Foundations and Trends® in
Theoretical Computer Science
Vol. 8, Nos. 1–2 (2012) 1–141

DOI: 10.1561/0400000054

now
the essence of knowledge

$Lx = b$
Laplacian Solvers and Their Algorithmic Applications

By Nisheeth K. Vishnoi

Contents

Foundations and Trends® in
Theoretical Computer Science
Vol. 8, Nos. 1–2 (2012) 1–141

DOI: 10.1561/0400000054

now
the essence of knowledge

$Lx = b$
Laplacian Solvers and Their Algorithmic Applications

Nisheeth K. Vishnoi

Microsoft Research, India, nisheeth.vishnoi@gmail.com

Abstract

The ability to solve a system of linear equations lies at the heart of areas such as optimization, scientific computing, and computer science, and has traditionally been a central topic of research in the area of numerical linear algebra. An important class of instances that arise in practice has the form $L\mathbf{x} = \mathbf{b}$, where L is the Laplacian of an undirected graph. After decades of sustained research and combining tools from disparate areas, we now have Laplacian solvers that run in time nearly-linear in the sparsity (that is, the number of edges in the associated graph) of the system, which is a distant goal for general systems. Surprisingly, and perhaps not the original motivation behind this line of research, Laplacian solvers are impacting the theory of fast algorithms for fundamental graph problems. In this monograph, the emerging paradigm of employing Laplacian solvers to design novel fast algorithms for graph problems is illustrated through a small but carefully chosen set of examples. A part of this monograph is also dedicated to developing the ideas that go into the construction of near-linear-time Laplacian solvers. An understanding of these methods, which marry techniques from linear algebra and graph theory, will not only enrich the tool-set of an algorithm designer but will also provide the ability to adapt these methods to design fast algorithms for other fundamental problems.

Preface

The ability to solve a system of linear equations lies at the heart of areas such as optimization, scientific computing, and computer science and, traditionally, has been a central topic of research in numerical linear algebra. Consider a system $A\mathbf{x} = \mathbf{b}$ with n equations in n variables. Broadly, solvers for such a system of equations fall into two categories. The first is Gaussian elimination-based methods which, essentially, can be made to run in the time it takes to multiply two $n \times n$ matrices, (currently $O(n^{2.3\ldots})$ time). The second consists of iterative methods, such as the conjugate gradient method. These reduce the problem to computing n matrix–vector products, and thus make the running time proportional to mn where m is the number of nonzero entries, or sparsity, of A.[1] While this bound of n in the number of iterations is tight in the worst case, it can often be improved if A has additional structure, thus, making iterative methods popular in practice.

An important class of such instances has the form $L\mathbf{x} = \mathbf{b}$, where L is the Laplacian of an undirected graph G with n vertices and m edges

[1] Strictly speaking, this bound on the running time assumes that the numbers have bounded precision.

with m (typically) much smaller than n^2. Perhaps the simplest setting in which such *Laplacian systems* arise is when one tries to compute currents and voltages in a resistive electrical network. Laplacian systems are also important in practice, e.g., in areas such as scientific computing and computer vision. The fact that the system of equations comes from an underlying undirected graph made the problem of designing solvers especially attractive to theoretical computer scientists who entered the fray with tools developed in the context of graph algorithms and with the goal of bringing the running time down to $O(m)$. This effort gained serious momentum in the last 15 years, perhaps in light of an explosive growth in instance sizes which means an algorithm that does not scale near-linearly is likely to be impractical.

After decades of sustained research, we now have a solver for Laplacian systems that runs in $O(m \log n)$ time. While many researchers have contributed to this line of work, Spielman and Teng spearheaded this endeavor and were the first to bring the running time down to $\tilde{O}(m)$ by combining tools from graph partitioning, random walks, and low-stretch spanning trees with numerical methods based on Gaussian elimination and the conjugate gradient. Surprisingly, and not the original motivation behind this line of research, Laplacian solvers are impacting the theory of fast algorithms for fundamental graph problems; giving back to an area that empowered this work in the first place.

That is the story this monograph aims to tell in a comprehensive manner to researchers and aspiring students who work in algorithms or numerical linear algebra. The emerging paradigm of employing Laplacian solvers to design novel fast algorithms for graph problems is illustrated through a small but carefully chosen set of problems such as graph partitioning, computing the matrix exponential, simulating random walks, graph sparsification, and single-commodity flows. A significant part of this monograph is also dedicated to developing the algorithms and ideas that go into the proof of the Spielman–Teng Laplacian solver. It is a belief of the author that an understanding of these methods, which marry techniques from linear algebra and graph theory, will not only enrich the tool-set of an algorithm designer, but will also provide the ability to adapt these methods to design fast algorithms for other fundamental problems.

How to use this monograph. This monograph can be used as the text for a graduate-level course or act as a supplement to a course on spectral graph theory or algorithms. The writing style, which deliberately emphasizes the presentation of key ideas over rigor, should even be accessible to advanced undergraduates. If one desires to teach a course based on this monograph, then the best order is to go through the sections linearly. Essential are Sections 1 and 2 that contain the basic linear algebra material necessary to follow this monograph and Section 3 which contains the statement and a discussion of the main theorem regarding Laplacian solvers. Parts of this monograph can also be read independently. For instance, Sections 5–7 contain the Cheeger inequality based spectral algorithm for graph partitioning. Sections 15 and 16 can be read in isolation to understand the conjugate gradient method. Section 19 looks ahead into computing more general functions than the inverse and presents the Lanczos method. A dependency diagram between sections appears in Figure 1. For someone solely interested in a near-linear-time algorithm for solving Laplacian systems, the quick path to Section 14, where the approach of a short and new proof is presented, should suffice. However, the author recommends going all

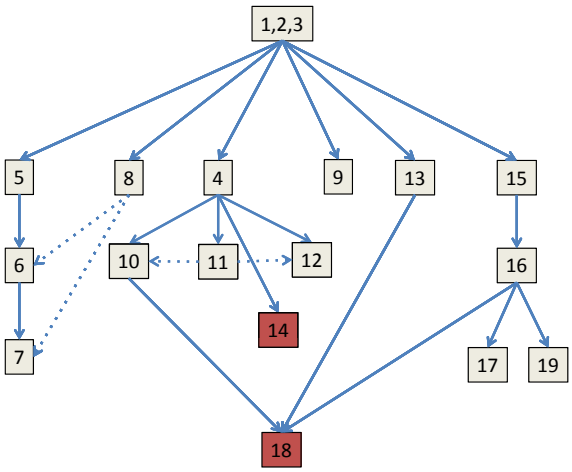

Fig. 1 The dependency diagram among the sections in this monograph. A dotted line from i to j means that the results of Section j use some results of Section i in a black-box manner and a full understanding is not required.

the way to Section 18 where multiple techniques developed earlier in the monograph come together to give an $\tilde{O}(m)$ Laplacian solver.

Acknowledgments. This monograph is partly based on lectures delivered by the author in a course at the Indian Institute of Science, Bangalore. Thanks to the scribes: Deeparnab Chakrabarty, Avishek Chatterjee, Jugal Garg, T. S. Jayaram, Swaprava Nath, and Deepak R. Special thanks to Elisa Celis, Deeparnab Chakrabarty, Lorenzo Orecchia, Nikhil Srivastava, and Sushant Sachdeva for reading through various parts of this monograph and providing valuable feedback. Finally, thanks to the reviewer(s) for several insightful comments which helped improve the presentation of the material in this monograph.

Bangalore
15 January 2013

Nisheeth K. Vishnoi
Microsoft Research India

Notation

- The set of real numbers is denoted by $\mathbb{R}$, and $\mathbb{R}_{\geq 0}$ denotes the set of nonnegative reals. We only consider real numbers in this monograph.
- The set of integers is denoted by $\mathbb{Z}$, and $\mathbb{Z}_{\geq 0}$ denotes the set of nonnegative integers.
- Vectors are denoted by boldface, e.g., $\mathbf{u}, \mathbf{v}$. A vector $\mathbf{v} \in \mathbb{R}^n$ is a column vector but often written as $\mathbf{v} = (v_1, \ldots, v_n)$. The transpose of a vector $\mathbf{v}$ is denoted by $\mathbf{v}^\top$.
- For vectors $\mathbf{u}, \mathbf{v}$, their inner product is denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$ or $\mathbf{u}^\top \mathbf{v}$.
- For a vector $\mathbf{v}$, $\|\mathbf{v}\|$ denotes its ℓ_2 or Euclidean norm where $\|\mathbf{v}\| \stackrel{\text{def}}{=} \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$. We sometimes also refer to the ℓ_1 or Manhattan distance norm $\|\mathbf{v}\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^{n} |v_i|$.
- The outer product of a vector $\mathbf{v}$ with itself is denoted by $\mathbf{v}\mathbf{v}^\top$.
- Matrices are denoted by capitals, e.g., A, L. The transpose of A is denoted by $A^\top$.
- We use t_A to denote the time it takes to multiply the matrix A with a vector.

- The A-norm of a vector $\mathbf{v}$ is denoted by $\|\mathbf{v}\|_A \stackrel{\text{def}}{=} \sqrt{\mathbf{v}^\top A \mathbf{v}}$.
- For a real symmetric matrix A, its real eigenvalues are ordered $\lambda_1(A) \leq \lambda_2(A) \leq \cdots \leq \lambda_n(A)$. We let $\Lambda(A) \stackrel{\text{def}}{=} [\lambda_1(A), \lambda_n(A)]$.
- A positive-semidefinite (PSD) matrix is denoted by $A \succeq 0$ and a positive-definite matrix $A \succ 0$.
- The norm of a symmetric matrix A is denoted by $\|A\| \stackrel{\text{def}}{=} \max\{|\lambda_1(A)|, |\lambda_n(A)|\}$. For a symmetric PSD matrix A, $\|A\| = \lambda_n(A)$.
- Thinking of a matrix A as a linear operator, we denote the image of A by $\mathrm{Im}(A)$ and the rank of A by $\mathrm{rank}(A)$.
- A graph G has a vertex set V and an edge set E. All graphs are assumed to be undirected unless stated otherwise. If the graph is weighted, there is a weight function $w : E \mapsto \mathbb{R}_{\geq 0}$. Typically, n is reserved for the number of vertices $|V|$, and m for the number of edges $|E|$.
- $\mathbb{E}_{\mathcal{F}}[\cdot]$ denotes the expectation and $\mathbb{P}_{\mathcal{F}}[\cdot]$ denotes the probability over a distribution $\mathcal{F}$. The subscript is dropped when clear from context.
- The following acronyms are used liberally, with respect to (w.r.t.), without loss of generality (w.l.o.g.), with high probability (w.h.p.), if and only if (iff), right-hand side (r.h.s.), left-hand side (l.h.s.), and such that (s.t.).
- Standard big-o notation is used to describe the limiting behavior of a function. $\tilde{O}$ denotes potential logarithmic factors which are ignored, i.e., $f = \tilde{O}(g)$ is equivalent to $f = O(g \log^k(g))$ for some constant k.

Part I

Basics

1

Basic Linear Algebra

This section reviews basics from linear algebra, such as eigenvalues and eigenvectors, that are relevant to this monograph. The spectral theorem for symmetric matrices and min–max characterizations of eigenvalues are derived.

1.1 Spectral Decomposition of Symmetric Matrices

One way to think of an $m \times n$ matrix A with real entries is as a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^m$ which maps a vector $\mathbf{v} \in \mathbb{R}^n$ to $A\mathbf{v} \in \mathbb{R}^m$. Let $\dim(S)$ be dimension of S, i.e., the maximum number of linearly independent vectors in S. The rank of A is defined to be the dimension of the image of this linear transformation. Formally, the image of A is defined to be $\mathrm{Im}(A) \stackrel{\text{def}}{=} \{\mathbf{u} \in \mathbb{R}^m : \mathbf{u} = A\mathbf{v} \text{ for some } \mathbf{v} \in \mathbb{R}^n\}$, and the rank is defined to be $\mathrm{rank}(A) \stackrel{\text{def}}{=} \dim(\mathrm{Im}(A))$ and is at most $\min\{m, n\}$.

We are primarily interested in the case when A is square, i.e., $m = n$, and symmetric, i.e., $A^\top = A$. Of interest are vectors $\mathbf{v}$ such that $A\mathbf{v} = \lambda\mathbf{v}$ for some λ. Such a vector is called an eigenvector of A with respect to (w.r.t.) the eigenvalue λ. It is a basic result in linear algebra that every real matrix has n eigenvalues, though some of them could

be complex. If A is symmetric, then one can show that the eigenvalues are real. For a complex number $z = a + ib$ with $a, b \in \mathbb{R}$, its conjugate is defined as $\bar{z} = a - ib$. For a vector $\mathbf{v}$, its conjugate transpose $\mathbf{v}^\star$ is the transpose of the vector whose entries are conjugates of those in $\mathbf{v}$. Thus, $\mathbf{v}^\star\mathbf{v} = \|\mathbf{v}\|^2$.

Lemma 1.1. If A is a real symmetric $n \times n$ matrix, then all of its eigenvalues are real.

Proof. Let λ be an eigenvalue of A, possibly complex, and $\mathbf{v}$ be the corresponding eigenvector. Then, $A\mathbf{v} = \lambda\mathbf{v}$. Conjugating both sides we obtain that $\mathbf{v}^\star A^\top = \overline{\lambda}\mathbf{v}^\star$, where $\mathbf{v}^\star$ is the conjugate transpose of $\mathbf{v}$. Hence, $\mathbf{v}^\star A\mathbf{v} = \overline{\lambda}\mathbf{v}^\star\mathbf{v}$, since A is symmetric. Thus, $\lambda\|\mathbf{v}\|^2 = \overline{\lambda}\|\mathbf{v}\|^2$ which implies that $\lambda = \overline{\lambda}$. Thus, $\lambda \in \mathbb{R}$. □

Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the n real eigenvalues, or the *spectrum*, of A with corresponding eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$. For a symmetric matrix, its *norm* is

$$\|A\| \stackrel{\text{def}}{=} \max\{|\lambda_1(A)|, |\lambda_n(A)|\}.$$

We now study eigenvectors that correspond to distinct eigenvalues.

Lemma 1.2. Let λ_i and λ_j be two eigenvalues of a symmetric matrix A, and $\mathbf{u}_i$, $\mathbf{u}_j$ be the corresponding eigenvectors. If $\lambda_i \neq \lambda_j$, then $\langle \mathbf{u}_i, \mathbf{u}_j \rangle = 0$.

Proof. Given $A\mathbf{u}_i = \lambda_i\mathbf{u}_i$ and $A\mathbf{u}_j = \lambda_j\mathbf{u}_j$, we have the following sequence of equalities. Since A is symmetric, $\mathbf{u}_i^\top A^\top = \mathbf{u}_i^\top A$. Thus, $\mathbf{u}_i^\top A\mathbf{u}_j = \lambda_i\mathbf{u}_i^\top\mathbf{u}_j$ on the one hand, and $\mathbf{u}_i^\top A\mathbf{u}_j = \lambda_j\mathbf{u}_i^\top\mathbf{u}_j$ on the other. Therefore, $\lambda_j\mathbf{u}_i^\top\mathbf{u}_j = \lambda_i\mathbf{u}_i^\top\mathbf{u}_j$. This implies that $\mathbf{u}_i^\top\mathbf{u}_j = 0$ since $\lambda_i \neq \lambda_j$. □

Hence, the eigenvectors corresponding to different eigenvalues are orthogonal. Moreover, if $\mathbf{u}_i$ and $\mathbf{u}_j$ correspond to the same eigenvalue λ, and are linearly independent, then any linear combination

is also an eigenvector corresponding to the same eigenvalue. The maximal eigenspace of an eigenvalue is the space spanned by all eigenvectors corresponding to that eigenvalue. Hence, the above lemma implies that one can decompose $\mathbb{R}^n$ into maximal eigenspaces U_i, each of which corresponds to an eigenvalue of A, and the eigenspaces corresponding to distinct eigenvalues are orthogonal. Thus, if $\lambda_1 < \lambda_2 < \cdots < \lambda_k$ are the set of distinct eigenvalues of a real symmetric matrix A, and U_i is the eigenspace associated with λ_i, then, from the discussion above,

$$\sum_{i=1}^{k} \dim(U_i) = n.$$

Hence, given that we can pick an orthonormal basis for each U_i, we may assume that the eigenvectors of A form an orthonormal basis for $\mathbb{R}^n$. Thus, we have the following spectral decomposition theorem.

Theorem 1.3. Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the spectrum of A with corresponding eigenvalues $\mathbf{u}_1, \ldots, \mathbf{u}_n$. Then, $A = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$.

Proof. Let $B \stackrel{\text{def}}{=} \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$. Then,

$$\begin{aligned} B\mathbf{u}_j &= \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top \mathbf{u}_j \\ &= \lambda_j \mathbf{u}_j \\ &= A\mathbf{u}_j. \end{aligned}$$

The above is true for all j. Since $\mathbf{u}_j$s are orthonormal basis of $\mathbb{R}^n$, we have for all $\mathbf{v} \in \mathbb{R}^n$, $A\mathbf{v} = B\mathbf{v}$. This implies $A = B$. □

Thus, when A is a real and symmetric matrix, $\mathrm{Im}(A)$ is spanned by the eigenvectors with nonzero eigenvalues. From a computational perspective, such a decomposition can be computed in time polynomial in the bits needed to represent the entries of A.[1]

[1] To be very precise, one can only compute eigenvalues and eigenvectors to a high enough precision in polynomial time. We will ignore this distinction for this monograph as we do not need to know the *exact* values.

1.2 Min–Max Characterizations of Eigenvalues

Now we present a variational characterization of eigenvalues which is very useful.

Lemma 1.4. If A is an $n \times n$ real symmetric matrix, then the largest eigenvalue of A is

$$\lambda_n(A) = \max_{\mathbf{v}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{\mathbf{v}^\top A\mathbf{v}}{\mathbf{v}^\top\mathbf{v}}.$$

Proof. Let $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$ be the eigenvalues of A, and let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ be the corresponding orthonormal eigenvectors which span $\mathbb{R}^n$. Hence, for all $\mathbf{v} \in \mathbb{R}^n$, there exist $c_1, \ldots, c_n \in \mathbb{R}$ such that $\mathbf{v} = \sum_i c_i\mathbf{u}_i$. Thus,

$$\begin{aligned}\langle \mathbf{v}, \mathbf{v}\rangle &= \left\langle \sum_i c_i\mathbf{u}_i, \sum_i c_i\mathbf{u}_i \right\rangle \\ &= \sum_i c_i^2.\end{aligned}$$

Moreover,

$$\begin{aligned}\mathbf{v}^\top A\mathbf{v} &= \left(\sum_i c_i\mathbf{u}_i\right)^\top \left(\sum_j \lambda_j\mathbf{u}_j\mathbf{u}_j^\top\right)\left(\sum_k c_k\mathbf{u}_k\right) \\ &= \sum_{i,j,k} c_ic_k\lambda_j(\mathbf{u}_i^\top\mathbf{u}_j)\cdot(\mathbf{u}_j^\top\mathbf{u}_k) \\ &= \sum_i c_i^2\lambda_i \\ &\le \lambda_n\sum_i c_i^2 = \lambda_n\langle \mathbf{v}, \mathbf{v}\rangle.\end{aligned}$$

Hence, $\forall\ \mathbf{v} \neq 0$, $\frac{\mathbf{v}^\top A\mathbf{v}}{\mathbf{v}^\top\mathbf{v}} \le \lambda_n$. This implies,

$$\max_{\mathbf{v}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{\mathbf{v}^\top A\mathbf{v}}{\mathbf{v}^\top\mathbf{v}} \le \lambda_n.$$

Now note that setting $\mathbf{v} = \mathbf{u}_n$ achieves this maximum. Hence, the lemma follows. □

If one inspects the proof above, one can deduce the following lemma just as easily.

