$$

$$= ||A_T f_T||^2 + ||f_N||^2 + 2||f_T||^2 \geq ||f_N||^2 + ||f_T||^2 = ||f||^2$$

□

We know that $||A^+|| = \sup_{f\in\mathbb{R}^m} \frac{||f||}{||Af||}$. But since by our analysis of Randomized Kaczmarz method we know that we only need to consider vectors of the form $f_1 - f_2$, where both f_1 and f_2 are flow vectors, the following definition of $||A^+||$ suffices.

$$||A^+|| = \sup_{f \in \mathbb{R}^m, Bf=0} \frac{||f||}{||Af||}$$

Therefore from the above lemma, we can infer $||A^+|| \leq 1$.

We now look at $||A||_F$.

$$||A||_F = \sum_{i,j} a_{i,j}^2 = m - n + 1 + \sum_{e \in E/T} \mathrm{str}_T(e)$$

where, $\mathrm{str}_T(e)$ is the length of path between the endpoints of e in T, also called the stretch of e. Also, $\mathrm{str}_T(G) = \sum_{e \in E/T} \mathrm{str}_T(e)$

So the problem has now reduced to finding a spanning tree of the graph which minimizes the overall stretch. In this regard, we will again use the well known result by Elkin et al. [EEST08] stated below.

Theorem 5.4.1 ([EEST08]). For any undirected graph G, a spanning tree T can be constructed in $\widetilde{\mathcal{O}}(m \log n + n \log n \log \log n)$ such that the stretch corresponding to T in G is $\widetilde{\mathcal{O}}(m \log n)$.

From the above theorem, we can see that $||A||_F = \widetilde{\mathcal{O}}(m)$. Therefore, the number of iterations in the algorithm is $\widetilde{\mathcal{O}}(m)$.

Each iteration involves finding the projection on the sampled hyperplane, which in our case is addition of relevant amount of flow to one of the cycles in the graph. This can be done in $\mathcal{O}(\log n)$ time using *link-cut data structure.*

Therefore, the overall time complexity of this Laplacian solver is $\widetilde{\mathcal{O}}(m)$.

5.5 New directions - Randomized Kaczmarz methods based on Dihedral Angles

We explore possible improvements and propose better sampling procedures in the Randomized Kaczmarz algorithm or its applications to the Laplacian system.

Firstly, we will take another look at the geometric interpretation of the algorithm. Say we have the system of equations $Ax = b$, where A is an $m \times n$ matrix. In each iteration, we randomly select a hyperplane corresponding to a row of A, choosing $\langle a_i, x\rangle = b_i$ with probability $||a_i||^2/||A||_F^2$.

However, notice that if we scale any hyperplane with some constant, we change its probability of selection. Even though $\langle a_i, x\rangle = b_i$ and $c \cdot \langle a_i, x\rangle = c \cdot b_i$ are the same hyperplane, the probability of selection of both are different and depends on c. The change in scaling can in turn affect the number of iterations required for convergence of the algorithm. This has also been noted in the following articles [CHJ09] [SV09b].

To counter this problem, we look into finding a probability distribution which is independent of the scaling of the system. The motivation for this is that the Randomized Kaczmarz method, as described above, is a geometric algorithm. As long the geometric structure remains same, changing the algebraic structure should not have any impact on the algorithm or its complexity.

We will base our probability distribution on the dihedral angle between hyperplanes. There are two methods for this, which are listed below.

5.5.1 Method 1

Let $h :: \langle \alpha, x\rangle = \beta$ be some appropriately chosen hyperplane in the given space. The dihedral angle between h and h_i is defined as

$$D_{h,h_i} := \langle \hat{\alpha}, \hat{a}_i\rangle$$

We define probability of selecting h_i as

$$p_i = \frac{D^2_{h,h_i}}{\sum_i D^2_{h,h_i}} = \frac{\langle \hat{\alpha}, \hat{a}_i \rangle^2}{\sum_i \langle \hat{\alpha}, \hat{a}_i \rangle^2}$$