Lemma 1.5. If A is an $n \times n$ real symmetric matrix, then the smallest eigenvalue of A is

$$\lambda_1(A) = \min_{\mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}} \frac{\mathbf{v}^\top A \mathbf{v}}{\mathbf{v}^\top \mathbf{v}}.$$

More generally, one can extend the proof of the lemma above to the following. We leave it as a simple exercise.

Theorem 1.6. If A is an $n \times n$ real symmetric matrix, then for all $1 \leq k \leq n$, we have

$$\lambda_k(A) = \min_{\mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}, \mathbf{v}^\top \mathbf{u}_i = 0, \forall i \in \{1, \ldots, k-1\}} \frac{\mathbf{v}^\top A \mathbf{v}}{\mathbf{v}^\top \mathbf{v}},$$

and

$$\lambda_k(A) = \max_{\mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}, \mathbf{v}^\top \mathbf{u}_i = 0, \forall i \in \{k+1, \ldots, n\}} \frac{\mathbf{v}^\top A \mathbf{v}}{\mathbf{v}^\top \mathbf{v}}.$$

Notes

Some good texts to review basic linear algebra are [35, 82, 85]. Theorem 1.6 is also called the Courant–Fischer–Weyl min–max principle.

2

The Graph Laplacian

This section introduces the graph Laplacian and connects the second eigenvalue to the connectivity of the graph.

2.1 The Graph Laplacian and Its Eigenvalues

Consider an undirected graph $G = (V, E)$ with $n \stackrel{\text{def}}{=} |V|$ and $m \stackrel{\text{def}}{=} |E|$. We assume that G is unweighted; this assumption is made to simplify the presentation but the content of this section readily generalizes to the weighted setting. Two basic matrices associated with G, indexed by its vertices, are its adjacency matrix A and its degree matrix D. Let d_i denote the degree of vertex i.

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

and

$$D_{i,j} \stackrel{\text{def}}{=} \begin{cases} d_i & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The graph Laplacian of G is defined to be $L \stackrel{\text{def}}{=} D - A$. We often refer to this as the Laplacian. In Section 5, we introduce the normalized

Laplacian which is different from L. To investigate the spectral properties of L, it is useful to first define the $n \times n$ matrices L_e as follows: For every $e = ij \in E$, let $L_e(i,j) = L_e(j,i) \stackrel{\text{def}}{=} -1$, let $L_e(i,i) = L_e(j,j) = 1$, and let $L_e(i',j') = 0$, if $i' \notin \{i,j\}$ or $j' \notin \{i,j\}$. The following then follows from the definition of the Laplacian and L_es.

Lemma 2.1. Let L be the Laplacian of a graph $G = (V,E)$. Then, $L = \sum_{e \in E} L_e$.

This can be used to show that the smallest eigenvalue of a Laplacian is always nonnegative. Such matrices are called positive semidefinite, and are defined as follows.

Definition 2.1. A symmetric matrix A is called positive semidefinite (PSD) if $\lambda_1(A) \geq 0$. A PSD matrix is denoted by $A \succeq 0$. Further, A is said to be positive definite if $\lambda_1(A) > 0$, denoted $A \succ 0$.

Note that the Laplacian is a PSD matrix.

Lemma 2.2. Let L be the Laplacian of a graph $G = (V,E)$. Then, $L \succeq 0$.

Proof. For any $\mathbf{v} = (v_1, \ldots, v_n)$,

$$\begin{aligned}
\mathbf{v}^\top L \mathbf{v} &= \mathbf{v}^\top \sum_{e \in E} L_e \mathbf{v} \\
&= \sum_{e \in E} \mathbf{v}^\top L_e \mathbf{v} \\
&= \sum_{e=ij \in E} (v_i - v_j)^2 \\
&\geq 0.
\end{aligned}$$

Hence, $\min_{\mathbf{v} \neq \mathbf{0}} \mathbf{v}^\top L \mathbf{v} \geq 0$. Thus, appealing to Theorem 1.6, we obtain that $L \succeq 0$. □

Is $L \succ 0$? The answer to this is no: Let $\mathbf{1}$ denote the vector with all coordinates 1. Then, it follows from Lemma 2.1 that $L\mathbf{1} = 0 \cdot \mathbf{1}$. Hence, $\lambda_1(L) = 0$.
For weighted a graph $G = (V, E)$ with edge weights given by a weight function $w_G : E \mapsto \mathbb{R}_{\geq 0}$, one can define

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

and

$$D_{i,j} \stackrel{\text{def}}{=} \begin{cases} \sum_l w_G(il) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Then, the definition of the Laplacian remains the same, namely, $L = D - A$.

2.2 The Second Eigenvalue and Connectivity

What about the second eigenvalue, $\lambda_2(L)$, of the Laplacian? We will see in later sections on graph partitioning that the second eigenvalue of the Laplacian is intimately related to the *conductance* of the graph, which is a way to measure how well connected the graph is. In this section, we establish that $\lambda_2(L)$ determines if G is connected or not. This is the first result where we make a formal connection between the spectrum of the Laplacian and a property of the graph.

Theorem 2.3. Let L be the Laplacian of a graph $G = (V, E)$. Then, $\lambda_2(L) > 0$ iff G is connected.

Proof. If G is disconnected, then L has a block diagonal structure. It suffices to consider only two disconnected components. Assume the disconnected components of the graph are G_1 and G_2, and the corresponding vertex sets are V_1 and V_2. The Laplacian can then be rearranged as follows:

$$L = \begin{pmatrix} L(G_1) & 0 \\ 0 & L(G_2) \end{pmatrix},$$

where $L(G_i)$ is a $|V_i| \times |V_i|$ matrix for $i = 1, 2$. Consider the vector $\mathbf{x}_1 \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{0}_{|V_1|} \\ \mathbf{1}_{|V_2|} \end{pmatrix}$, where $\mathbf{1}_{|V_i|}$ and $\mathbf{0}_{|V_i|}$ denote the all 1 and all 0 vectors of dimension $|V_i|$, respectively. This is an eigenvector corresponding to the smallest eigenvalue, which is zero. Now consider $\mathbf{x}_2 \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{1}_{|V_1|} \\ \mathbf{0}_{|V_2|} \end{pmatrix}$. Since $\langle \mathbf{x}_2, \mathbf{x}_1 \rangle = 0$, the smallest eigenvalue, which is 0 has multiplicity at least 2. Hence, $\lambda_2(L) = 0$.

For the other direction, assume that $\lambda_2(L) = 0$ and that G is connected. Let $\mathbf{u}_2$ be the second eigenvector normalized to have length 1. Then, $\lambda_2(L) = \mathbf{u}_2^\top L \mathbf{u}_2$. Thus, $\sum_{e=ij \in E} (\mathbf{u}_2(i) - \mathbf{u}_2(j))^2 = 0$. Hence, for all $e = ij \in E$, $\mathbf{u}_2(i) = \mathbf{u}_2(j)$.

Since G is connected, there is a path from vertex 1 to every vertex $j \neq 1$, and for each intermediate edge $e = ik$, $\mathbf{u}_2(i) = \mathbf{u}_2(k)$. Hence, $\mathbf{u}_2(1) = \mathbf{u}_2(j), \forall j$. Hence, $\mathbf{u}_2 = (c, c, \ldots, c)$. However, we also know $\langle \mathbf{u}_2, \mathbf{1} \rangle = 0$ which implies that $c = 0$. This contradicts the fact that $\mathbf{u}_2$ is a nonzero eigenvector and establishes the theorem. □

Notes

There are several books on spectral graph theory which contain numerous properties of graphs, their Laplacians and their eigenvalues; see [23, 34, 88]. Due to the connection of the second eigenvalue of the Laplacian to the connectivity of the graph, it is sometimes called its *algebraic connectivity* or its Fiedler value, and is attributed to [28].

3

Laplacian Systems and Solvers

This section introduces Laplacian systems of linear equations and notes the properties of the near-linear-time Laplacian solver relevant for the applications presented in this monograph. Sections 5–12 cover several applications that reduce fundamental graph problems to solving a small number of Laplacian systems.

3.1 System of Linear Equations

An $n \times n$ matrix A and vector $\mathbf{b} \in \mathbb{R}^n$ together define a system of linear equations $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x} = (x_1, \ldots, x_n)$ are variables. By definition, a solution to this linear system exists if and only if (iff) $\mathbf{b}$ lies in the image of A. A is said to be invertible, i.e., a solution exists for all $\mathbf{b}$, if its image is $\mathbb{R}^n$, the entire space. In this case, the inverse of A is denoted by A^{-1}. The inverse of the linear operator A, when $\mathbf{b}$ is restricted to the image of A, is also well defined and is denoted by A^{+}. This is called the pseudo-inverse of A.

3.2 Laplacian Systems

Now consider the case when, in a system of equations $A\mathbf{x} = \mathbf{b}$, $A = L$ is the graph Laplacian of an undirected graph. Note that this system is not invertible unless $\mathbf{b} \in \mathrm{Im}(L)$. It follows from Theorem 2.3 that for a connected graph, $\mathrm{Im}(L)$ consists of all vectors orthogonal to the vector $\mathbf{1}$. Hence, we can solve the system of equations $L\mathbf{x} = \mathbf{b}$ if $\langle \mathbf{b}, \mathbf{1} \rangle = 0$. Such a system will be referred to as a Laplacian system of linear equations or, in short, a Laplacian system. Hence, we can define the pseudo-inverse of the Laplacian as the linear operator which takes a vector $\mathbf{b}$ orthogonal to $\mathbf{1}$ to its pre-image.

3.3 An Approximate, Linear-Time Laplacian Solver

In this section we summarize the near-linear algorithm known for solving a Laplacian system $L\mathbf{x} = \mathbf{b}$. This result is the basis of the applications and a part of the latter half of this monograph is devoted to its proof.

Theorem 3.1. There is an algorithm LSolve which takes as input a graph Laplacian L, a vector $\mathbf{b}$, and an error parameter $\varepsilon > 0$, and returns $\mathbf{x}$ satisfying

$$\|\mathbf{x} - L^+\mathbf{b}\|_L \leq \varepsilon \|L^+\mathbf{b}\|_L,$$

where $\|\mathbf{b}\|_L \stackrel{\text{def}}{=} \sqrt{\mathbf{b}^T L \mathbf{b}}$. The algorithm runs in $\widetilde{O}(m \log 1/\varepsilon)$ time, where m is the number of nonzero entries in L.

Let us first relate the norm in the theorem above to the Euclidean norm. For two vectors $\mathbf{v}, \mathbf{w}$ and a symmetric PSD matrix A,

$$\lambda_1(A)\|\mathbf{v} - \mathbf{w}\|^2 \leq \|\mathbf{v} - \mathbf{w}\|_A^2 = (\mathbf{v} - \mathbf{w})^T A (\mathbf{v} - \mathbf{w}) \leq \lambda_n(A)\|\mathbf{v} - \mathbf{w}\|^2.$$

In other words,

$$\|A^+\|^{1/2}\|\mathbf{v} - \mathbf{w}\| \leq \|\mathbf{v} - \mathbf{w}\|_A \leq \|A\|^{1/2}\|\mathbf{v} - \mathbf{w}\|.$$

Hence, the distortion in distances due to the A-norm is at most $\sqrt{\lambda_n(A)/\lambda_1(A)}$.

For the graph Laplacian of an unweighted and connected graph, when all vectors involved are orthogonal to $\mathbf{1}$, $\lambda_2(L)$ replaces $\lambda_1(L)$. It can be checked that when G is unweighted, $\lambda_2(L) \geq 1/\text{poly}(n)$ and $\lambda_n(L) \leq \text{Tr}(L) = m \leq n^2$. When G is weighted the ratio between the L-norm and the Euclidean norm scales polynomially with the ratio of the largest to the smallest weight edge. Finally, note that for any two vectors, $\|\mathbf{v} - \mathbf{w}\|_\infty \leq \|\mathbf{v} - \mathbf{w}\|$. Hence, by a choice of $\varepsilon \stackrel{\text{def}}{=} \delta/\text{poly}(n)$ in Theorem 3.1, we ensure that the approximate vector output by LSolve is δ close in every coordinate to the actual vector. Note that since the dependence on the tolerance on the running time of Theorem 3.1 is logarithmic, the running time of LSolve remains $\tilde{O}(m \log 1/\varepsilon)$.

3.4 Linearity of the Laplacian Solver

While the running time of LSolve is $\tilde{O}(m \log 1/\varepsilon)$, the algorithm produces an approximate solution that is off by a little from the exact solution. This creates the problem of estimating the error accumulation as one iterates this solver. To get around this, we note an important feature of LSolve: On input L, $\mathbf{b}$, and ε, it returns the vector $\mathbf{x} = Z\mathbf{b}$, where Z is an $n \times n$ matrix and depends only on L and ε. Z is a symmetric linear operator satisfying

$$(1 - \varepsilon)Z^+ \preceq L \preceq (1 + \varepsilon)Z^+, \tag{3.1}$$

and has the same image as L. Note that in applications, such as that in Section 9, for Z to satisfy $\|Z - L^+\| \leq \varepsilon$, the running time is increased to $\tilde{O}(m \log(1/\varepsilon\lambda_2(L)))$. Since in most of our applications $\lambda_2(L)$ is at least $1/\text{poly}(n)$, we ignore this distinction.

Notes

A good text to learn about matrices, their norms, and the inequalities that relate them is [17]. The pseudo-inverse is sometimes also referred to as the Moore–Penrose pseudo-inverse. Laplacian systems arise in many areas such as machine learning [15, 56, 89], computer vision [57, 73], partial differential equations and interior point methods [24, 30], and solvers are naturally needed; see also surveys [75] and [84].

We will see several applications that reduce fundamental graph problems to solving a small number of Laplacian systems in Sections 5–12. Sections 8 and 9 (a result of which is assumed in Sections 5 and 6) require the Laplacian solver to be linear. The notable applications we will *not* be covering are an algorithm to sample a random spanning tree [45] and computing multicommodity flows [46].

Theorem 3.1 was first proved in [77] and the full proof appears in a series of papers [78, 79, 80]. The original running time contained a very large power of the $\log n$ term (hidden in $\tilde{O}(\cdot)$). This power has since been reduced in a series of work [8, 9, 49, 62] and, finally, brought down to $\log n$ in [50]. We provide a proof of Theorem 3.1 in Section 18 along with its linearity property mentioned in this section. Recently, a simpler proof of Theorem 3.1 was presented in [47]. This proof does not require many of the ingredients such as spectral sparsifiers (Section 10), preconditioning (Section 17), and conjugate gradient (Section 16). While we present a sketch of this proof in Section 14, we recommend that the reader go through the proof in Section 18 and, in the process, familiarize themselves with this wide array of techniques which may be useful in general.

4

Graphs as Electrical Networks

This section introduces how a graph can be viewed as an electrical network composed of resistors. It is shown how voltages and currents can be computed by solving a Laplacian system. The notions of effective resistance, electrical flows, and their energy are presented. Viewing graphs as electrical networks will play an important role in applications presented in Sections 10–12 and in the approach to a simple proof of Theorem 3.1 presented in Section 14.

4.1 Incidence Matrices and Electrical Networks

Given an undirected, unweighted graph $G = (V, E)$, consider an arbitrary orientation of its edges. Let $B \in \{-1, 0, 1\}^{m \times n}$ be the matrix whose rows are indexed by the edges and columns by the vertices of G where the entry corresponding to (e, i) is 1 if a vertex i is the tail of the directed edge corresponding to e, is -1 if i is the head of the directed edge e, and is zero otherwise. B is called the (edge-vertex) *incidence matrix* of G. The Laplacian can now be expressed in terms of B. While B depends on the choice of the directions to the edges, the Laplacian does not.

Lemma 4.1. Let G be a graph with (arbitrarily chosen) incidence matrix B and Laplacian L. Then, $B^\top B = L$.

Proof. For the diagonal elements of $B^\top B$, i.e., $(B^\top B)_{i,i} = \sum_e B_{i,e} B_{e,i}$. The terms are nonzero only for those edges e which are incident to i, in which case the product is 1, and hence, this sum gives the degree of vertex i in the undirected graph. For other entries, $(B^\top B)_{i,j} = \sum_e B_{i,e} B_{e,j}$. The product terms are nonzero only when the edge e is shared by i and j. In either case the product is -1. Hence, $(B^\top B)_{i,j} = -1$, $\forall i \neq j$, $ij \in E$. Hence, $B^\top B = L$. □

Now we associate an electrical network to G. Replace each edge with a resistor of value 1. To make this circuit interesting, we need to add power sources to its vertices. Suppose $\mathbf{c}_{\text{ext}} \in \mathbb{R}^n$ is a vector which indicates how much current is going in at each vertex. This will induce voltages at each vertex and a current across each edge. We capture these by vectors $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{i} \in \mathbb{R}^m$, respectively. Kirchoff's law asserts that, for every vertex, the difference of the outgoing current and the incoming current on the edges adjacent to it equals the external current input at that vertex. Thus,

$$B^\top \mathbf{i} = \mathbf{c}_{\text{ext}}.$$

On the other hand, Ohm's law asserts that the current in an edge equals the voltage difference divided by the resistance of that edge. Since in our case all resistances are one, this gives us the following relation.

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

Combining these last two equalities with Lemma 4.1 we obtain that

$$B^\top B\mathbf{v} = L\mathbf{v} = \mathbf{c}_{\text{ext}}.$$

If $\langle \mathbf{c}_{\text{ext}}, \mathbf{1} \rangle = 0$, which means there is no current accumulation inside the electrical network, we can solve for $\mathbf{v} = L^+ \mathbf{c}_{\text{ext}}$. The voltage vector is not unique since we can add the same constant to each of its entries and it still satisfies Ohm's law. The currents across every edge, however,

are unique. Define vectors $\mathbf{e}_i \in \mathbb{R}^n$ for $i \in V$ which have a 1 at the i-th location and zero elsewhere. Then the current through the edge $e = ij$, taking into account the sign, is $(\mathbf{e}_i - \mathbf{e}_j)^\top \mathbf{v} = (\mathbf{e}_i - \mathbf{e}_j)^\top L^+ \mathbf{c}_{\text{ext}}$. Thus, computing voltages and currents in such a network is equivalent to solving a Laplacian system.

4.2 Effective Resistance and the Π Matrix

Now we consider a specific vector $\mathbf{c}_{\text{ext}}$ and introduce an important quantity related to each edge, the effective resistance. Consider $\mathbf{c}_{\text{ext}} \stackrel{\text{def}}{=} \mathbf{e}_i - \mathbf{e}_j$. The voltage difference between vertices i and j is then $(\mathbf{e}_i - \mathbf{e}_j)^\top L^+ (\mathbf{e}_i - \mathbf{e}_j)$. When $e = ij$ is an edge, this quantity is called the effective resistance of e.