We now use this probability distribution in the computation of

$\mathbb{E}||x_{t+1} - x^*||^2$. Therefore,

$$\begin{aligned}
\mathbb{E}||x_{t+1} - x^*||^2 &= ||x_t - x^*||^2 - \sum_{i=1}^{m} \frac{\langle \hat{\alpha}, \hat{a}_i \rangle^2}{\sum_j \langle \hat{\alpha}, \hat{a}_j \rangle^2} \langle \hat{a}_i, x^* - x_t \rangle^2 \\
&= ||x_t - x^*||^2 \left(1 - \frac{1}{\sum_j \langle \hat{\alpha}, \hat{a}_j \rangle^2} \sum_{i=1}^{m} \langle \hat{\alpha}, \hat{a}_i \rangle^2 \frac{\langle \hat{a}_i, x^* - x_t \rangle^2}{||x_t - x^*||^2} \right) \\
&= ||x_t - x^*||^2 \left(1 - \frac{1}{K} \sum_{i=1}^{m} \frac{\langle \alpha, \hat{a}_i \rangle^2 \langle \hat{a}_i, x^* - x_t \rangle^2}{||\alpha||^2 ||x_t - x^*||^2} \right)
\end{aligned}$$

where $K = \sum_j \langle \hat{\alpha}, \hat{a}_j \rangle^2 \leq m$ (number of rows of A).

Also,

$$\begin{aligned}
\sum_i \frac{\langle \alpha, \hat{a}_i \rangle^2 \langle \hat{a}_i, x^* - x_t \rangle^2}{||\alpha||^2 ||x_t - x^*||^2} &= \frac{\langle \hat{a}_i \otimes \hat{a}_i, \alpha \otimes (x^* - x_t) \rangle^2}{||\alpha \otimes (x^* - x_t)||^2} \\
&= \frac{||A_{\otimes}.(\alpha \otimes (x^* - x_t))||^2}{||\alpha \otimes (x^* - x_t)||^2}
\end{aligned}$$

where $A_{\otimes}$ is the matrix whose i-th row is $\hat{a}_i \otimes \hat{a}_i$.

Therefore,

$$\begin{aligned}
\mathbb{E}||x_{t+1} - x^*||^2 &\leq ||x_t - x^*||^2 (1 - \frac{1}{K \cdot ||A^+_{\otimes}||^2}) \\
&\leq ||x_0 - x^*||^2 (1 - \frac{1}{K \cdot ||A^+_{\otimes}||^2})^t
\end{aligned}$$

So the number of iterations required for this algorithm to get ϵ-close to solution is $\mathcal{O}(K \cdot ||A^+_{\otimes}||^2 \log \frac{1}{\epsilon})$.

The choice of α plays a crucial role in the determining the complexity of this procedure. However, a trivial upper bound, independent of α can be obtained in the following way

$$||A_{\otimes}||_F^2 = \sum_{i=1}^{m} ||\hat{a}_i \otimes \hat{a}_i||^2 = \sum_{i=1}^{m} ||\hat{a}_i||^4 = m$$
$$\Rightarrow \quad K \leq ||A_{\otimes}||_F^2$$

Hence, the number of iterations to get ϵ-close to the solution is $\mathcal{O}(\kappa^2(A_{\otimes}) \log \frac{1}{\epsilon})$.

The complexity bound for the simple Randomized Kaczmarz algorithm and Dihedral angle based algorithm cannot be compared for a general system of equations. However, in certain scenarios, it can be shown that the for a certain choice of α, the dihedral angle based method outperforms the simple Randomized Kaczmarz algorithm.

5.5.1.1 Application of Dihedral Angles based Randomized Kaczmarz algorithm to Laplacian systems

Once again we will use the modified system of equations based on flow inequalities, to get the solution to the Laplacian system on a graph $G = (V, E)$. That is, the linear system in consideration is the following :

$$\langle \mathbf{1}_{C_e}, f \rangle = 0 \; \forall e \in E/T, \text{ such that } B^T f = b$$

where, T is a spanning tree of G and B is the edge-incidence matrix of G.