Definition 4.1. Given a graph $G = (V, E)$ with Laplacian L and edge $e = ij \in E$,

$$R_{\text{eff}}(e) \stackrel{\text{def}}{=} (\mathbf{e}_i - \mathbf{e}_j)^\top L^+ (\mathbf{e}_i - \mathbf{e}_j).$$

$R_{\text{eff}}(e)$ is the potential difference across e when a unit current is inducted at i and taken out at j.

We can also consider the current through an edge f when a unit current is inducted and taken out at the endpoints of a possibly different edge $e \in E$. We capture this by the Π matrix which we now define formally. Let us denote the rows from the B matrix $\mathbf{b}_e$ and $\mathbf{b}_f$ respectively. Then,

$$\Pi(f, e) \stackrel{\text{def}}{=} \mathbf{b}_f^\top L^+ \mathbf{b}_e.$$

In matrix notation,

$$\Pi = B L^+ B^\top.$$

Note also that $\Pi(e, e)$ is the effective resistance of the edge e. The matrix Π has several interesting properties. The first is trivial and follows from the fact the Laplacian and, hence, its pseudo-inverse is symmetric.

Proposition 4.2. Π is symmetric.

Additionally, Π is a projection matrix.

Proposition 4.3. $\Pi^2 = \Pi$.

Proof. $\Pi^2 = BL^+B^\top \cdot BL^+B^\top = B\underbrace{L^+B^\top B}_{=I}L^+B^\top = BL^+B^\top = \Pi$.
The third equality comes from the fact that the rows of B are orthogonal to $\mathbf{1}$. □

Proposition 4.4. The eigenvalues of Π are all either 0 or 1.

Proof. Let $\mathbf{v}$ be an eigenvector of Π corresponding to the eigenvalue λ. Hence, $\Pi\mathbf{v} = \lambda\mathbf{v} \Rightarrow \Pi^2\mathbf{v} = \lambda\Pi\mathbf{v} \Rightarrow \Pi\mathbf{v} = \lambda^2\mathbf{v} \Rightarrow \lambda\mathbf{v} = \lambda^2\mathbf{v} \Rightarrow (\lambda^2 - \lambda)\mathbf{v} = \mathbf{0} \Rightarrow \lambda = 0, 1$. □

Hence, it is an easy exercise to prove that if G is connected, then $\text{rank}(\Pi) = n - 1$. In this case, Π has exactly $n - 1$ eigenvalues which are 1 and $m - n + 1$ eigenvalues which are 0.

Finally, we note the following theorem which establishes a connection between spanning trees and effective resistances. While this theorem is not explicitly used in this monograph, the intuition arising from it is employed in sparsifying graphs.

Theorem 4.5. Let T be a spanning tree chosen uniformly at random from all spanning trees in G. Then, the probability that an edge e belongs to T is given by

$$\mathbb{P}[e \in T] = R_{\text{eff}}(e) = \Pi(e, e).$$

4.3 Electrical Flows and Energy

Given a graph $G = (V, E)$, we fix an orientation of its edges thus resulting in an incidence matrix B. Moreover, we associate unit

resistance with each edge of the graph. Now suppose that for vertices s and t one unit of current is injected at vertex s and taken out at t. The electrical flow on each edge is then captured by the vector

$$\mathbf{f}^\star \stackrel{\text{def}}{=} BL^+(\mathbf{e}_s - \mathbf{e}_t).$$

In general, an *s,t-flow* is an assignment of values to directed edges such that the total incoming flow is the same as the total outgoing flow at all vertices except s and t. For a flow vector $\mathbf{f}$, the energy it consumes is defined to be

$$E(\mathbf{f}) \stackrel{\text{def}}{=} \sum_e f_e^2.$$

Thus, the energy of $\mathbf{f}^\star$ is

$$\sum_e (\mathbf{b}_e^\top L^+(\mathbf{e}_s - \mathbf{e}_t))^2 = \sum_e (\mathbf{e}_s - \mathbf{e}_t)^\top L^+ \mathbf{b}_e \mathbf{b}_e^\top L^+(\mathbf{e}_s - \mathbf{e}_t). \qquad (4.1)$$

By taking the summation inside and noting that $L = B^\top B = \sum_e \mathbf{b}_e \mathbf{b}_e^\top$, Equation (4.1) is equivalent to

$$(\mathbf{e}_s - \mathbf{e}_t)^\top L^+ L L^+ (\mathbf{e}_s - \mathbf{e}_t) = (\mathbf{e}_s - \mathbf{e}_t)^\top L^+ (\mathbf{e}_s - \mathbf{e}_t).$$

Thus we have proved the following proposition.

Proposition 4.6. If $\mathbf{f}^\star$ is the unit s,t-flow, then its energy is

$$E(\mathbf{f}^\star) = (\mathbf{e}_s - \mathbf{e}_t)^\top L^+ (\mathbf{e}_s - \mathbf{e}_t).$$

The following is an important property of electrical s,t-flows and will be useful in finding applications related to combinatorial flows in graphs.

Theorem 4.7. Given a graph $G = (V, E)$ with unit resistances across all edges, if $\mathbf{f}^\star \stackrel{\text{def}}{=} BL^+(\mathbf{e}_s - \mathbf{e}_t)$, then $\mathbf{f}^\star$ minimizes the energy consumed $E(\mathbf{f}) \stackrel{\text{def}}{=} \sum_e f_e^2$ among all unit flows from s to t.

Proof. Let $\Pi = BL^+B^\top$ be as before. Proposition 4.3 implies that $\Pi^2 = \Pi$ and, hence, for all vectors $\mathbf{g}$, $\|\mathbf{g}\|^2 \geq \|\Pi\mathbf{g}\|^2$ where equality holds

iff $\mathbf{g}$ is in the image of Π. Let $\mathbf{f}$ be any flow such that $B^\top \mathbf{f} = \mathbf{e}_s - \mathbf{e}_t$, i.e., any unit s,t-flow. Then,

$$E(\mathbf{f}) = \|\mathbf{f}\|^2 \geq \|\Pi \mathbf{f}\|^2 = \mathbf{f}^\top \Pi \Pi \mathbf{f} = \mathbf{f}^\top \Pi \mathbf{f} = \mathbf{f}^\top B L^+ B \mathbf{f}.$$

Hence, using the fact that $B^\top \mathbf{f} = \mathbf{e}_s - \mathbf{e}_t$, one obtains that

$$E(\mathbf{f}) \geq \mathbf{f}^\top B L^+ B \mathbf{f} = (\mathbf{e}_s - \mathbf{e}_t)^\top L^+ (\mathbf{e}_s - \mathbf{e}_t) \overset{\text{Prop. } 4.6}{=} E(\mathbf{f}^\star).$$

Hence, for any unit s,t-flow $\mathbf{f}$, $E(\mathbf{f}) \geq E(\mathbf{f}^\star)$. □

4.4 Weighted Graphs

Our results also extend to weighted graphs. Suppose the graph $G = (V, E)$ has weights given by $w : E \mapsto \mathbb{R}_{\geq 0}$. Let W be the $m \times m$ diagonal matrix such that $W(e, e) = w(e)$. Then, for an incidence matrix B given an orientation of G, the Laplacian is $L \overset{\text{def}}{=} B^\top W B$. The Π matrix in this setting is $W^{1/2} B L^+ B^\top W^{1/2}$. To set up a corresponding electrical network, we associate a resistance $r_e \overset{\text{def}}{=} 1/w(e)$ with each edge. Thus, for a given voltage vector $\mathbf{v}$, the current vector, by Ohm's Law, is $\mathbf{i} = WB\mathbf{v}$. The effective resistance remains $R_{\text{eff}}(e) = (\mathbf{e}_i - \mathbf{e}_j)^\top L^+ (\mathbf{e}_i - \mathbf{e}_j)$ where L^+ is the pseudo-inverse of the Laplacian which involves W. For vertices s and t, the unit s,t-flow is $\mathbf{f}^\star \overset{\text{def}}{=} WBL^+(\mathbf{e}_s - \mathbf{e}_t)$ and its energy is defined to be $\sum_e r_e (f_e^\star)^2$, which turns out to be $(\mathbf{e}_s - \mathbf{e}_t)^\top L^+ (\mathbf{e}_s - \mathbf{e}_t)$. It is an easy exercise to check that all the results in this section hold for this weighted setting.

Notes

The connection between graphs, random walks, and electrical networks is an important one, and its study has yielded many surprising results. The books [25] and [54] are good pointers for readers intending to explore this connection.

While we do not need Theorem 4.5 for this monograph, an interested reader can try to prove it using the Matrix-Tree Theorem, or refer to [34]. It forms the basis for another application that uses Laplacian solvers to develop fast algorithms: Generating a random spanning tree in time $\tilde{O}(mn^{1/2})$, see [45].

Part II

Applications

5

Graph Partitioning I
The Normalized Laplacian

This section introduces the fundamental problem of finding a cut of least conductance in a graph, called Sparsest Cut. A quadratic program is presented which captures the Sparsest Cut problem exactly. Subsequently, a relaxation of this program is considered where the optimal value is essentially the second eigenvalue of the normalized Laplacian; this provides a lower bound on the conductance of the graph. In Sections 6 and 7 this connection is used to come up with approximation algorithms for the Sparsest Cut and related Balanced Edge-Separator problems. Finally, in Section 8, it is shown how Laplacian solvers can be used to compute the second eigenvector and the associated second eigenvector in $\tilde{O}(m)$ time.

5.1 Graph Conductance

Given an undirected, unweighted graph $G = (V, E)$ with n vertices and m edges, we are interested in vertex cuts in the graph. A vertex cut is a partition of V into two parts, $S \subseteq V$ and $\bar{S} \stackrel{\text{def}}{=} V \backslash S$, which we denote by $(S, \bar{S})$. Before we go on to define conductance, we need a way to measure the *size* of a cut given by $S \subseteq V$. One measure is its

cardinality $|S|$. Another is the sum of degrees of all the vertices in S. If the graph is regular, i.e., all vertices have the same degree, then these two are the same up to a factor of this fixed degree. Otherwise, they are different and a part of the latter is called the volume of the set. Formally, for $S \subseteq V$, the volume of S is $\mathrm{vol}(S) \stackrel{\text{def}}{=} \sum_{i \in S} d_i$, where d_i is the degree of vertex i. By a slight abuse of notation, we define $\mathrm{vol}(G) \stackrel{\text{def}}{=} \mathrm{vol}(V) = \sum_{i \in V} d_i = 2m$. The number of edges that cross the cut $(S, \bar{S})$, i.e., have one end point in S and the other in $\bar{S}$, is denoted $|E(S, \bar{S})|$. Now we define the conductance of a cut.

Definition 5.1. The conductance of a cut $(S, \bar{S})$ (also referred to as its normalized cut value or cut ratio) is defined to be

$$\phi(S) \stackrel{\text{def}}{=} \frac{|E(S, \bar{S})|}{\min\{\mathrm{vol}(S), \mathrm{vol}(\bar{S})\}}.$$

The conductance of a cut measures the ratio of edges going out of a cut to the total edges incident to the smaller side of the cut. This is always a number between 0 and 1. Some authors define the conductance to be

$$\frac{|E(S, \bar{S})|}{\min\{|S|, |\bar{S}|\}},$$

where $|S|$ denotes the cardinality of S. The former definition is preferred to the latter one as $\phi(S)$ lies between 0 and 1 in case of the former while there is no such bound on $\phi(S)$ in the latter. Specifically, the latter is not a *dimension-less* quantity: if we replace each edge by k copies of itself, this value will change, while the value given by the former definition remains invariant. The graph conductance problem is to compute the conductance of the graph which is defined to be

$$\phi(G) \stackrel{\text{def}}{=} \min_{\emptyset \neq S \subsetneq V} \phi(S).$$

This problem is often referred to as the Sparsest Cut problem and is NP-hard. This, and its cousin, the Balanced Edge-Separator problem (to be introduced in Section 7) are intensely studied, both in theory and practice, and have far reaching connections to spectral graph theory, the study of random walks, and metric embeddings. Besides

being theoretically rich, they are of great practical importance as they play a central role in the design of recursive algorithms, image segmentation, community detection, and clustering.

Another quantity which is closely related to the conductance and is often easier to manipulate is the following.

Definition 5.2. For a cut $(S, \bar{S})$, the h-value of a set is defined to be

$$h(S) \stackrel{\text{def}}{=} \frac{|E(S,\bar{S})|}{\text{vol}(S) \cdot \text{vol}(\bar{S})} \cdot \text{vol}(G)$$

and the h-value of the graph is defined to be

$$h(G) \stackrel{\text{def}}{=} \min_{\emptyset \neq S \subsetneq V} h(S).$$

We first observe the relation between h and ϕ.

Lemma 5.1. For all S, $\phi(S) \leq h(S) \leq 2\phi(S)$.

Proof. This follows from the observations that for any cut $(S, \bar{S})$,

$$\text{vol}(G) \geq \max\{\text{vol}(S), \text{vol}(\bar{S})\} \geq \text{vol}(G)/2$$

and

$$\max\{\text{vol}(S), \text{vol}(\bar{S})\} \cdot \min\big\{\text{vol}(S), \text{vol}(\bar{S})\big\} = \text{vol}(S) \cdot \text{vol}(\bar{S}).$$

Hence, $\phi(G) \leq h(G) \leq 2\phi(G)$. □

Thus, the h-value captures the conductance of a graph up to a factor of 2. Computing the h-value of a graph is not any easier than computing its conductance. However, as we see next, it can be formulated as a mathematical program which can then be relaxed to an eigenvalue problem involving the Laplacian.

5.2 A Mathematical Program

We will write down a mathematical program which captures the h-value of the conductance. First, we introduce some notation which will be useful.

5.2.1 Probability Measures on Vertices and Cuts as Vectors

Let $\nu : E \to [0,1]$ be the uniform probability measure defined on the set of edges E,

$$\nu(e) \stackrel{\text{def}}{=} \frac{1}{m}.$$

For a subset of edges $F \subseteq E$, $\nu(F) \stackrel{\text{def}}{=} \sum_{e \in F} \nu(e)$. Next, we consider a measure on the vertices induced by the degrees. For $i \in V$,

$$\mu(i) \stackrel{\text{def}}{=} \frac{d_i}{\text{vol}(G)} = \frac{d_i}{2m}.$$

Note that μ is a probability measure on V. We extend μ to subsets $S \subseteq V$,

$$\mu(S) \stackrel{\text{def}}{=} \sum_{i \in S} \mu(i) = \sum_{i \in S} \frac{d_i}{\text{vol}(G)} = \frac{\text{vol}(S)}{\text{vol}(G)}.$$

With these definitions it follows that

$$\phi(S) = \frac{\nu(E(S,\bar{S}))}{2\min\{\mu(S),\mu(\bar{S})\}} \quad \text{and} \quad h(S) = \frac{\nu(E(S,\bar{S}))}{2\mu(S)\mu(\bar{S})}.$$

Given $S \subseteq V$, let $\mathbf{1}_S : V \mapsto \{0,1\}$ denote the indicator function for the set S by

$$1_S(i) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$(1_S(i) - 1_S(j))^2 = \begin{cases} 1 & \text{if } (i \in S \text{ and } j \in \bar{S}) \text{ or } (i \in \bar{S} \text{ and } j \in S), \\ 0 & \text{if } (i \in S \text{ and } j \in S) \text{ or } (i \in \bar{S} \text{ and } j \in \bar{S}). \end{cases}$$

Therefore,

$$\mathop{\mathbb{E}}_{ij \leftarrow \nu}[(1_S(i) - 1_S(j))^2] = \frac{|E(S,\bar{S})|}{m} = \nu(E(S,\bar{S})).$$

Moreover, for any S

$$\begin{aligned}
&\mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[(1_S(i)-1_S(j))^2] \\
&\quad= \mathop{\mathbb{P}}_{(i,j)\leftarrow\mu\times\mu}[(1_S(i)-1_S(j))^2=1] \\
&\quad= \mathop{\mathbb{P}}_{(i,j)\leftarrow\mu\times\mu}[i\in S, j\in\bar{S} \text{ or } i\in\bar{S}, j\in S] \\
&\quad= \mathop{\mathbb{P}}_{i\leftarrow\mu}[i\in S]\mathop{\mathbb{P}}_{j\leftarrow\mu}[j\in\bar{S}]+\mathop{\mathbb{P}}_{i\leftarrow\mu}[i\in\bar{S}]\mathop{\mathbb{P}}_{j\leftarrow\mu}[j\in S] \\
&\qquad\text{(since } i \text{ and } j \text{ are chosen independently)} \\
&\quad= \mu(S)\mu(\bar{S})+\mu(\bar{S})\mu(S)=2\mu(S)\mu(\bar{S}).
\end{aligned}$$

Therefore,

$$h(S)=\frac{\nu(E(S,\bar{S}))}{2\mu(S)\mu(\bar{S})}=\frac{\mathbb{E}_{ij\leftarrow\nu}[(1_S(i)-1_S(j))^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(1_S(i)-1_S(j))^2]}.$$

Hence, noting the one-to-one correspondence between sets $S\subseteq V$ and functions $\mathbf{x}\in\{0,1\}^n$, we have proved the following mathematical programming characterization of the conductance.

Lemma 5.2. Consider the h-value of a graph G and the probability distributions ν and μ as above. Then, for $\mathbf{x}\neq\mathbf{0},\mathbf{1}$,

$$h(G)=\min_{\mathbf{x}\in\{0,1\}^n}\frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i-x_j)^2]}{\mathbb{E}_{i\leftarrow\mu}\mathbb{E}_{j\leftarrow\mu}[(x_i-x_j)^2]}.$$

As noted before, this quantity in the right-hand side (r.h.s.) of the lemma above is hard to compute. Let us now try to relax the condition that $\mathbf{x}\in\{0,1\}^n$ to $\mathbf{x}\in\mathbb{R}^n$. We will refer to this as the *real* conductance; this notation, as we will see shortly, will go away.

Definition 5.3. The real conductance of a graph G is

$$h_{\mathbb{R}}(G)\stackrel{\text{def}}{=}\min_{\mathbf{x}\in\mathbb{R}^n}\frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i-x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i-x_j)^2]}.$$

Since we are relaxing from a 0/1 embedding of the vertices to a real embedding, it immediately follows from the definition that $h_{\mathbb{R}}(G)$ is at

most $h(G)$. The optimal solution of the optimization problem above provides a real embedding of the graph. Two questions need to be addressed: Is $h_{\mathbb{R}}(G)$ efficiently computable? How small can $h_{\mathbb{R}}(G)$ get when compared to $h(G)$; in other words, how good an approximation is $h_{\mathbb{R}}(G)$ to $h(G)$? In the remainder of the section we address the first problem. We show that computing $h_{\mathbb{R}}(G)$ reduces to computing an eigenvalue of a matrix, in fact closely related to the graph Laplacian. In the next section we lower bound $h_{\mathbb{R}}(G)$ by a function in $h(G)$ and present an algorithm that finds a reasonable cut using $h_{\mathbb{R}}$.