In this case, the probability of selecting the i-th hyperplane is

$$p_i = \frac{D_{h,h_i}^2}{\sum_i D_{h,h_i}^2} = \frac{\langle \hat{\alpha}, \hat{\mathbf{1}_{C_i}} \rangle^2}{\sum_i \langle \hat{\alpha}, \hat{\mathbf{1}_{C_i}} \rangle^2}$$

Let $\alpha = \mathbf{1}$. Then,

$$\langle \hat{\mathbf{1}}, \hat{\mathbf{1}_{C_i}} \rangle^2 = \langle \frac{\mathbf{1}}{\sqrt{m}}, \frac{\mathbf{1}_{C_i}}{\sqrt{|C_i|}} \rangle^2 = \frac{\langle \mathbf{1}, \mathbf{1}_{C_i} \rangle^2}{m|C_i|} = \frac{|C_i|^2}{m|C_i|} = \frac{|C_i|}{m}$$

Therefore, for this choice of α,

$$p_i = \frac{\langle \hat{\mathbf{1}}, \hat{\mathbf{1}_{C_i}} \rangle^2}{\sum_i \langle \hat{\mathbf{1}}, \hat{\mathbf{1}_{C_i}} \rangle^2} = \frac{|C_i|}{\sum_i |C_i|}$$

which is same as the probability distribution in the simple Randomized Kaczmarz case. Therefore, for $\alpha = \mathbf{1}$, the Dihedral angle based Randomized Kaczmarz yields similar time complexity bounds as simple Randomized Kaczmarz algorithm.

There may exist a choice of α for which the new method performs better than simple Randomized Kaczmarz algorithm. However, the best possible α is dependent on the linear system in consideration and the search for one can be treated as an optimization problem in itself.

5.5.2 Method 2

We look at the analysis of the $t+1$-th iteration of the Randomized Kaczmarz to get better insight into choice of hyperplanes. Specifically, we look at the following step

$$\mathbb{E}||x_{t+1} - x^*||^2 = ||x_t - x^*||^2 - \sum_{i=1}^{m} p_i \langle \hat{a}_i, x^* - x_t \rangle^2$$

where p_i is the probability of choosing the i-th hyperplane.

Say $h_j :: \langle a_j, x \rangle = b_j$ was the hyperplane selected in the t-th iteration. Then

$$\begin{aligned} & \langle a_j, x^* \rangle = \langle a_j, x_t \rangle \\ \Rightarrow \quad & \langle a_j, x^* - x_t \rangle = 0 \end{aligned}$$

$\sum_{i=1}^{m} p_i \langle \hat{a}_i, x^* - x_t \rangle^2$ is a convex combination of $\langle \hat{a}_i, x^* - x_t \rangle^2$, for all i. This will be maximised when $p_k = 1$, where $k = \arg\max_i \langle \hat{a}_i, x^* - x_t \rangle^2$, and all other probabilities are 0.

So in each iteration, we need to find the hyperplane h_i, where
$i = \arg\max_i \langle \hat{a}_i, x^* - x_t \rangle$ given that $\langle \hat{a}_j, x^* - x_t \rangle = 0$.

But we do not know the value of x^*, so we need to relax our system and admit an approximate solution. We can relax it in the following way : find the hyperplane h_i, where
$i = \arg\max_i \max_x \langle \hat{a_i}, x \rangle$ given that $\langle \hat{a_j}, x \rangle = 0$.

So the solution space of x is all those vectors which are orthogonal to $\hat{a_j}$. Now $\langle \hat{a_i}, x \rangle$ will be maximized if $\hat{a_i}$ has a large component in the direction of x and correspondingly small in the direction of a_j.

This problem is equivalent to finding the hyperplane h_i, where
$i = \arg\min_i \langle \hat{a_i}, \hat{a_j} \rangle$.

However, note that this can be interpreted as a deterministic solution, where in each iteration, a hyperplane is chosen as per the condition above. But such a deterministic solution is not desirable. This is because in many cases, it may happen that for any two (or any other small number of) hyperplanes h_i and h_j have the smallest $\langle \hat{a_i}, \hat{a_j} \rangle$, compared to other pairs. Then the choice of hyperplanes will keep cycling between these two and there will be no convergence, since other hyperplanes will remain unsatisfied.

Therefore, we convert the above strategy to a randomized algorithm, with the probability of selecting h_i,

$$p_i = 1 - \frac{\langle \hat{a_i}, \hat{a_j} \rangle^2}{\sum_i \langle \hat{a_i}, \hat{a_j} \rangle^2}$$

where h_j is the hyperplane chosen in the previous iteration.

Wallace and Sekman [WS14], in their paper "Deterministic Versus Randomized Kaczmarz Iterative Projection" propose and analyse such a selection procedure for the hyperplanes. They experimentally show that it works atleast as good as simple Randomized Kaczmarz method and considerably better than deterministic Kaczmarz method. However they fall short of providing a proof for their observation. The above reasoning is an small explanation in concurrence with their experimental results.

Chapter 6

Future Directions

Based on our study, we propose the following interesting directions of work in the field of Spectral Sparsification and Laplacian solvers:

- Showing that Laplacian solvers based on spectral sparsification are equivalent to the Randomized Kaczmarz based Laplacian solver.
- Extending the results of spectral sparsification to more general matrices. For example, in [SHS16], it was shown that the sparsification technique of [SS08] can be extended to sparsify sets of positive-semidefinite matrices of arbitrary rank. Also in [Sri10], the result of [BSS09] was extended to general quadratic forms.
- Finding deterministic sparsification algorithms with better time complexity than the algorithm of [BSS09].
- Empirically comparing the time complexity of convergence of Randomized Kaczmarz method given by Strohmer and Vershayin, and the one based on sampling proportional to Dihedral angles.
- Finding Laplacian solvers independent of sparsifiers and low-stretch spanning trees. A major step in this direction was provided by Kyng and Sachdeva who gave a randomized algorithm to construct sparse Cholesky decompostion of Laplacian matrices [KS16].

Bibliography

[AM85] N. Alon and V. Milman. λ_1, isoperimetric inequalities for graphs and superconductors. *Journal of Combinatorial Theory*, 38:73–88, 1985.

[AW06] R Ahlswede and A Winter. Strong converse for identification via quantum channels. In *Proceedings of IEEE Transactions on Information Theory*, volume 48, pages 569–579, 2006.

[BH12] J. C. A. Barata and M. S. Hussein. The Moore-Penrose Pseudoinverse. A Tutorial Review of the Theory. *Brazillian Journal of Physics*, 42:146–165, 2012.

[BK96] Andras Benczur and David Karger. Approximating s-t minimum cuts in $\tilde{\mathcal{O}}(n^2)$ time. In *Proceedings of the 28th ACM symposium on Theory of computing (STOC)*, pages 47–55, 1996.

[BSS09] Joshua Batson, Daniel Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *Proceedings of the 41st ACM symposium on Theory of computing (STOC)*, page 1704–1721, 2009.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[Che70] Jeff Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in Analysis*, 625:195–199, 1970.

[CHJ09] Y. Censor, G. Herman, and M. Jiang. A note on the behavior of the randomized Kaczmarz algorithm of Strohmer and Vershynin. *Journal of Fourier Analysis and Applications*, 15:431–436, 2009.

[CRR$^+$96] AK Chandra, P Raghavan, WL Ruzzo, R Smolensky, and P Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6:312–340, 1996.

[DZ07] J. Ding and A. Zhou. Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226, 2007.

[EEST08] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[Fie73] M. Fiedler. Algebraic connectivity of graphs. *Czech. Math. Journal*, 23:298–305, 1973.

[Fie10] M. Fiedler. Spectral radius and hamiltonicity of graphs. *Linear Algebra and its Applications*, 432:2170–2173, 2010.

[Geo73] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[GT79] R. Gilbert and R. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[GT86] J. Gilbert and R. Tarjan. The analysis of a nested dissection. *Numerische Mathematik*, 50(4):377–404, 1986.