5.3 The Normalized Laplacian and Its Second Eigenvalue

Recall that $h_{\mathbb{R}}(G) \stackrel{\text{def}}{=} \min_{\mathbf{x}\in\mathbb{R}^n} \frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i-x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i-x_j)^2]}$. Note that the r.h.s. above remains unchanged if we add or subtract the same quantity to every x_i. One can thereby subtract $\mathbb{E}_{i\leftarrow\mu}[x_i]$ from every x_i, and assume that we optimize with an additional constraint on $\mathbf{x}$, namely, $\mathbb{E}_{i\leftarrow\mu}[x_i] = 0$. This condition can be written as $\langle \mathbf{x}, D\mathbf{1}\rangle = 0$. First note the following simple series of equalities for any $\mathbf{x} \in \mathbb{R}^n$ based on simple properties of expectations.

$$\begin{aligned}
\mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2] &= \mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[{x_i}^2 + {x_j}^2 - 2x_ix_j] \\
&= \mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[{x_i}^2 + x_j^2] - 2\mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[x_ix_j] \\
&= \mathop{\mathbb{E}}_{i\leftarrow\mu}[{x_i}^2 + x_j^2] - 2\mathop{\mathbb{E}}_{i\leftarrow\mu}[x_i]\mathop{\mathbb{E}}_{j\leftarrow\mu}[x_j] \\
&= 2\mathop{\mathbb{E}}_{i\leftarrow\mu}[{x_i}^2] - 2(\mathop{\mathbb{E}}_{i\leftarrow\mu}[x_i])^2.
\end{aligned}$$

Therefore, by our assumption, $\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2] = 2\,\mathbb{E}_{i\leftarrow\mu}[{x_i}^2]$. Hence, we obtain

$$\begin{aligned}
\mathop{\mathbb{E}}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2] &= 2\mathop{\mathbb{E}}_{i\leftarrow\mu}[x_i^2] \\
&= \frac{2\sum_{i\in V} d_i {x_i}^2}{\mathrm{vol}(G)} \\
&= \frac{2\mathbf{x}^\top D\mathbf{x}}{\mathrm{vol}(G)}.
\end{aligned}$$

Moreover, from the definition of L it follows that

$$\begin{aligned}\mathop{\mathbb{E}}_{ij\leftarrow\nu}\left[(x_i - x_j)^2\right] &= \frac{\sum_{e=ij\in E}(x_i - x_j)^2}{m} \\ &= \frac{\mathbf{x}^\top L\mathbf{x}}{m} = \frac{2\mathbf{x}^\top L\mathbf{x}}{\mathrm{vol}(G)}.\end{aligned}$$

Therefore, we can write

$$h_{\mathbb{R}}(G) = \min_{\mathbf{x}\in\mathbb{R}^n,\, \langle \mathbf{x}, D\mathbf{1}\rangle = 0} \frac{\mathbf{x}^\top L\mathbf{x}}{\mathbf{x}^\top D\mathbf{x}}.$$

This is not quite an eigenvalue problem. We will now reduce it to one. Substitute $\mathbf{y} \stackrel{\text{def}}{=} D^{1/2}\mathbf{x}$ in the equation above. Then,

$$\begin{aligned}h_{\mathbb{R}}(G) &= \min_{\mathbf{y}\in\mathbb{R}^n,\, \langle D^{-1/2}\mathbf{y}, D\mathbf{1}\rangle = 0} \frac{(D^{-1/2}\mathbf{y})^\top L(D^{-1/2}\mathbf{y})}{(D^{-1/2}\mathbf{y})^\top D(D^{-1/2}\mathbf{y})} \\ &= \min_{\mathbf{y}\in\mathbb{R}^n,\, \langle \mathbf{y}, D^{1/2}\mathbf{1}\rangle = 0} \frac{\mathbf{y}^\top D^{-1/2}LD^{-1/2}\mathbf{y}}{\mathbf{y}^\top\mathbf{y}}.\end{aligned}$$

This is an eigenvalue problem for the following matrix which we refer to as the normalized Laplacian.

Definition 5.4. The normalized Laplacian of a graph G is defined to be

$$\mathcal{L} \stackrel{\text{def}}{=} D^{-1/2}LD^{-1/2},$$

where D is the degree matrix and L is the graph Laplacian as in Section 2.

Note that $\mathcal{L}$ is symmetric and, hence, has a set of orthonormal eigenvectors which we denote $D^{1/2}\mathbf{1} = \mathbf{u}_1, \ldots, \mathbf{u}_n$ with eigenvalues $0 = \lambda_1(\mathcal{L}) \leq \cdots \leq \lambda_n(\mathcal{L})$. Hence,

$$h_{\mathbb{R}}(G) = \min_{\mathbf{y}\in\mathbb{R}^n, \langle \mathbf{y}, D^{1/2}\mathbf{1}\rangle = 0} \frac{\mathbf{y}^\top\mathcal{L}\mathbf{y}}{\mathbf{y}^\top\mathbf{y}} = \lambda_2(\mathcal{L}).$$

We summarize what we have proved in the following theorem.

Theorem 5.3. $\lambda_2(\mathcal{L}) = h_{\mathbb{R}}(G)$.

We also obtain the following corollary.

Corollary 5.4. $\lambda_2(\mathcal{L}) \leq 2\phi(G)$.

Thus, we have a handle on the quantity $\lambda_2(\mathcal{L})$, which we can compute efficiently. Can we use this information to recover a cut in G of conductance close to $\lambda_2(\mathcal{L})$? It turns out that the eigenvector corresponding to $\lambda_2(\mathcal{L})$ can be used to find a cut of small conductance, which results in an approximation algorithm for Sparsest Cut. This is the content of the next section.

Notes

For a good, though out-dated, survey on the graph partitioning problem and its applications see [74]. There has been a lot of activity on and around this problem in the last decade. In fact, an important component of the original proof of Theorem 3.1 by Spielman and Teng relied on being able to partition graphs in near-linear-time, see [79]. The proof of NP-hardness of Sparsest Cut can be found in [32]. Theorem 5.3 and Corollary 5.4 are folklore and more on them can be found in the book [23].

6

Graph Partitioning II
A Spectral Algorithm for Conductance

This section shows that the conductance of a graph can be roughly upper bounded by the square root of the second eigenvalue of the normalized Laplacian. The proof that is presented implies an algorithm to find such a cut from the second eigenvector.

6.1 Sparse Cuts from ℓ_1 Embeddings

Recall the ℓ_2^2 problem which arose from the relaxation of $h(G)$ and Theorem 5.3[1]

$$\lambda_2(\mathcal{L}) = \min_{\mathbf{x}\in\mathbb{R}^n} \frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i - x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2]} \leq h(G) \leq 2\phi(G).$$

We will relate the mathematical program that captures graph conductance with an ℓ_1-minimization program (Theorem 6.1) and then relate that ℓ_1-minimization to the ℓ_2^2 program (Theorem 6.3). In particular,

[1] Here by ℓ_2^2 we mean that it is an optimization problem where both the numerator and the denominator are squared-Euclidean distances. This is not to be confused with an ℓ_2^2 metric space.

we will show that

$$\phi(G) = \min_{\substack{\mathbf{y}\in\mathbb{R}^n \\ \mu_{1/2}(\mathbf{y})=0}} \frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]} \leq 2\sqrt{\lambda_2(\mathcal{L})}.$$

Here, $\mu_{1/2}(\mathbf{y})$ is defined to be a t such that $\mu(\{i : y_i < t\}) \leq 1/2$ and $\mu(\{i : y_i > t\}) \leq 1/2$. Note that $\mu_{1/2}(\cdot)$ is not uniquely defined.

Theorem 6.1. For any graph G on n vertices, the graph conductance

$$\phi(G) = \min_{\mathbf{y}\in\mathbb{R}^n, \mu_{1/2}(\mathbf{y})=0} \frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]}.$$

Proof. Let $(S, \bar{S})$ be such that $\phi(S) = \phi(G)$. Without loss of generality (W.l.o.g.) assume that $\mu(S) \leq \mu(\bar{S})$. Then $\mathbb{E}_{ij\leftarrow\nu}[|1_S(i) - 1_S(j)|] = \nu(E(S, \bar{S}))$ and $\mathbb{E}_{i\leftarrow\mu}[|1_S(i)|] = \mu(S) = \min\{\mu(S), \mu(\bar{S})\}$. Since $\{i : 1_S(i) \leq t\}$ is $\emptyset$ for $t < 0$ and $\bar{S}$ for $t = 0$, we have $\mu_{1/2}(\mathbf{1}_S) = 0$. Combining, we obtain

$$\min_{\mathbf{y}\in\mathbb{R}^n, \mu_{1/2}(\mathbf{y})=0} \frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]} \leq \frac{\mathbb{E}_{ij\leftarrow\nu}[|1_S(i) - 1_S(j)|]}{\mathbb{E}_{i\leftarrow\mu}[|1_S(i)|]} = \phi(G).$$

It remains to show that $\frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]} \geq \phi(G)$ for every $\mathbf{y} \in \mathbb{R}^n$ such that $\mu_{1/2}(\mathbf{y}) = 0$. Fix an arbitrary $\mathbf{y} \in \mathbb{R}$ with $\mu_{1/2}(\mathbf{y}) = 0$. For convenience we assume that all entries of $\mathbf{y}$ are distinct. Re-index the vertices of G such that $y_1 < y_2 < \cdots < y_n$. This gives an embedding of the vertices of G on $\mathbb{R}$. The natural cuts on this embedding, called *sweep cuts*, are given by $S_i \stackrel{\text{def}}{=} \{1, \ldots, i\}, 1 \leq i \leq n-1$. Let $\widehat{S}$ be the sweep cut with minimum conductance. We show that the conductance $\phi(\widehat{S})$ is a lower bound on $\frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]}$. This completes the proof since $\phi(\widehat{S})$ itself is bounded below by $\phi(G)$. First, note that

$$\begin{aligned}
\mathbb{E}_{ij\leftarrow\nu}[|y_i - y_j|] &= \frac{1}{m}\sum_{ij\in E} |y_i - y_j| \\
&= \frac{1}{m}\sum_{ij\in E} \sum_{l=\min\{i,j\}}^{\max\{i,j\}-1} (y_{l+1} - y_l) \\
&= \sum_{l=1}^{n-1} \nu(E(S_l, \bar{S}_l))(y_{l+1} - y_l). \qquad (6.1)
\end{aligned}$$

The third equality above follows since, in the double summation, the term $(y_{l+1} - y_l)$ is counted once for every edge that crosses $(S_l, \bar{S}_l)$. For sake of notational convenience we assign $S_0 \stackrel{\text{def}}{=} \emptyset$ and $S_n \stackrel{\text{def}}{=} V$ as two limiting sweep cuts so that $\{i\} = S_i - S_{i-1} = \bar{S}_{i-1} - \bar{S}_i$ for every $i \in [n]$. Hence $\mu_i = \mu(S_i) - \mu(S_{i-1}) = \mu(\bar{S}_{i-1}) - \mu(\bar{S}_i)$. We use this to express $\mathbb{E}_{i \leftarrow \mu}[|y_i|]$ below. Assume that there is a k such that $y_k = 0$. Thus, since $\mu_{1/2}(\mathbf{y}) = 0$, $\mu(\bar{S}_k) \leq \mu(S_k)$. If no such k exists, the proof is only simpler.

$$\begin{aligned}
\mathbb{E}_{i \leftarrow \mu}[|y_i|] &= \sum_{i=1}^{n} \mu_i |y_i| = \sum_{i=1}^{k} \mu_i(-y_i) + \sum_{i=k+1}^{n} \mu_i(y_i) \\
&= \sum_{i=1}^{k} (\mu(S_i) - \mu(S_{i-1}))(-y_i) + \sum_{i=k+1}^{n} (\mu(\bar{S}_{i-1}) - \mu(\bar{S}_i))(y_i) \\
&= \mu(S_0)y_1 + \sum_{i=1}^{k-1} \mu(S_i)(y_{i+1} - y_i) + \mu(S_k)(-y_k) + \\
&\quad \mu(\bar{S}_k)y_{k+1} + \sum_{i=k+1}^{n-1} \mu(\bar{S}_i)(y_{i+1} - y_i) + \mu(\bar{S}_n)(-y_n) \\
&= \sum_{i=1}^{k-1} \mu(S_i)(y_{i+1} - y_i) + \sum_{i=k}^{n-1} \mu(\bar{S}_i)(y_{i+1} - y_i) \\
&\quad (\text{since } \mu(S_0) = \mu(\bar{S}_n) = 0,\ y_k = 0 \text{ and } \mu(\bar{S}_k) \leq \mu(S_k)) \\
&= \sum_{i=1}^{n-1} \min\{\mu(S_i), \mu(\bar{S}_i)\}(y_{i+1} - y_i). \qquad (6.2)
\end{aligned}$$

Where the last equality follows since $\mu(\bar{S}_i) \geq \mu(S_i)$ for all $i \leq k$ and $\mu(\bar{S}_i) \leq \mu(S_i)$ for all $i \geq k$. Putting Equations (6.1) and (6.2) together, we get the desired inequality:

$$\frac{\mathbb{E}_{ij \leftarrow \nu}[|y_i - y_j|]}{\mathbb{E}_{i \leftarrow \mu}[|y_i|]} = \frac{\sum_{i=1}^{n-1} \nu(E(S_i, \bar{S}_i))(y_{i+1} - y_i)}{\sum_{i=1}^{n-1} \min\{\mu(S_i), \mu(\bar{S}_i)\}(y_{i+1} - y_i)}$$

$$\begin{aligned} & \overset{\text{Prop. 6.2}}{\geq} \min_{i\in[n-1]} \left\{ \frac{\nu(E(S_i,\bar{S}_i)}{\min\{\mu(S_i),\mu(\bar{S}_i)\}} \right\} \\ &= \min_{i\in[n-1]} \phi(S_i) \\ &= \phi(\widehat{S}). \end{aligned}$$

□

In the proof we have used the following simple proposition.

Proposition 6.2. For $b_1,\ldots,b_n \geq 0$,

$$\frac{a_1+\cdots+a_n}{b_1+\cdots+b_n} \geq \min_{i=1}^{n} \frac{a_i}{b_i}.$$

Proof. Let $\min_i {}^{a_i}/_{b_i} = \theta$. Then, since $b_i \geq 0$, for all i, $a_i \geq \theta \cdot b_i$. Hence, $\sum_i a_i \geq \theta \cdot \sum_i b_i$. Thus, $\sum_i {}^{a_i}/_{\sum_i b_i} \geq \theta$. □

6.2 An ℓ_1 Embedding from an ℓ_2^2 Embedding

The next theorem permits us to go from an ℓ_2^2 solution to an ℓ_1 solution which satisfies the conditions of Theorem 6.1 with a quadratic loss in the objective value.

Theorem 6.3. If there is an $\mathbf{x} \in \mathbb{R}^n$ such that $\frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i-x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i-x_j)^2]} = \varepsilon$, then there exists a $\mathbf{y} \in \mathbb{R}^n$ with $\mu_{1/2}(\mathbf{y}) = 0$ such that $\frac{\mathbb{E}_{ij\leftarrow\nu}[|y_i-y_j|]}{\mathbb{E}_{i\leftarrow\mu}[|y_i|]} \leq 2\sqrt{\varepsilon}$.

Before we prove this theorem, we will prove a couple simple propositions which are used in its proof. For $y \in \mathbb{R}$, let $\operatorname{sgn}(y) = 1$ if $y \geq 0$, and -1 otherwise.

Proposition 6.4. For all $y \geq z \in \mathbb{R}$,

$$\left|\operatorname{sgn}(y)\cdot y^2 - \operatorname{sgn}(z)\cdot z^2\right| \leq (y-z)(|y|+|z|).$$

Proof.

(1) If $\operatorname{sgn}(y) = \operatorname{sgn}(z)$, then $|\operatorname{sgn}(y) \cdot y^2 - \operatorname{sgn}(z) \cdot z^2| = |y^2 - z^2| = (y - z) \cdot |y + z| = (y - z)(|y| + |z|)$ as $y \geq z$.
(2) If $\operatorname{sgn}(y) \neq \operatorname{sgn}(y)$, then $y \geq z$, $(y - z) = |y| + |z|$. Hence, $|\operatorname{sgn}(y) \cdot y^2 - \operatorname{sgn}(z) \cdot z^2| = y^2 + z^2 \leq (|y| + |z|)^2 = (y - z)(|y| + |z|)$.

□

Proposition 6.5. For $a, b \in \mathbb{R}$, $(a + b)^2 \leq 2(a^2 + b^2)$.

Proof. Observe that $a^2 + b^2 - 2ab \geq 0$. Hence, $2a^2 + 2b^2 - 2ab \geq a^2 + b^2$. Hence,

$$2a^2 + 2b^2 \geq a^2 + 2ab + b^2 = (a + b)^2.$$

□

Proof. [of Theorem 6.3] Since the left-hand side (l.h.s.) of the hypothesis is shift invariant for $\mathbf{x}$, we can assume w.l.o.g. that $\mu_{1/2}(\mathbf{x}) = 0$. Let $\mathbf{y} = (y_1, \ldots, y_n)$ be defined such that

$$y_i \stackrel{\text{def}}{=} \operatorname{sgn}(x_i) x_i^2.$$

Hence, $\mu_{1/2}(\mathbf{y}) = 0$. Further,

$$\begin{aligned}
\mathbb{E}_{ij \leftarrow \nu}[|y_i - y_j|] &\overset{\text{Prop. 6.4}}{\leq} \mathbb{E}_{ij \leftarrow \nu}[|x_i - x_j|(|x_i| + |x_j|)] \\
&\overset{\text{Cauchy-Schwarz}}{\leq} \sqrt{\mathbb{E}_{ij \leftarrow \nu}[|x_i - x_j|^2]} \sqrt{\mathbb{E}_{ij \leftarrow \nu}[(|x_i| + |x_j|)^2]} \\
&\overset{\text{Prop. 6.5}}{\leq} \sqrt{\mathbb{E}_{ij \leftarrow \nu}[(x_i - x_j)^2]} \sqrt{2\mathbb{E}_{ij \leftarrow \nu}[{x_i}^2 + {x_j}^2]} \\
&\overset{\text{double counting}}{=} \sqrt{\mathbb{E}_{ij \leftarrow \nu}[(x_i - x_j)^2]} \sqrt{2\mathbb{E}_{i \leftarrow \mu}[x_i^2]} \\
&= \sqrt{\mathbb{E}_{ij \leftarrow \nu}[(x_i - x_j)^2]} \sqrt{2\mathbb{E}_{i \leftarrow \mu}[|y_i|]}.
\end{aligned}$$

Hence,

$$\frac{\mathbb{E}_{ij \leftarrow \nu}[|y_i - y_j|]}{\mathbb{E}_{i \leftarrow \mu}[|y_i|]} \leq \sqrt{\frac{2\mathbb{E}_{ij \leftarrow \nu}[(x_i - x_j)^2]}{\mathbb{E}_{i \leftarrow \mu}[|y_i|]}}$$

$$
\begin{aligned}
&= \sqrt{\frac{2\mathbb{E}_{ij\leftarrow\nu}[(x_i - x_j)^2]}{\mathbb{E}_{i\leftarrow\mu}[x_i^2]}} \\
&\leq \sqrt{\frac{4\mathbb{E}_{ij\leftarrow\nu}[(x_i - x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2]}} \\
&\qquad (\text{since } \mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2] \leq 2\mathbb{E}_{i\leftarrow\mu}[x_i^2]) \\
&= 2\sqrt{\varepsilon}. \qquad \square
\end{aligned}
$$

As a corollary we obtain the following theorem known as Cheeger's inequality.