[Hae] Willem Haemers. Interlacing eigenvalues and graphs. `http://members.upc.nl/w.haemers/interlacing.pdf`.

[HS52] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[Kac37] S. Kaczmarz. Angenherte Au sung von Systemen linearer Gleichungen. *Bulletin International de l'Acadmie Polonaise des Sciences et des Lettres*, pages 355–357, 1937.

[KMP11] Ioannis Koutis, Gary Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 590–598, 2011.

[KMST10] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *Proceedings of the 42nd ACM symposium on Theory of computing (STOC)*, pages 57–66, 2010.

[KOSZ13] J Kelner, L Orecchia, A Sidford, and Z Zhu. A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time. In *Proceedings of the 45th ACM symposium on Theory of computing (STOC)*, pages 911–920, 2013.

[Kra03] Ilia Krasikov. On extreme zeros of classical orthogonal polynomials. 2003. http://arxiv.org/pdf/math/0306286v1.pdf.

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate Gaussian Elimination for Laplacians - Fast, Sparse, and Simple. 2016. http://arxiv.org/pdf/1605.02353v1.pdf.

[LS13] Yin Tat Lee and Aaron Sidford. Efficient Accelerated Coordinate Descent Methods and Faster Algorithms for Solving Linear Systems. In *Proceedings of Foundations of Computer Science (FOCS)*, 2013.

[MSS13] Adam Marcus, Daniel Spielman, and Nikhil Srivastava. Interlacing Families I: Bipartite Ramanujan Graphs of All Degrees. In *Proceedings of Foundations of Computer Science (FOCS)*, 2013.

[MSS14] Adam Marcus, Daniel Spielman, and Nikhil Srivastava. Ramanujan Graphs and the Solution of the Kadison-Singer Problem. In *Proceedings of ICM'14*, 2014.

[MSS15] Adam Marcus, Daniel Spielman, and Nikhil Srivastava. Interlacing Families II: Mixed Characteristic Polynomials and the Kadison-Singer Problem . *Annals of Mathematics*, 182(1):327–350, 2015.

[Rud99] M. Rudelson. Random Vectors in the Isotropic Position. *Journal of Functional Analysis*, 164(1):60–72, 1999.

[SHS16] Marcel K. De Carli Silva, Nicholas J. A. Harvey, and Cristiane M. Sato. Sparse Sums of Positive Semidefinite Matrices. *Journal ACM Transactions on Algorithms (TALG) - Special Issue on SODA'12 and Regular Papers*, 12, 2016.

[SM50] J Sherman and W.J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *Annals of Mathematical Statistics*, 21:124–127, 1950.

[Sri10] N. Srivastava. Spectral sparsification and restricted invertibility, 2010. http://www.cs.yale.edu/homes/srivastava/dissertation.pdf.

[SS08] Daniel Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th ACM symposium on Theory of computing (STOC)*, pages 563–568, 2008.

[ST04] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2004.

[ST08] Daniel Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR, abs/cs/0607105*, 2008.

[ST11] Daniel Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[SV09a] T. Strohmer and R. Vershayin. A randomized Kaczmarz algorithm for linear systems with exponential convergence. *Journal of Fourier Analysis and Applications*, 15:262–278, 2009.

[SV09b] T. Strohmer and R. Vershayin. Comments on the randomized Kaczmarz method. *Journal of Fourier Analysis and Applications*, 15:437–440, 2009.

[Vai91] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners., 1991.

[Vis13] Nisheeth K. Vishnoi. *Lx=b, Laplacian Solvers and Their Algorithmic Applications*. Now, 2013. http://research.microsoft.com/en-us/um/people/nvishno/site/lxb-web.pdf.

[WS14] T. Wallace and A. Sekman. Deterministic Versus Randomized Kaczmarz Iterative Projection. 2014. http://arxiv.org/abs/1407.5593/.

[Yan80] M. Yannakakis. Computing the Minimum Fill-In is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1980.