Theorem 6.6. For any graph G,

$$\frac{\lambda_2(\mathcal{L})}{2} \leq \phi(G) \leq 2\sqrt{2\lambda_2(\mathcal{L})}.$$

Proof. The first inequality was proved in the last section, see Corollary 5.4. The second inequality follows from Theorems 6.1 and 6.3 by choosing $\mathbf{x}$ to be the vector that minimizes $\frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i - x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2]}$. $\square$

It is an easy exercise to show that Cheeger's inequality is tight up to constant factors for the cycle on n vertices.

Algorithm 6.1 SpectralCut

Input: $G(V, E)$, an undirected graph with $V = \{1, \ldots, n\}$
Output: S, a subset of V with conductance $\phi(S) \leq 2\sqrt{\phi(G)}$

1: $D \leftarrow$ degree matrix of G
2: $L \leftarrow$ Laplacian of G
3: $\mathcal{L} \leftarrow D^{-1/2} L D^{-1/2}$
4: $\mathbf{y}^\star \leftarrow \arg\min_{\mathbf{y}\in\mathbb{R}^n : \langle \mathbf{y}, D^{1/2}\mathbf{1}\rangle = 0} \frac{\mathbf{y}^\top \mathcal{L} \mathbf{y}}{\mathbf{y}^\top \mathbf{y}}$
5: $\mathbf{x}^\star \leftarrow D^{-1/2}\mathbf{y}^\star$
6: Re-index V so that $x_1^\star \leq x_2^\star \leq \cdots \leq x_n^\star$
7: $S_i \leftarrow \{1, \ldots, i\}$ for $i = 1, 2, \ldots, n-1$
8: Return S_i that has minimum conductance from among $1, \ldots, n-1$

The spectral algorithm for Sparsest Cut is given in Algorithm 6.1. Note that $\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i-x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i-x_j)^2]}$ is computed by solving an eigenvector problem on the normalized Laplacian $\mathcal{L}$. The proof of correctness follows from Theorems 5.3 and 6.6.

Notes

Theorem 6.6 is attributed to [7, 20]. There are several ways to prove Theorem 6.3 and the proof presented here is influenced by the work of [58].

In terms of approximation algorithms for Sparsest Cut, the best result is due to [13] who gave an $O(\sqrt{\log n})$-factor approximation algorithm. Building on a sequence of work [10, 12, 48, 61], Sherman [71] showed how to obtain this approximation ratio of $\sqrt{\log n}$ in essentially single-commodity flow time. Now there are improved algorithms for this flow problem, see Section 12, resulting in an algorithm that runs in time $\tilde{O}(m^{4/3})$. Obtaining a $\sqrt{\log n}$ approximation in $\tilde{O}(m)$ time, possibly bypassing the reduction to single-commodity flows, remains open. Toward this, getting a $\log n$ factor approximation in $\tilde{O}(m)$ seems like a challenging problem. Finally, it should be noted that Madry [55] gave an algorithm that, for every integer $k \geq 1$, achieves, roughly, an $O((\log n)^{(k+1/2)})$ approximation in $\tilde{O}(m + 2^k \cdot n^{1+2^{-k}})$ time.

7

Graph Partitioning III
Balanced Cuts

This section builds up on Section 6 to introduce the problem of finding sparse cuts that are also balanced. Here, a balance parameter $b \in (0, 1/2]$ is given and the goal is to find the cut of least conductance among cuts whose smaller side has a fractional volume of at least b. We show how one can recursively apply Algorithm 6.1 to solve this problem.

7.1 The Balanced Edge-Separator Problem

In many applications, we not only want to find the cut that minimizes conductance, but would also like the two sides of the cut to be comparable in volume. Formally, given a graph G and a balance parameter $b \in (0, 1/2]$, the goal is to find a cut $(S, \bar{S})$ with minimum conductance such that $\min\{\mu(S), \mu(\bar{S})\} \geq b$. This problem, referred to as Balanced Edge-Separator, is also NP-hard and we present an approximation algorithm for it in this section. In fact, we recursively use the algorithm SpectralCut presented in the previous section to give a *pseudo*-approximation algorithm to this problem: We present an algorithm (BalancedCut, see Algorithm 7.1) that accepts an undirected graph G, a balance requirement $b \in (0, 1/2]$, and a conductance requirement $\gamma \in (0, 1)$ as input. If the graph contains

Algorithm 7.1 BalancedCut

Input: $G(V,E)$, an undirected graph with $V = \{1,\ldots,n\}$. A balance parameter $b \in (0, 1/2]$. A target conductance parameter $\gamma \in (0,1)$.

Output: Either a $b/2$-balanced cut $(S,\bar{S})$ with $\phi(S) \leq 4\sqrt{\gamma}$ or certify that every b-balanced cut in G has conductance at least γ.

1: $\mu_G \leftarrow$ degree measure of G
2: $H \leftarrow G$
3: $S' \leftarrow \emptyset$
4: **repeat**
5: $\quad\mathcal{L}_H \leftarrow$ normalized Laplacian of H
6: $\quad$**if** $\lambda_2(\mathcal{L}_H) > 4\gamma$ **then**
7: $\quad\quad$**return** NO
8: $\quad$**end if**
9: $\quad S \leftarrow$ SpectralCut(H)
10: $\quad S' \leftarrow \arg\min\{\mu_G(S), \mu_G(V(H) \setminus S)\} \cup S'$
11: $\quad$**if** $\min\{\mu_G(S'), \mu_G(V(G) \setminus S')\} \geq b/2$ **then**
12: $\quad\quad$**return** $(S',\bar{S}')$ and $(S,\bar{S})$ *// At least one of the cuts will be shown to satisfy the output requirement of being $b/2$ balanced and conductance at most $4\sqrt{\gamma}$.*
13: $\quad$**else** *// Restrict the problem to the induced subgraph of H on $V \setminus S$ with every edge that crosses $V \setminus S$ to S replaced by a self-loop at the $V \setminus S$ end*
14: $\quad\quad V' \leftarrow V(H) \setminus S$
15: $\quad\quad E' \leftarrow \{ij : i,j \in V'\} \cup \{ii : ij \in E(V',S)\}$ *// Note that E' is a multiset*
16: $\quad\quad H = (V',E')$
17: $\quad$***end if***
18: **until**

a b-balanced cut of conductance at most γ, then BalancedCut will return a $b/2$-balanced cut of conductance $O(\sqrt{\gamma})$, otherwise it will certify that every b-balanced cut in G has conductance at least γ. This $b/2$ can be improved to $(1-\varepsilon)b$ for any $\varepsilon > 0$. Assuming this, even if the algorithm outputs a $(1-\varepsilon)b$ balanced cut, the cut with least conductance with this volume could be significantly smaller than γ. In this sense, the algorithm is a pseudo-approximation.

7.2 The Algorithm and Its Analysis

The following is the main theorem of this section.

Theorem 7.1. Given an undirected graph G, a balance parameter $b \in (0, 1/2]$ and a target conductance $\gamma \in (0, 1)$, BalancedCut either finds a $b/2$-balanced cut in G of conductance at most $O(\sqrt{\gamma})$, or certifies that every b-balanced cut in G has conductance more than γ.

The algorithm BalancedCut, which appears below, starts by checking if the second smallest eigenvalue of the normalized Laplacian of G, $\mathcal{L}$ is at most $O(\gamma)$, the target conductance. If not, then it outputs NO. On the other hand, if $\lambda_2(\mathcal{L}) \leq O(\gamma)$, it makes a call to SpectralCut which returns a cut $(S_0, \bar{S}_0)$ of conductance at most $O(\sqrt{\gamma})$ by Theorem 6.3. Assume that $\mu(S_0) \leq \mu(\bar{S}_0)$. If $(S_0, \bar{S}_0)$ is already $b/2$-balanced, we return $(S_0, \bar{S}_0)$. Otherwise, we construct a smaller graph G_1 by removing S_0 from G and replacing each edge that had crossed the cut with a self-loop at its endpoint in $\bar{S}_0$. We always remove the smaller side of the cut output by SpectralCut from the current graph. This ensures that the total number of edges incident on any subset of vertices of G_1 is the same as it was in G, i.e., the volume of a subset does not change in subsequent iterations.

BalancedCut recurses on G_1 and does so until the relative volume of the union of all the deleted vertices exceeds $b/2$ for the first time. In this case, we are able to recover a $b/2$-balanced cut of conductance at most $O(\sqrt{\gamma})$. It is obvious that BalancedCut either outputs an NO or stops with an output of pair of cuts $(S, \bar{S})$ and $(S', \bar{S}')$. Since the graph is changing in every iteration, in the description of the algorithm and the analysis we keep track of the degree measure μ and the normalized Laplacian $\mathcal{L}$ by subscripting it with the graph involved. The following two lemmata imply Theorem 7.1.

Lemma 7.2. If BalancedCut deletes a set of degree measure less than $b/2$ before it returns an NO, then every b-balanced cut in G has conductance at least γ.

Proof. When BalancedCut stops by returning an NO, we know that

(1) $\lambda_2(\mathcal{L}_H) > 4\gamma$, and
(2) $\text{vol}(V(H)) \geq (1 - b/2)\text{vol}(G)$.

Suppose for sake of contradiction that there is a b-balanced cut in G of conductance at most γ. Then, such a cut when restricted to H has relative volume at least $b/2$, and the number of edges going out of it in H is at most those going out in G. Thus, since the volume has shrunk by a factor of at most 2 and the edges have not increased, the conductance of the cut in H is at most 2 times that in G. Hence, such a cut will have conductance at most 2γ in H. But $\lambda_2(\mathcal{L}_H) > 4\gamma$ and hence, from the easy side of Cheeger's inequality (Corollary 5.4), we can infer that $\phi(H)$, even with degrees from G, is at least $\lambda_2(\mathcal{L}_H)/2 > 2\gamma$. Thus, H does not have a cut of conductance at most 2γ. □

Lemma 7.3. When BalancedCut terminates by returning a pair of cuts, one of the two cuts it outputs has a relative volume at least $b/2$ and conductance at most $4\sqrt{\gamma}$.

Proof. There are two cases at the time of termination: If $(S', \bar{S}')$ and $(S, \bar{S})$ are returned, either

(1) $\text{vol}(S')/\text{vol}(G) \geq 1/2$, or
(2) $b/2 \leq \text{vol}(S')/\text{vol}(G) \leq 1/2$.

In the first case, since we stopped the first time the relative volume exceeded $b/2$ (and in fact went above $1/2$), it must be the case that output $(S, \bar{S})$ of SpectralCut in that iteration is $b/2$-balanced. The conductance of this cut in H is at most $2\sqrt{4\gamma} = 4\sqrt{\gamma}$ by the guarantee of SpectralCut. Hence, the lemma is satisfied.

In the second case, we have that $(S', \bar{S}')$ is $b/2$-balanced. What about its conductance? S', the smaller side, consists of the union of disjoint cuts $S_0, S_1, \ldots, S_t$ for some t and each is of conductance at most $4\sqrt{\gamma}$ in the respective G_i by the guarantee of SpectralCut. We argue that the conductance of S' is also at most $4\sqrt{\gamma}$ in G. First note that

$$|E_G(S', \bar{S}')| \leq |E_G(S_0, \bar{S}_0)| + |E_{G_1}(S_1, \bar{S}_1)| + \cdots + |E_{G_t}(S_t, \bar{S}_t)|.$$

This is because every edge contributing to the l.h.s. also contributes to the r.h.s. The inequality occurs because there could be more, e.g., if there is an edge going between S_1 and S_2. Now, the conductance of each S_i in G_i is at most $4\sqrt{\gamma}$. Hence,

$$|E_G(S', \bar{S}')| \leq 4\sqrt{\gamma} \cdot \mathrm{vol}_G(S_0) + 4\sqrt{\gamma} \cdot \mathrm{vol}_{G_1}(S_1) + \cdots + 4\sqrt{\gamma} \cdot \mathrm{vol}_{G_t}(S_t).$$

Finally, note that $\mathrm{vol}_{G_i}(S_i) = \mathrm{vol}_G(S_i)$ for all i due to the way in which we split the graph but retain the edges adjacent to the removed edges as self-loops. Hence, $\mathrm{vol}_G(S') = \sum_{i=0}^{t} \mathrm{vol}_{G_i}(S_i)$. Thus,

$$\frac{|E_G(S', \bar{S}')|}{\mathrm{vol}_G(S')} \leq 4\sqrt{\gamma}.$$

This concludes the proof of Theorem 7.1. □

Notes

Theorem 7.1 is folklore and appears implicitly in [79] and [13]. There are other nearly-linear time algorithms, where we allow the running time to depend on $1/\gamma$ for the BalancedEdge-Separator problem which achieve a γ versus $\sqrt{\gamma \operatorname{polylog} n}$ bound, see for instance [8, 9, 79]. These algorithms are *local* in the sense that they instead bound the total work by showing that each iteration runs in time proportional to the smaller side of the cut. This approach was pioneered in [79] which relied on mixing time results in [53]. The first near-linear algorithm for BalancedEdge-Separator that removes polylog n factors in the approximation was obtained by [62]. Recently, this dependence of the running time on γ has also removed and an $\tilde{O}(m)$ time algorithm, which achieves a γ versus $\sqrt{\gamma}$ bound, has been obtained by Orecchia et al. [60]. Their proof relies crucially on Laplacian solvers and highlights of their algorithm are presented in Sections 9 and 19.

8

Graph Partitioning IV
Computing the Second Eigenvector

This final section on graph partitioning addresses the problem of how to compute an approximation to the second eigenvector, using the near-linear-time Laplacian solver, a primitive that was needed in Sections 6 and 7.

8.1 The Power Method

Before we show how to compute the second smallest eigenvector of the normalized Laplacian of a graph, we present the power method. This is a well-known method to compute an approximation to the largest eigenvector of a matrix A. It start with a random unit vector and repeatedly applies A to it while normalizing it at every step.

Lemma 8.1. Given a symmetric matrix $A \in \mathbb{R}^{n \times n}$, an error parameter $\varepsilon > 0$ and a positive integer $k > 1/2\varepsilon \log(9n/4)$, the following holds with probability at least $1/2$ over a vector $\mathbf{v}$ chosen uniformly at random from the n-dimensional unit sphere,

$$\frac{\|A^{k+1}\mathbf{v}\|}{\|A^k\mathbf{v}\|} \geq (1-\varepsilon)|\lambda_n(A)|,$$

where $\lambda_n(A)$ is the eigenvalue with the largest magnitude of A.

Proof. Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be the eigenvectors of A corresponding to the eigenvalues $|\lambda_1| \leq \cdots \leq |\lambda_n|$. We can write the random unit vector $\mathbf{v}$ in the basis $\{\mathbf{u}_i\}_{i\in[n]}$ as $\sum_{i=1}^{n} \alpha_i \mathbf{u}_i$. From a standard calculation involving Gaussian distributions we deduce that, with probability at least $1/2$, $|\alpha_n| \geq \frac{2}{3\sqrt{n}}$. Using Hölder's inequality,[1]

$$\begin{aligned}
\|A^k\mathbf{v}\|^2 &= \sum_i \alpha_i^2 \lambda_i^{2k} \\
&\leq \left(\sum_i \alpha_i^2 \lambda_i^{2k+2}\right)^{k/k+1} \left(\sum_i \alpha_i^2\right)^{1/k+1} \\
&= \left(\sum_i \alpha_i^2 \lambda_i^{2k+2}\right)^{k/k+1} \\
&= \|A^{k+1}\mathbf{v}\|^{2k/k+1}.
\end{aligned}$$

Note that $\|A^{k+1}\mathbf{v}\|^{2/k+1} \geq \alpha_n^{2/k+1}\lambda_n^2$. Thus, it follows that

$$\|A^k\mathbf{v}\|^2 \leq \|A^{k+1}\mathbf{v}\|^{2k/k+1} \cdot \frac{\|A^{k+1}\mathbf{v}\|^{2/k+1}}{\alpha_n^{2/k+1}\lambda_n^2} = \frac{\|A^{k+1}\mathbf{v}\|^2}{\alpha_n^{2/k+1}\lambda_n^2}.$$

Substituting $k+1 \geq \frac{1}{2\varepsilon}\log(9n/4)$ in the r.h.s. above gives $|\alpha_n|^{1/k+1} \geq e^{-\varepsilon} \geq 1-\varepsilon$, completing the proof. □

8.2 The Second Eigenvector via Powering

The following is the main theorem of this section.

Theorem 8.2. Given an undirected, unweighted graph G on n vertices and m edges, there is an algorithm that outputs a vector $\mathbf{x}$ such that

$$\frac{\mathbb{E}_{ij\leftarrow\nu}[(x_i - x_j)^2]}{\mathbb{E}_{(i,j)\leftarrow\mu\times\mu}[(x_i - x_j)^2]} \leq 2\lambda_2(\mathcal{L})$$

and runs in $\tilde{O}(m+n)$ time. Here $\mathcal{L}$ is the normalized Laplacian of G.

---

[1] For vectors $\mathbf{u}, \mathbf{v}$, and $p, q \geq 0$ such that $1/p + 1/q = 1$, $|\langle \mathbf{u}, \mathbf{v}\rangle| \leq \|\mathbf{u}\|_p \|\mathbf{v}\|_q$. In this application we choose $p \stackrel{\text{def}}{=} k+1, q \stackrel{\text{def}}{=} {}^{k+1}/_k$ and $u_i \stackrel{\text{def}}{=} \alpha_i^{2/k+1}, v_i \stackrel{\text{def}}{=} \alpha_i^{2k/k+1}\lambda_i^{2k}$.

8.2.1 The Trivial Application

Let us see what the power method in Lemma 8.1 gives us when we apply it to $I - \mathcal{L}$, where $\mathcal{L}$ is the normalized Laplacian of a graph G. Recall that $\mathcal{L} = I - D^{-1/2}AD^{-1/2}$, where D is the degree matrix of G and A its adjacency matrix. Since we know the largest eigenvector for $D^{-1/2}AD^{-1/2}$ explicitly, i.e., $D^{1/2}\mathbf{1}$, we can work orthogonal to it by removing the component along it from the random starting vector. Thus, the power method converges to the second largest eigenvector of $D^{-1/2}AD^{-1/2}$, which is the second smallest eigenvector of $\mathcal{L}$. Hence, if we apply the power method to estimate $\lambda_2(\mathcal{L})$, we need $\varepsilon \sim \lambda_2(\mathcal{L})/2$ in order to be able to approximate $\lambda_2(\mathcal{L})$ up to a factor of 2. This requires the computation of $(D^{-1/2}AD^{-1/2})^k\mathbf{v}$ for $k = \Theta(\log n/\lambda_2(\mathcal{L}))$, which gives a time bound of $O(m \log n/\lambda_2(\mathcal{L}))$ since $\lambda_2(\mathcal{L})$ can be as small as $1/m$ and, hence, k may be large. In the next section we see how to remove this dependency on $1/\lambda_2(\mathcal{L})$ and obtain an algorithm which runs in time $\tilde{O}(m)$ and computes a vector that approximates $\lambda_2(\mathcal{L})$ to within a factor of 2.

8.2.2 Invoking the Laplacian Solver

Start by observing that $\lambda_2 \stackrel{\text{def}}{=} \lambda_2(\mathcal{L})$ is the largest eigenvalue of $\mathcal{L}^+$, the pseudo-inverse of $\mathcal{L}$. Thus, if we could compute $\mathcal{L}^+\mathbf{u}$, for a vector $\mathbf{u}$ which is orthogonal to $D^{1/2}\mathbf{1}$, it would suffice to use the power method with $\varepsilon = 1/2$ in order to approximate λ_2 up to a factor of 2. Hence, we would only require $k = \Theta(\log n)$. Unfortunately, computing $\mathcal{L}^+$ exactly is an expensive operation. Theorem 3.1 from Section 3 implies that there is an algorithm, LSolve, that given $\mathbf{v}$ can approximate $\mathcal{L}^+\mathbf{v}$ in time $O(m \log n/\sqrt{\lambda_2})$. Roughly, this implies that the total time required to estimate λ_2 via this method is $O(m(\log n)^2/\sqrt{\lambda_2})$. This is the limit of methods which do not use the structure of $\mathcal{L}$. Note that $\mathcal{L}^+ = D^{1/2}L^+D^{1/2}$ hence, it is sufficient to compute $L^+\mathbf{v}$ when $\langle \mathbf{v}, \mathbf{1} \rangle = 0$.

Recall the procedure LSolve in Theorem 3.1 and its linearity in Section 3.4. An immediate corollary of these results is that there is a randomized procedure which, given a positive integer k, a graph Laplacian L, a vector $\mathbf{v} \in \mathbb{R}^n$, and an $\varepsilon > 0$, returns the vector $Z^k\mathbf{v}$, where Z is the symmetric linear operator implicit in LSolve satisfying

$(1 - \varepsilon/4)Z^+ \preceq L \preceq (1 + \varepsilon/4)Z^+$ with the same image as L. Moreover, this algorithm can be implemented in time $O(mk \log n \log 1/\varepsilon)$.

Coming back to our application, suppose we are given $\varepsilon < 1/5$. We choose k so that the Rayleigh quotient of $Z^k\mathbf{v}$, which is supposed to be an approximation to $(L^+)^k\mathbf{v}$, w.r.t. L^+ becomes at most $(1 + \varepsilon)\lambda_2(L)$ for a random vector $\mathbf{v}$. Thus, k is $O(1/\varepsilon \log n/\varepsilon)$. The important point is that k *does not depend* on $\lambda_2(L)$ (or in the normalized case, $\lambda_2(\mathcal{L})$).

Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be the eigenvectors of Z^+ corresponding to the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n = 0$. (For this proof it turns out to be more convenient to label the eigenvalues in decreasing order.) We express $\mathbf{v}$ in the basis $\{\mathbf{u}_i\}_i$ as $\mathbf{v} = \sum_i \alpha_i \mathbf{u}_i$. We know that with probability at least $1/2$, we have, $|\alpha_1| \geq 2/3\sqrt{n}$. Let j be the smallest index such that $\lambda_j \leq (1 + \varepsilon/8)\lambda_{n-1}$. Then,

$$\begin{aligned}
\frac{(Z^k\mathbf{v})^\top Z^+ (Z^k\mathbf{v})}{(Z^k\mathbf{v})^\top (Z^k\mathbf{v})} &= \frac{\sum_{i=1}^{n-1} \alpha_i^2 \lambda_i^{-(2k-1)}}{\sum_{i=1}^{n-1} \alpha_i^2 \lambda_i^{-2k}} \\
&\leq \frac{\sum_{i=1}^{j} \alpha_i^2 \lambda_i^{-(2k-1)}}{\sum_{i=1}^{n-1} \alpha_i^2 \lambda_i^{-2k}} + \frac{\sum_{i>j}^{n-1} \alpha_i^2 \lambda_i^{-(2k-1)}}{\sum_{i>j}^{n-1} \alpha_i^2 \lambda_i^{-2k}} \\
&\leq \frac{\sum_{i=1}^{j} \alpha_i^2 \lambda_i^{-(2k-1)}}{\alpha_{n-1}^2 \lambda_{n-1}^{-2k}} + \lambda_j \\
&\leq \lambda_{n-1} \frac{\sum_{i=1}^{j} \alpha_i^2 (1 + \varepsilon/8)^{-(2k-1)}}{\alpha_{n-1}^2} + (1 + \varepsilon/8)\lambda_{n-1} \\
&\leq \lambda_{n-1} \cdot \frac{9n}{4} \sum_{i=1}^{j} \alpha_i^2 e^{-\frac{\varepsilon(2k-1)}{16}} + (1 + \varepsilon/8)\lambda_{n-1} \\
&\leq \lambda_{n-1} \frac{\varepsilon}{8} + \lambda_{n-1}(1 + \varepsilon/8) \\
&\leq \lambda_{n-1}(1 + \varepsilon/4),
\end{aligned}$$

where the third last line has used $(1 + x)^{-1} \leq e^{-x/2}$ for $x \leq 1$ and $k \geq 8/\varepsilon \log 18n/\varepsilon + 1$. Let $\hat{\mathbf{v}}$ be the unit vector $Z^k\mathbf{v}/\|Z^k\mathbf{v}\|$. Thus, $\hat{\mathbf{v}}L\hat{\mathbf{v}} \leq (1 + \varepsilon/4)\hat{\mathbf{v}}Z^+\hat{\mathbf{v}} \leq (1 + \varepsilon/4)^2\lambda_{n-1}(Z^+) \leq \frac{(1+\varepsilon/4)^2}{1-\varepsilon/4}\lambda_2(L) \leq (1 + \varepsilon)\lambda_2(L)$, where the last inequality uses $\varepsilon < 1/5$. This, when combined with Lemma 8.1, completes the proof of Theorem 8.2.

Notes

Lemma 8.1 is folklore. Theorem 8.2 is from [78]. It is important to note that there is *no* known proof of Theorem 8.2 without invoking a Laplacian solver; it seems important to understand why.

9

The Matrix Exponential and Random Walks

This section considers the problem of computing the matrix exponential, $\exp(-tL)\mathbf{v}$, for a graph Laplacian L and a vector $\mathbf{v}$. The matrix exponential is fundamental in several areas of mathematics and science, and of particular importance in random walks and optimization. Combining results from approximation theory that provide rational approximations to the exponential function with the Laplacian solver, an algorithm that approximates $\exp(-tL)\mathbf{v}$ up to ε-error is obtained. This algorithm runs in $\tilde{O}(m \log t \log 1/\varepsilon)$ time.

9.1 The Matrix Exponential

Suppose A is a symmetric $n \times n$ matrix. The matrix exponential of A is defined to be

$$\exp(A) \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \frac{A^i}{i!}.$$

Since A is symmetric, an equivalent way to define $\exp(A)$ is to first write the spectral decomposition of $A = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$, where λ_is are

the eigenvalues of A and $\mathbf{u}_i$s are its orthonormal eigenvectors. Then,

$$\exp(A) = \sum_{i=1}^{n} \exp(\lambda_i)\mathbf{u}_i\mathbf{u}_i^\top.$$

Thus, $\exp(A)$ is a symmetric PSD matrix. This matrix plays a fundamental role in mathematics and science and, recently, has been used to design fast algorithms to solve semidefinite programming formulations for several graph problems. In particular, it has been used in an intricate algorithm for Balanced Edge-Separator that runs in time $\tilde{O}(m)$ and achieves the spectral bound as in Theorem 7.1. In such settings, the primitive that is often required is $\exp(-L)\mathbf{v}$, where L is the graph Laplacian of a weighted graph. One way to compute an approximation to $\exp(-L)\mathbf{v}$ is to truncate the power series of $\exp(-L)$ after t terms and output

$$\mathbf{u} \stackrel{\text{def}}{=} \sum_{i=0}^{t} \frac{(-1)^i L^i \mathbf{v}}{i!}.$$

If we want $\|\mathbf{u} - \exp(-L)\mathbf{v}\| \leq \varepsilon\|\mathbf{v}\|$, then one may have to choose $t \sim \|L\| + \log 1/\varepsilon$. Thus, the time to compute $\mathbf{u}$ could be as large as $O(m(\|L\| + \log 1/\varepsilon))$. In matrix terms, this algorithm uses the fact that the matrix polynomial $\sum_{i=0}^{t} \frac{(-1)^i L^i}{i!}$ approximates $\exp(-L)$ up to an error of ε when $t \sim \|L\| + \log 1/\varepsilon$. This dependence on $\|L\|$ is prohibitive for many applications and in this section we present an algorithm where the dependence is brought down to $\log \|L\|$ as in the following theorem.

Theorem 9.1. There is an algorithm that, given the graph Laplacian L of a weighted graph with n vertices and m edges, a vector $\mathbf{v}$, and a parameter $0 < \delta \leq 1$, outputs a vector $\mathbf{u}$ such that $\|\exp(-L)\mathbf{v} - \mathbf{u}\| \leq \delta\|\mathbf{v}\|$ in time $\tilde{O}((m+n)\log(1+\|L\|)\text{polylog } 1/\delta)$.

The proof of this theorem essentially reduces computing $\exp(-L)\mathbf{v}$ to solving a small number of Laplacian systems by employing a powerful result from approximation theory. Before we give the details of this latter result, we mention a generalization of the Laplacian solver to

Symmetric, Diagonally Dominant (SDD) matrices. A symmetric matrix A is said to be SDD if for all i,

$$A_{ii} \geq \sum_j |A_{ij}|.$$

Notice that a graph Laplacian is SDD. Moreover, for any $\gamma > 0$, $I + \gamma L$ is also SDD. While it is not straightforward, it can be shown that if one has black-box access to a solver for Laplacian systems, such as Theorem 3.1, one can solve SDD systems. We omit the proof.

Theorem 9.2. Given an $n \times n$ SDD matrix A with m nonzero entries, a vector $\mathbf{b}$, and an error parameter $\varepsilon > 0$, it is possible to obtain a vector $\mathbf{u}$ such that

$$\|\mathbf{u} - A^+\mathbf{b}\|_A \leq \varepsilon \|A^+\mathbf{b}\|_A.$$

The time required for this computation is $\tilde{O}(m \log n \log(1/(\varepsilon \|A^+\|)))$. Moreover, $\mathbf{u} = Z\mathbf{b}$ where Z depends on A and ε, and is such that $\|Z - A^+\| \leq \varepsilon$.

9.2 Rational Approximations to the Exponential

The starting point for the proof of Theorem 9.2 is the following result which shows that simple rational functions provide uniform approximations to $\exp(-x)$ over $[0, \infty)$ where the error term decays exponentially with the degree. The proof of this theorem is beyond the scope of this monograph.

Theorem 9.3. There exists constants $c \geq 1$ and k_0 such that, for any integer $k \geq k_0$, there exists a polynomial $P_k(x)$ of degree k such that,

$$\sup_{x \in [0,\infty)} \left| \exp(-x) - P_k\left(\frac{1}{1 + x/k}\right) \right| \leq ck \cdot 2^{-k}.$$

A corollary of this theorem is that for any unit vector $\mathbf{v}$,

$$\|\exp(-L)\mathbf{v} - P_k((I + L/k)^+)\mathbf{v}\| \leq O(k2^{-k}).$$

To see this, first write the spectral decomposition $L = \sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$ where $\{\mathbf{u}\}_{i=1}^n$ form an orthonormal basis. Then, note that

$$(I + {}^{L}\!/_{k})^{+} = \sum_i (1 + {}^{\lambda_i}\!/_{k})^{-1} \mathbf{u}_i \mathbf{u}_i^\top$$

and, hence,

$$P_k((I + {}^{L}\!/_{k})^{+}) = \sum_i P_k((1 + {}^{\lambda_i}\!/_{k})^{-1}) \mathbf{u}_i \mathbf{u}_i^\top.$$

Here, we have used the spectral decomposition theorem from Section 1 on pseudo-inverses which can be readily justified. Thus, if $\mathbf{v} = \sum_i \beta_i \mathbf{u}_i$ with $\sum_i \beta_i^2 = 1$, then

$$P_k((I + {}^{L}\!/_{k})^{+})\mathbf{v} = \sum_i \beta_i P_k((1 + {}^{\lambda_i}\!/_{k})^{-1}) \mathbf{u}_i,$$

and

$$\exp(-L)\mathbf{v} = \sum_i \beta_i \exp(-\lambda_i) \mathbf{u}_i.$$

Thus, using orthonormality of $\{\mathbf{u}_i\}_{i=1}^n$ we obtain that

$$\begin{aligned} &\| \exp(-L)\mathbf{v} - P_k((I + {}^{L}\!/_{k})^{+})\mathbf{v} \|^2 \\ &\quad \leq \sum_i \beta_i^2 \cdot (\exp(-\lambda_i) - P_k((1 + {}^{\lambda_i}\!/_{k})^{-1}))^2. \end{aligned}$$

Hence, $\| \exp(-L)\mathbf{v} - P_k((I + {}^{L}\!/_{k})^{+})\mathbf{v} \|$ is at most

$$\sqrt{\sum_i \beta_i^2} \cdot \max_i |\exp(-\lambda_i) - P_k((1 + {}^{\lambda_i}\!/_{k})^{-1})|.$$

Since $\sum_i \beta_i^2 = 1$, this implies that

$$\begin{aligned} &\| \exp(-L)\mathbf{v} - P_k((I + {}^{L}\!/_{k})^{+})\mathbf{v} \| \\ &\quad \leq \max_{x \in \Lambda(L)} |\exp(-x) - P_k((1 + {}^{x}\!/_{k})^{-1})|, \end{aligned}$$

where $\Lambda(L) \stackrel{\text{def}}{=} [\lambda_1(L), \lambda_n(L)]$. Thus, by the choice of P_k as in Theorem 9.3, since all eigenvalues of the Laplacian are nonnegative, we have proved the following lemma.

Lemma 9.4. There exists constants $c \geq 1$ and k_0 such that, for any integer $k \geq k_0$, there exists a polynomial $P_k(x)$ of degree k such that, for any graph Laplacian L and a vector $\mathbf{v}$,

$$\|\exp(-L)\mathbf{v} - P_k((I + L/k)^+)\mathbf{v}\| \leq O(k2^{-k})\|\mathbf{v}\|.$$

Proof of Theorem 9.1

To use $P_k((I + L/k)^+)\mathbf{v}$ as an approximation to $\exp(-L)\mathbf{v}$, let us assume that P_k is given to us as $P_k(x) = \sum_{i=0}^{k} c_i x^i$ with $|c_i| = O(k^k)$. This is not obvious and a justification is needed to assume this. We omit the details. Thus, we need to compute

$$\sum_{i=0}^{k} c_i((I + L/k)^+)^i\mathbf{v}.$$

Here, we assume that on input $\mathbf{v}$ and $(I + L/k)$, the output of Theorem 9.2 is $Z\mathbf{v}$ where $\|Z - (I + L/k)^+\| \leq \varepsilon$. Since $\|(I + L/k)^+\| \leq 1$ and Z is ε-close to it, we obtain that $\|Z\| \leq (1 + \varepsilon)$. Thus, using the simple inequality that for $a, b \in \mathbb{R}$,

$$a^j - b^j = (a - b) \cdot \left(\sum_{i=0}^{j-1} (-1)^i a^{j-1-i} b^i\right),$$

and we obtain that

$$\|Z^j\mathbf{v} - ((I + L/k)^+)^j\mathbf{v}\| \leq 2\varepsilon j(1 + \varepsilon)^j\|\mathbf{v}\|.$$

Therefore, by the triangle inequality,

$$\left\|\sum_{i=0}^{k} c_i Z^i\mathbf{v} - \sum_{i=0}^{k} c_i((I + L/k)^+)^i\mathbf{v}\right\| \leq \varepsilon k^2(1 + \varepsilon)^k \max_i |c_i|.$$

Since $\mathbf{u} = \sum_{i=0}^{k} c_i Z^i\mathbf{v}$, combining the inequality above with Lemma 9.4, we get that

$$\|\exp(-L)\mathbf{v} - \mathbf{u}\| \leq \varepsilon k^2(1 + \varepsilon)^k k^{O(k)}\|\mathbf{v}\| + O(k2^{-k})\|\mathbf{v}\|$$

where we have used that $\max_i |c_i| = k^{O(k)}$. For a parameter δ, choose $k = \log 2/\delta$ and $\varepsilon = \frac{\delta(1+\|L\|)}{2k^{O(k)}}$ to obtain that

$$\|\exp(-L)\mathbf{v} - \mathbf{u}\| \leq \delta\|\mathbf{v}\|.$$

This completes the proof of Theorem 9.1.

9.3 Simulating Continuous-Time Random Walks

In this section we show how one can simulate continuous-time random walks on an undirected graph with m edges for time t in $\tilde{O}(m \log t)$ time. Not only are random walks of fundamental importance, the ability to simulate them in near-linear-time results in near-linear-time algorithms for the Balanced Edge-Separator problem. From an applications point of view, the bound on the running time is essentially independent of t. Now we phrase the problem and show how this simulation result is an immediate corollary of Theorem 9.1. For an undirected graph G, let D denote its degree matrix and A its adjacency matrix. Then, the transition matrix of a one-step random walk in G is $W \stackrel{\text{def}}{=} AD^{-1}$. Starting with a distribution $\mathbf{v}$ at time 0, the t-step transition probability of the discrete-time random walk on G is given by $W^t\mathbf{v}$. The continuous-time random walk on G for time t, also called the *heat-kernel* walk, is defined by the following probability matrix:

$$\widetilde{W}_t \stackrel{\text{def}}{=} \exp(-t(I - W)) = \exp(-t)\sum_{i=0}^{\infty} \frac{t^i}{i!} W^i.$$

Note that this can be interpreted as a the discrete-time random walk after a Poisson-distributed number of steps with rate t. Here, given a starting vector $\mathbf{v}$, the goal is to compute $\widetilde{W}_t\mathbf{v}$. Before we show how to do this, note that W is not symmetric. However, it can be symmetrized by considering the spectrally equivalent matrix $D^{-1/2}WD^{1/2}$ which is $I - \mathcal{L}$ where $\mathcal{L}$ is the normalized Laplacian of G. Thus, $W = D^{1/2}(I - \mathcal{L})D^{-1/2}$, and hence,

$$\widetilde{W}_t = D^{1/2}\exp(-t\mathcal{L})D^{-1/2}.$$

Now, it is easy to see that Theorem 9.1 applies since $t\mathcal{L} \succeq 0$ and $\|t\mathcal{L}\| \leq O(t)$. Thus, given $\delta > 0$, we can use the algorithm in the proof

of Theorem 9.1 to obtain a vector $\mathbf{u}$ such that

$$\|\exp(-t\mathcal{L})D^{-1/2}\mathbf{v} - \mathbf{u}\| \leq \delta\|D^{-1/2}\mathbf{v}\|.$$

The approximation to $\widetilde{W}_t\mathbf{v}$ is then $D^{1/2}\mathbf{u}$. It follows that

$$\|\widetilde{W}_t\mathbf{v} - D^{1/2}\mathbf{u}\| \leq \delta\|D^{1/2}\| \cdot \|D^{-1/2}\|\|\mathbf{v}\|.$$

This proves the following theorem.

Theorem 9.5. There is an algorithm that, given an undirected graph G with m edges, a vector $\mathbf{v}$, a time $t \geq 0$, and a $\delta > 0$, outputs a vector $\hat{\mathbf{u}}$ such that

$$\|\widetilde{W}_t\mathbf{v} - \hat{\mathbf{u}}\| \leq \delta\sqrt{\frac{d_{\max}}{d_{\min}}} \cdot \|\mathbf{v}\|.$$

The time taken by the algorithm is $\tilde{O}(m\log(1+t)\text{polylog}1/\delta)$. Here, $d_{\max}$ is the largest degree of G and $d_{\min}$ the smallest.

Notes

For more on the matrix exponential and the role it plays in the design of fast algorithms for semidefinite programs the reader is referred to sample applications as in [12, 40, 41, 60, 61, 62]. Two thesis on this topic are [44] and [59].

The reduction from SDD systems to Laplacian systems appears in [38]. Theorems 9.1 and 9.5 are implicit in the work of Orecchia et al. [60]. Theorem 9.3 on rational approximations was proved in [70]. The fact that the coefficients can be computed and are bounded is not present in this paper, but can be proved. While this section shows how to reduce the computation of $\exp(-L\mathbf{v})$ to polylog n computations of the form $L^+\mathbf{v}$, one may ask if the converse is true. Recently, Sachdeva and Vishnoi [69] answered this question by presenting a simple reduction in the other direction: The inverse of a positive-definite matrix can be approximated by a weighted-sum of a small number of matrix exponentials. This proves that the problems of exponentiating and inverting are essentially equivalent up to polylog factors. This reduction makes

no use of the fact that L is a Laplacian, but only that it is a symmetric positive-semidefinite matrix.

For more on continuous-time random walks, the reader is referred to the book [51]. It is a challenging open problem to prove an analog of Theorem 9.5 for discrete-time (lazy) random walks: Find a vector $\mathbf{u}$ s.t. $\|W^t\mathbf{v} - \mathbf{u}\| \leq \varepsilon\|\mathbf{v}\|$ in $\tilde{O}(m \log(1+t) \log 1/\varepsilon)$ time.

10

Graph Sparsification I
Sparsification via Effective Resistances

In this section a way to spectrally sparsify graphs is introduced that uses the connection to electrical networks presented in Section 4. In the next section we show how sparse spectral sparsifiers can be computed in $\tilde{O}(m)$ time using Laplacians. Spectral sparsifiers introduced in this section play a crucial role in the proof of Theorem 3.1 presented in Section 18.

10.1 Graph Sparsification

In the cut sparsification problem, one is given an undirected unweighted graph $G = (V, E)$ and a parameter $\varepsilon > 0$, and the goal is to find a weighted graph $H = (V, E')$ such that for every cut $(S, \bar{S})$ of V, the weight of the edges that cross this cut in H is within a multiplicative $1 \pm \varepsilon$ factor of the number of edges in G that cross this cut. The goal is to keep the number of edges (not counting weights) of H small. Moreover, finding H quickly is also important for applications. Benczur and Karger gave a remarkable algorithm which produces cut sparsifiers of size (i.e., number of edges) $\tilde{O}(n/\varepsilon^2)$ in time $\tilde{O}(m)$. Such a procedure has found use in many fundamental graph algorithms where it allows the running time to be brought down from its dependency on m to n

(note that m can be as big as $\Omega(n^2)$). In particular, cut sparsifiers can be used in the algorithms presented in Sections 6, 7, and 9.

A stronger notion than cut sparsification is that of spectral sparsification. This will play an important role in constructing Laplacian solvers in Section 18. However, in this section and the next, we show how to construct spectral sparsifiers in $\tilde{O}(m)$ time *using* Laplacian solvers.

Definition 10.1. Given an undirected graph $G = (V, E)$ with a parameter $\varepsilon > 0$, a weighted graph $H = (V, E')$ is said to be an ε-spectral sparsifier of G if

$$\frac{1}{(1+\varepsilon)} \leq \frac{\mathbf{x}^\top L_H \mathbf{x}}{\mathbf{x}^\top L_G \mathbf{x}} \leq (1+\varepsilon), \quad \forall \mathbf{x} \in \mathbb{R}^n.$$

Where L_G and L_H denote the graph Laplacians for G and H, respectively.

The goal is to minimize the number of edges of H and construct it as quickly as possible. In particular, we consider whether spectral sparsifiers with $\tilde{O}(n/\text{poly}(\varepsilon))$ edges exist and can be constructed in $\tilde{O}(m)$ time. Before we go on, notice that if H is an ε-spectral sparsifier for G then it is an ε-cut sparsifier for G as well. To see this, plug the vectors $\mathbf{x} = \mathbf{1}_S$, (the indicator vector for a cut $(S, \bar{S})$), into the definition above. There are also easy-to-construct examples that show these notions are *not* the same. An ε-cut sparsifier of G can be an arbitrarily bad spectral sparsifier for G.

We show the existence of ε-spectral sparsifiers of G of size $O(n \log n/\varepsilon^2)$. In the next section, we show how to construct such a sparsifier in time $\tilde{O}(m \log 1/\varepsilon)$.[1] Formally, we prove the following theorem.

Theorem 10.1. There exists a randomized algorithm that, given a graph G and an approximation parameter $\varepsilon > 0$, constructs a spectral sparisifer H of size $O(n \log n/\varepsilon^2)$ with probability $1 - 1/n$.

---

[1] If G is weighted, then there is also a dependency on the log of the ratio of the largest to the smallest weight in G in the running time. The running time crucially relies on the Spielman–Teng Laplacian solver.

In the next section we refine this theorem to prove the following, which also includes a bound on the running time.

Theorem 10.2. There exists a randomized algorithm that, given a graph G and an approximation parameter $\varepsilon > 0$, constructs a spectral sparisifer H of size $O(n \log n/\varepsilon^2)$ in time $\tilde{O}(m \log 1/\varepsilon)$ with probability $1 - 1/n$.

10.2 Spectral Sparsification Using Effective Resistances

The proof of Theorem 10.1 relies on an edge sampling algorithm: Repeatedly sample (with replacement) edges from G according to a carefully chosen probability function over the edges and weight each sampled edge inversely to this probability. The resulting multiset of weighted edges, normalized by the number of samples, is the output of the algorithm.[2] Formally, let p_e be the probability that edge e is chosen by the sampling algorithm. Let Y be the random variable such that $\mathbb{P}[Y = e] = p_e$. Let T be a parameter denoting the number of samples that are taken by the algorithm. Let $Y_1, Y_2, \ldots, Y_T$ be i.i.d. copies of Y. Then the weighted multi-set of edges equals

$$\left\{ \left(Y_1, \frac{1}{T \cdot p_{Y_1}} \right), \left(Y_2, \frac{1}{T \cdot p_{Y_2}} \right), \ldots, \left(Y_T, \frac{1}{T \cdot p_{Y_T}} \right) \right\}.$$

What is the Laplacian of the associated graph H? Let B be an incidence matrix for G and let $\mathbf{b}_e$ be the column vector corresponding to edge e in $B^\top$. Then, $L_G = \sum_e \mathbf{b}_e \mathbf{b}_e^\top = B^\top B$. For notational convenience, define $\mathbf{u}_e \stackrel{\text{def}}{=} \mathbf{b}_e/\sqrt{p_e}$. Then, a simple calculation shows that

$$L_H \stackrel{\text{def}}{=} \frac{1}{T} \sum_{i=1}^{\top} \frac{\mathbf{b}_{Y_i} \mathbf{b}_{Y_i}^\top}{p_{Y_i}} = \frac{1}{T} \sum_{i=1}^{\top} \mathbf{u}_{Y_i} \mathbf{u}_{Y_i}^\top.$$

To analyze the effectiveness of this scheme, first note that

$$\mathbb{E}[\mathbf{u}_Y \mathbf{u}_Y^\top] = \sum_e \mathbb{P}[Y = e] \cdot \mathbf{u}_e \mathbf{u}_e^\top = \sum_e \mathbf{b}_e \mathbf{b}_e^\top = L_G.$$

[2] This multi-set can be converted to a weighted graph by additively combining weights of repeated edges.

Therefore,

$$\mathbb{E}[L_H] = \frac{1}{T}\sum_{i=1}^{\top}\mathbb{E}[\mathbf{u}_{Y_i}\mathbf{u}_{Y_i}] = L_G.$$

So far this is abstract because we have not specified the p_es. We want to choose p_e so that L_H behaves similar to L_G w.h.p. This is where effective resistances and the intuition from Theorem 4.5 come into play. We let

$$p_e \stackrel{\text{def}}{=} \frac{R_e}{n-1},$$

where $R_e \stackrel{\text{def}}{=} R_{\text{eff}}(e)$ is the effective resistance of the edge e defined to be $\mathbf{b}_e^\top L_G^+ \mathbf{b}_e$.[3] The normalization factor of $n-1$ is due to the fact $\sum_e R_e = n-1$. What is the intuition behind choosing an edge with probability proportional to its effective resistance? Unfortunately, there does not exist a very satisfactory answer to this question. We give a rough idea here. By Theorem 4.5, R_e is proportional to the probability of an edge being in a random spanning tree and choosing a small number of random spanning trees from G independently seems like a good strategy to spectrally sparsify G. We move on to analyze this strategy.

Proof of Theorem 10.1

For a fixed orientation of the edges of G, recall that the matrix $\Pi \stackrel{\text{def}}{=} BL_G^+B^\top$ introduced in Section 4 satisfies the following properties:

(1) $\Pi^2 = \Pi$,
(2) Let Π_e denote the column of Π corresponding to edge e. Then $R_e = \|\Pi_e\|^2$,
(3) $\sum_e R_e = n-1$, and
(4) Π is unitarily equivalent to $\sum_{j=1}^{n-1}\mathbf{e}_j\mathbf{e}_j^\top$, where $\mathbf{e}_j$ is the j-th standard basis vector in $\mathbb{R}^m$. This is because Π is symmetric and PSD, and all its eigenvalues are 0 or 1. Thus, if G is connected, the multiplicity of eigenvalue 1 is $n-1$ and that of 0 is $m-n+1$.

[3] If the graph G is weighted, then we choose p_e proportional to $w_G(e)R_e$.

Define $\mathbf{v}_e \stackrel{\text{def}}{=} \Pi_e/\sqrt{p_e}$. Let $M \stackrel{\text{def}}{=} \mathbf{v}_Y \mathbf{v}_Y^\top$ and $M_i \stackrel{\text{def}}{=} \mathbf{v}_{Y_i}\mathbf{v}_{Y_i}^\top$, $i = 1, 2, \ldots, T$ be i.i.d. copies of M. The following theorem is required. It is an analog of Chernoff bound for sums of matrix-valued random variables.

Theorem 10.3. Let $\varepsilon > 0$ be a small enough constant. Let $M \in \mathbb{R}^{d\times d}$ be a random, symmetric PSD matrix such that $\mathbb{E}[M] = I_d$, where I_d is the d-dimensional identity matrix. Let $\rho \stackrel{\text{def}}{=} \sup_M \|M\|$. Let T be a non-negative integer and let $M_1, \ldots, M_T$ be independent copies of M. Then,

$$\mathbb{P}\left[\left\|\frac{1}{T}\sum_{i=1}^{T} M_i - I_d\right\| > \varepsilon\right] \leq 2d \cdot \exp\left(-\frac{T\varepsilon^2}{2\rho}\right).$$

Theorem 10.3 also holds when $\mathbb{E}[M] = \sum_{j=1}^{d'} \mathbf{e}_j\mathbf{e}_j^\top$ for some $d' \leq d$, in which case the above bound holds with d' in place of d. A slightly more general result, which is needed for our application, is that the bound also holds when $\mathbb{E}[M]$ is unitarily equivalent to the above matrix. This follows because the operator norm is unitarily invariant. As an application of Theorem 10.3, we first calculate

$$\mathbb{E}[M] = \mathbb{E}[\mathbf{v}_e\mathbf{v}_e^\top] = \sum_e \Pi_e\Pi_e^\top = \Pi,$$

which by Property (4) above is unitarily equivalent to $\sum_{j=1}^{n-1}\mathbf{e}_j\mathbf{e}_j^\top$. Note that Property (2) implies that

$$\|\mathbf{v}_e\|^2 = \frac{\|\Pi_e\|^2}{p_e} = \frac{R_e}{p_e} = n - 1.$$

Therefore $\|M_i\| \leq n - 1$ for all i. Applying Theorem 10.3, we obtain

$$\mathbb{P}[\|\widetilde{\Pi} - \Pi\| > \varepsilon] \leq 2(n-1) \cdot \exp\left(-\frac{T\varepsilon^2}{2(n-1)}\right), \tag{10.1}$$

where $\widetilde{\Pi} \stackrel{\text{def}}{=} \frac{1}{T}\sum \mathbf{v}_{Y_i}\mathbf{v}_{Y_i}^\top$. Setting $T = O(n\log n/\varepsilon^2)$ ensures that this failure probability is $n^{-\Omega(1)}$. How do we relate this to L_H? Observe that since $\Pi = BL_G^+B^\top$, it follows that for any e:

$$\mathbf{v}_e = \frac{\Pi_e}{\sqrt{p_e}} = \frac{BL_G^+\mathbf{b}_e}{\sqrt{p_e}} = BL_G^+\mathbf{u}_e.$$

Therefore,

$$\widetilde{\Pi} = \frac{1}{T}\sum \mathbf{v}_{Y_i}\mathbf{v}_{Y_i}^\top = \frac{1}{T}\sum BL_G^+\mathbf{u}_{Y_i}\mathbf{u}_{Y_i}^\top L_G^+B^\top = BL_G^+L_HL_G^+B^\top,$$

and

$$\Pi = BL_G^+B^\top = BL_G^+L_GL_G^+B^\top.$$

Now,

$$\left\|\widetilde{\Pi} - \Pi\right\| = \sup_{\mathbf{x}\neq\mathbf{0}}\left|\frac{\mathbf{x}^\top(\widetilde{\Pi}-\Pi)\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}\right| = \sup_{\mathbf{x}\neq\mathbf{0}}\left|\frac{\mathbf{x}^\top BL_G^+(L_H-L_G)L_G^+B^\top\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}\right|.$$

Because G is connected, if $\mathbf{z}$ is a vector such that $B\mathbf{z} = 0$, then $\mathbf{z}$ must be parallel to the all 1s vector. Hence, for any $\mathbf{z} \neq \mathbf{0}$ such that $\langle \mathbf{z}, \mathbf{1}\rangle = 0$, $B\mathbf{z} \neq \mathbf{0}$. Therefore, we can substitute $\mathbf{x} = B\mathbf{z}$ in the above equation. Using the property that $L_GL_G^+\mathbf{z} = \mathbf{z}$ for any such $\mathbf{z}$, we obtain

$$\begin{aligned}
\left\|\widetilde{\Pi} - \Pi\right\| &\geq \sup_{\substack{\mathbf{z}\neq\mathbf{0}\\ \langle\mathbf{z},\mathbf{1}\rangle=0}}\left|\frac{\mathbf{z}^\top B^\top BL_G^+(L_H-L_G)L_G^+B^\top B\mathbf{z}}{\mathbf{z}^\top B^\top B\mathbf{z}}\right| \\
&= \sup_{\substack{\mathbf{z}\neq\mathbf{0}\\ \langle\mathbf{z},\mathbf{1}\rangle=0}}\left|\frac{\mathbf{z}^\top L_GL_G^+(L_H-L_G)L_G^+L_G\mathbf{z}}{\mathbf{z}^\top L_G\mathbf{z}}\right| \\
&= \sup_{\substack{\mathbf{z}\neq\mathbf{0}\\ \langle\mathbf{z},\mathbf{1}\rangle=0}}\left|\frac{\mathbf{z}^\top(L_H-L_G)\mathbf{z}}{\mathbf{z}^\top L_G\mathbf{z}}\right|.
\end{aligned}$$

Combining with Equation (10.1) yields

$$\mathbb{P}\left[\sup_{\substack{\mathbf{z}\neq\mathbf{0}\\ \langle\mathbf{z},\mathbf{1}\rangle=0}}\left|\frac{\mathbf{z}^\top(L_H-L_G)\mathbf{z}}{\mathbf{z}^\top L_G\mathbf{z}}\right| > \varepsilon\right] \leq \mathbb{P}[\|\widetilde{\Pi}-\Pi\| > \varepsilon] = n^{-\Omega(1)}$$

as desired. This completes the proof of Theorem 10.1.

10.3 Crude Spectral Sparsfication

In this section we note that the algorithm and the proof underlying Theorem 10.1 can be modified to obtain a *crude* spectral sparsifier.

The exact knowledge of R_e is no longer necessary. Instead, the proof above can be modified to work when we have $q_e \geq R_e$ for all e. The number of samples required to ensure that the resulting graph is a spectral sparsifier can be easily shown to be about $W \stackrel{\text{def}}{=} \sum_e q_e$, which replaces $n-1$. We record this theorem here (for weighted graphs) and show how to use it in Section 18.

Theorem 10.4. Consider graph $G = (V, E)$ with edge weights w_G, $\gamma > 0$, and numbers q_e such that $q_e \geq w_G(e)R_e$ for all e. If $W \stackrel{\text{def}}{=} \sum_e q_e$, then the spectral sparsifier in Theorem 10.1 that takes $O(W \log W \log 1/\gamma)$ samples from the probability distribution induced by the q_es produces a sampled graph H that satisfies

$$G \preceq 2H \preceq 3G$$

with probability $1 - \gamma$.

Notes

The notion of cut sparsification was introduced in [16]. The state of the art on cut sparsification can be found in [31].

Spectral sparsification was introduced in [80] and plays a crucial role in the construction of Laplacian solvers, and the first algorithm for a spectral sparsifier was given in this paper. Their sparsifier had size $O(n\text{polylog}n)$ and could be constructed in $\tilde{O}(m)$ time. Theorem 10.1 was proved in [76] where it is showed how to construct ε-spectral sparsifiers of size $O(n \log n/\varepsilon^2)$. This has been improved to $O(n/\varepsilon^2)$ in [14], coming close to the so-called Ramanujan bound.

Theorem 10.4 was observed in [49] and will be used in Section 18. Theorem 10.3, which is used in the proof of Theorem 10.1 was proved in [66] using a non-commutative Khinchine inequality. Subsequently, it has been proved in an elementary manner using the matrix exponential (introduced in Section 9) independently by several researchers, see [4].

11

Graph Sparsification II
Computing Electrical Quantities

In this section the near-linear-time Laplacian solver is used to approximately compute effective resistances for all edges in $\tilde{O}(m)$ time. This was needed in Section 10. This section starts by presenting the simpler task of approximately computing currents and voltages when a unit current is input at a vertex and taken out at another, again using Laplacian solvers. These primitives are also used in Section 12.

11.1 Computing Voltages and Currents

Given $s,t \in V$ with one unit of current flowing into s and one unit flowing out of t, the goal is to compute the vector $L^+(\mathbf{e}_s - \mathbf{e}_t)$; the vector of voltages at the vertices of the graph. We employ Theorem 3.1 to compute this vector in an approximate sense in $\tilde{O}(m)$ time. Recall that this theorem says there is an algorithm LSolve which takes a graph Laplacian L, a vector $\mathbf{y}$, and an error parameter $\delta > 0$, and returns an $\mathbf{x}$ satisfying

$$\|\mathbf{x} - L^+\mathbf{y}\|_L \leq \delta \|L^+\mathbf{y}\|_L.$$

Note that for any two vectors $\|\mathbf{v} - \mathbf{w}\|_\infty \leq \|\mathbf{v} - \mathbf{w}\|$. Hence, by a choice of $\delta \stackrel{\text{def}}{=} \varepsilon/\text{poly}(n)$ in Theorem 3.1, we can ensure the approximate vector of

voltages we produce is ε-close in every coordinate to the actual voltage vector. Since the dependence of the running time in Theorem 3.1 on the error tolerance is logarithmic, the running time remains $\tilde{O}(m \log 1/\varepsilon)$.

Thus, for any small enough constant $\varepsilon > 0$, we obtain a vector $\mathbf{u}$ of voltages such that $\|\mathbf{u} - L^+(\mathbf{e}_s - \mathbf{e}_t)\|_\infty \leq \varepsilon$. Further, if $\mathbf{u}$ is the vector of voltages, then the vector of electrical currents, assuming all resistances are 1, is $B\mathbf{u}$, and can be computed in additional $O(m+n)$ time. Note that since $\|\mathbf{u} - L^+(\mathbf{e}_s - \mathbf{e}_t)\|_\infty \leq \varepsilon$,

$$\|B\mathbf{u} - BL^+(\mathbf{e}_s - \mathbf{e}_t)\|_\infty \leq 2\varepsilon.$$

Hence,

$$\|B^\top B\mathbf{u} - B^\top BL^+(\mathbf{e}_s - \mathbf{e}_t)\|_\infty = \|L\mathbf{u} - (\mathbf{e}_s - \mathbf{e}_t)\|_\infty \leq 2\varepsilon n.$$

We may assume that our current vector satisfies

$$\|L\mathbf{u} - (\mathbf{e}_s - \mathbf{e}_t)\|_\infty \leq \varepsilon.$$

Thus, $L\mathbf{u}$ may not be a unit s,t-flow, but very close to it. If one desires, one can compute in time $O(m+n)$ a vector $\tilde{\mathbf{v}}$ from $\mathbf{u}$ such that $L\tilde{\mathbf{v}} = \mathbf{e}_s - \mathbf{e}_t$ which is indeed a unit s,t-flow, and $\|\mathbf{u} - \tilde{\mathbf{v}}\|_\infty \leq \varepsilon$. We leave it as an exercise for the reader. We summarize the results of this section in the following theorem which states the result for a weighted graph; the extension is straightforward.

Theorem 11.1. There is an $\widetilde{O}(m(\log r)\log 1/\varepsilon)$ time algorithm which on input $\varepsilon > 0$, $G = (V, E, w)$ with $r \stackrel{\text{def}}{=} w_{\max}/w_{\min}$, and $s, t \in V$, finds vectors $\tilde{\mathbf{v}} \in \mathbb{R}^V$ and $\tilde{\mathbf{f}} \in \mathbb{R}^E$ such that if $\mathbf{v} \stackrel{\text{def}}{=} L^+(\mathbf{e}_s - \mathbf{e}_t)$ and $\mathbf{f} \stackrel{\text{def}}{=} WBL^+(\mathbf{e}_s - \mathbf{e}_t)$,

(1) $L\tilde{\mathbf{v}} = \mathbf{e}_s - \mathbf{e}_t$ and $\tilde{\mathbf{f}} = WB\tilde{\mathbf{v}}$,
(2) $\|\mathbf{v} - \tilde{\mathbf{v}}\|_\infty \leq \varepsilon$,
(3) $\|\mathbf{f} - \tilde{\mathbf{f}}\|_\infty \leq \varepsilon$, and
(4) $|\sum_e f_e^2/w_e - \sum_e \tilde{f}_e^2/w_e| \leq \varepsilon$.

Here W is an $m \times m$ diagonal matrix with $W(e,e) = w_e$.

11.2 Computing Effective Resistances

For an edge $e = ij$, let R_e denote the effective resistance of the edge e which, recall, is defined to be $(\mathbf{e}_i - \mathbf{e}_j)^\top L^+(\mathbf{e}_i - \mathbf{e}_j)$. For our spectral sparsification application we required a way to compute R_e for *all* edges. As noted in the previous section, if we are interested in near-linear time algorithms, we have to use an approximation $\tilde{R}_e$ of R_e. It can be shown that any $O(1)$ approximation suffices for our spectral sparsification application; it just increases the sample size by a constant factor. We saw in the last section that for a given $\varepsilon > 0$ and edge e, we can compute $\tilde{R}_e$ such that

$$(1 - \varepsilon)R_e \leq \tilde{R}_e \leq (1 + \varepsilon)R_e$$

in time $\tilde{O}(m \log 1/\varepsilon)$ using the LSolve. A naive extension that computes $\tilde{R}_e$ for all edges takes $\tilde{O}(m^2 \log 1/\varepsilon)$ time. Can we reduce this to $\tilde{O}(m \log 1/\varepsilon)$? The answer, as we will see shortly, is yes. The key observation is that for an edge $e = ij$,

$$R_e = \|BL^+(\mathbf{e}_i - \mathbf{e}_j)\|^2.$$

Hence, if we let $\mathbf{w}_i \stackrel{\text{def}}{=} BL^+\mathbf{e}_i$, then

$$R_e = \|\mathbf{w}_i - \mathbf{w}_j\|^2$$

for all e. The $\mathbf{w}_i$s live in $\mathbb{R}^m$. Is it necessary to compute $\mathbf{w}_i$s if all we are interested in the ℓ_2 distance of m pairs among them? The answer is no. The Johnson–Lindenstrauss Lemma postulates that computing the projections of these vectors on a random $\log m/\varepsilon^2$ dimensional space preserves distances up to a multiplicative factor of $1 \pm \varepsilon$. Hence, our approach is to choose a *random* matrix A which is roughly $\log m \times m$ and compute $A\mathbf{w}_i = ABL^+\mathbf{e}_i$ for all i. Several ways to choose A work and, in particular, we appeal to the following theorem where entries of A are ± 1 up to normalization. This random matrix A corresponds to the random subspace.

Theorem 11.2. Given fixed vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$ and $\varepsilon > 0$, let $A \in \{-1/\sqrt{k}, 1/\sqrt{k}\}^{k \times m}$ be a random matrix with $k \geq c \log n/\varepsilon^2$ for some constant $c > 0$. Then, with probability at least $1 - 1/n$, for all $1 \leq i, j \leq n$

$$(1 - \varepsilon)\|\mathbf{w}_i - \mathbf{w}_j\|^2 \leq \|A\mathbf{w}_i - A\mathbf{w}_j\|^2 \leq (1 + \varepsilon)\|\mathbf{w}_i - \mathbf{w}_j\|^2.$$

Getting back to our problem of computing R_e for all e, let $Z \stackrel{\text{def}}{=} ABL^+$ where A is the random matrix promised in the theorem above. We compute an approximation $\widetilde{Z}$ by using LSolve to approximately compute the rows of Z. Let the vectors $\mathbf{z}_i$ and $\tilde{\mathbf{z}}_i$ denote the ith rows of Z and $\tilde{Z}$, respectively (so that $\mathbf{z}_i$ is the i-th column of $Z^\top$). Now we can construct the matrix $\widetilde{Z}$ in the following three steps:

(1) Let A be a random $\pm 1/\sqrt{k}$ matrix of dimension $k \times m$ where $k = O(\log n/\varepsilon^2)$.
(2) Compute $Y = AB$. Note that this takes $2m \times O(\log n/\varepsilon^2) + m = \widetilde{O}(m/\varepsilon^2)$ time since B has $2m$ entries.
(3) Let $\mathbf{y}_i^\top$, for $1 \le i \le k$, denote the rows of Y, and compute $\tilde{\mathbf{z}}_i \stackrel{\text{def}}{=} \text{LSolve}(L, \mathbf{y}_i, \delta)$ for each i.

We now prove that, for our purposes, it suffices to call LSolve with $\delta \stackrel{\text{def}}{=} \frac{\varepsilon}{\text{poly}(n)}$. We do not explicitly specify the $\text{poly}(n)$, but it can be recovered from the proof below. First, we claim that if $\|\mathbf{z}_i - \tilde{\mathbf{z}}_i\| \le \varepsilon$ for all i, then for any vector $\mathbf{u}$ with $\|\mathbf{u}\| = O(1)$,

$$\frac{|\|Z\mathbf{u}\|^2 - \|\tilde{Z}\mathbf{u}\|^2|}{\|Z\mathbf{u}\|^2} \le \varepsilon \cdot \text{poly}(n).$$

In our application $u = \mathbf{e}_i - \mathbf{e}_j$ for some i, j. Hence, $\|\mathbf{u}\| = O(1)$. To see why our claim is true, let $a_i \stackrel{\text{def}}{=} \langle \mathbf{z}_i, \mathbf{u} \rangle$ and $b_i \stackrel{\text{def}}{=} \langle \tilde{\mathbf{z}}_i, \mathbf{u} \rangle$. Then, $\|Z\mathbf{u}\|^2 = \sum_i a_i^2$ and $\|\tilde{Z}\mathbf{u}\|^2 = \sum_i b_i^2$. Thus,

$$|\|Z\mathbf{u}\|^2 - \|\tilde{Z}\mathbf{u}\|^2| = \left|\sum_i (a_i^2 - b_i^2)\right| \le \sqrt{\sum_i (a_i - b_i)^2 \sum_i (a_i + b_i)^2}.$$

Note that

$$\sum_i (a_i + b_i)^2 \le 2 \sum_i (a_i^2 + b_i^2) = 2\|Z\mathbf{u}\|^2 + 2\|\tilde{Z}\mathbf{u}\|^2 = \text{poly}(n).$$

This is because $\|Z\mathbf{u}\|^2 = \|ABL^+(\mathbf{e}_i - \mathbf{e}_j)\|^2$ for some i, j. From Johnson–Lindenstrauss, we know that w.h.p. this is within a constant factor of $\|BL^+(\mathbf{e}_i - \mathbf{e}_j)\|^2 = (\mathbf{e}_i - \mathbf{e}_j)^\top L^+(\mathbf{e}_i - \mathbf{e}_j) = \text{poly}(n)$ as noted before. We need to estimate $\|\tilde{Z}\mathbf{u}\|^2$ and show that it is $\text{poly}(n)$ as well. Note that

$$|a_i - b_i| = |\langle \mathbf{z}_i - \tilde{\mathbf{z}}_i, \mathbf{u} \rangle| \le \|\mathbf{z}_i - \tilde{\mathbf{z}}_i\| \|\mathbf{u}\| = \varepsilon \|\mathbf{u}\| = O(\varepsilon).$$

Hence,

$$\sum_i (a_i - b_i)^2 \leq O(\varepsilon^2 \cdot k).$$

Thus, $b_i \leq a_i + O(\varepsilon)$ and, hence,

$$\begin{aligned}\sum_i b_i^2 &\leq \sum_i a_i^2 + O(\varepsilon \sum_i a_i) + O(\varepsilon^2 \cdot k) \\ &\leq \sum_i a_i^2 + O(\varepsilon \cdot \sqrt{k} \sum_i a_i^2) + O(\varepsilon^2 \cdot k).\end{aligned}$$

Therefore,

$$\|\tilde{Z}\mathbf{u}\|^2 = \sum_i b_i^2 \leq a_i^2 + \varepsilon^2 \cdot \text{poly}(k) = \|Z\mathbf{u}\|^2 + \varepsilon^2 \cdot \text{poly}(k) = \text{poly}(n),$$

and thus

$$|\|Z\mathbf{u}\|^2 - \|\tilde{Z}\mathbf{u}\|^2| \leq \sqrt{\sum_i (a_i - b_i)^2 \sum_i (a_i + b_i)^2} \leq \varepsilon \cdot \text{poly}(n).$$

Since $\|Z\mathbf{u}\|^2$ is the effective resistance of some edge, it is at least $1/\text{poly}(n)$ when the graph is unweighted. Hence,

$$\frac{|\|Z\mathbf{u}\|^2 - \|\tilde{Z}\mathbf{u}\|^2|}{\|Z\mathbf{u}\|^2} \leq \varepsilon \cdot \text{poly}(n).$$

To summarize, we have proved the following lemma.

Lemma 11.3. Suppose

$$(1 - \varepsilon)R_{ij} \leq \|Z(\mathbf{e}_i - \mathbf{e}_j)\|^2 \leq (1 + \varepsilon)R_{ij}$$

for every pair $i, j \in V$. If for all i, $\|\mathbf{z}_i - \tilde{\mathbf{z}}_i\|_L \leq \varepsilon \|\mathbf{z}_i\|_L$ where $\varepsilon \leq \frac{\delta}{\text{poly}(n)}$, then

$$(1 - \delta)R_{ij} \leq \|\widetilde{Z}(\mathbf{e}_i - \mathbf{e}_j)\|^2 \leq (1 + \delta)R_{ij}$$

for all $i, j \in V$.

Thus, the construction of $\widetilde{Z}$ takes $\widetilde{O}(m \log(1/\delta)/\varepsilon^2) = \widetilde{O}(m/\varepsilon^2)$ time. We can then find the approximate resistance $\|\widetilde{Z}(\mathbf{e}_i - \mathbf{e}_j)\|^2 \approx R_{ij}$ for any $i, j \in V$ in $O(\log n/\varepsilon^2)$ time. When the edges are weighted, one can extend this analysis to prove the following theorem.

Theorem 11.4. There is an $\widetilde{O}(m(\log r)/\varepsilon^2)$ time algorithm which, on input $\varepsilon > 0$ and $G = (V, E, w)$ with $r \stackrel{\text{def}}{=} w_{\max}/w_{\min}$, computes an $O(\log n/\varepsilon^2) \times n$ matrix $\widetilde{Z}$ such that with probability at least $1 - 1/n$

$$(1 - \varepsilon)R_{ij} \leq \|\widetilde{Z}(\mathbf{e}_i - \mathbf{e}_j)\|^2 \leq (1 + \varepsilon)R_{ij}$$

for every pair of vertices $i, j \in V$.

Notes

More details concerning Theorem 11.1 can be found in [22]. Theorem 11.2 was proved in [3]. Theorem 11.4 is from [76].

12

Cuts and Flows

This section presents an algorithm which reduces the s,t-max-flow/min-cut problems in undirected graphs to their corresponding electrical analogs. Since electrical flows/voltages can be computed in $\tilde{O}(m)$ time using Laplacian solvers as shown in Section 11, this section is dedicated to showing how, using the multiplicative weight update method, $\tilde{O}(\sqrt{m})$-electrical flow computations suffice to approximately compute the max-flow and min-cut between s and t.

12.1 Maximum Flows, Minimum Cuts

Given an undirected graph $G = (V, E)$ with edge capacities $c : E \mapsto \mathbb{R}_{\geq 0}$, the s,t-MaxFlow problem is to compute a flow of maximum value from a source $s \in V$ to a sink $t \in V$. To talk about a flow on G, it is convenient to direct its edges arbitrarily, captured by an incidence matrix B, and allow the flow value on an edge to be positive or negative. A flow $\mathbf{f} = (f_e)_{e \in E}$ is then a (combinatorial) s,t-flow if

(1) $\mathbf{f}^\top B\mathbf{e}_v = 0$ for all $v \in V \backslash \{s,t\}$,
(2) $\mathbf{f}^\top B\mathbf{e}_s + \mathbf{f}^\top B\mathbf{e}_t = 0$, and
(3) $|f_e| \leq c_e$ for all e.

The goal of the s,t-MaxFlow problem is to compute a flow which maximizes the flow out of s, i.e., $|\mathbf{f}^\top B\mathbf{e}_s|$. It is well known that the maximum s,t-flow is the same as the minimum capacity cut that separates s and t. The capacity of a cut $(S,\bar{S})$ is the sum of the capacities of all the edges with one endpoint in S and another in $\bar{S}$. This is the Max-Flow–Min-Cut theorem. The corresponding problem of finding the minimum cut separating s and t is called the s,t-MinCut problem.

In this section we give approximation algorithms for both the s,t-MaxFlow and the s,t-MinCut problem. Departing from traditional approaches, the algorithms presented in this section compute combinatorial flows and cuts by combining the information obtained from electrical flows and potentials which can be computed quickly to a good accuracy, see Section 11. We prove the following theorems.

Theorem 12.1. There is an algorithm such that, given an undirected, capacitated graph $G = (V,E,c)$, two vertices s,t, and an $\varepsilon > 0$, such that the maximum value from s to t is $F^\star$, the algorithm outputs a (combinatorial) flow of value at least $(1-\varepsilon)F^\star$ and runs in $\tilde{O}(m^{3/2}\text{poly}(1/\varepsilon))$ time.

Theorem 12.2. There is an algorithm such that, given an undirected, capacitated graph $G = (V,E,c)$, two vertices s,t, and an $\varepsilon > 0$, such that there is a cut separating s and t of value $F^\star$, the algorithm outputs an s,t-cut of value at most $(1+\varepsilon)F^\star$ and runs in $\tilde{O}(m^{3/2}\text{poly}(1/\varepsilon))$ time.

It is important to note that these algorithms work only for undirected graphs. Since we only compute a $1 \pm \varepsilon$-approximation to the max-flow–min-cut problems, there is a simple trick to ensure that the ratio of the largest to the smallest capacity in the input graph is at most $O(m^2/\varepsilon)$. Hence, the running times do not depend on the capacities. Proving this is left as an exercise. Finally, the algorithms in these theorems can be modified to obtain a running time which depends on $m^{4/3}$ rather than $m^{3/2}$; refer to the notes at the end of this section. At this point the reader may review the connection between graphs and electrical networks in Section 4.

12.2 Combinatorial versus Electrical Flows

Recall that given resistances r_e for edges in G captured by an $m \times m$ diagonal matrix R where $R(e,e) \stackrel{\text{def}}{=} r_e$ and 0 otherwise, if F units of current is injected at s and taken out at t, then the electrical flow vector $\mathbf{f}$ can be written as $\mathbf{f} \stackrel{\text{def}}{=} R^{-1}BL^{\top}F(\mathbf{e}_s - \mathbf{e}_t)$. Here, the Laplacian is defined to be $L \stackrel{\text{def}}{=} B^{\top}R^{-1}B$. Using Theorem 11.1 from Section 11, we know that $\mathbf{f}$ can be computed approximately to a precision of ε in $\tilde{O}(m \log 1/\varepsilon)$ time. How good a substitute is $\mathbf{f}$ for a combinatorial s, t-flow? First note that in the input to the s,t-MaxFlow problem there are no resistances. We will have to determine how to set them. The problem with electrical flows is that they could violate edge capacities. There does not seem to be an easy way to find resistances such that the electrical flow obtained by the electrical network satisfies the edge capacities and also maximizes the s,t-flow. In fact, it is not immediately clear that such resistances exist; we leave showing that they do as an exercise to the reader. The algorithm in the proof of Theorem 12.1 is iterative and starts with a set of resistances. At each iteration it computes the s,t-electrical flow and updates the resistances of the edges based on the congestion due to this flow. The update rule is governed by the intuition that the higher the resistance of an edge, the less the electrical flow will run through it. The ingenuity is in defining an update rule such that the number of iterations is small and the average of the electrical flows computed satisfies the capacity constraints.

Energy of Flows

In Section 4 we proved (see Theorem 4.7) that for graph $G = (V,E)$ with resistances r_e given by a diagonal matrix R, if $\mathbf{f}^{\star} \stackrel{\text{def}}{=} R^{-1}BL^{+}(\mathbf{e}_s - \mathbf{e}_t)$, then $\mathbf{f}^{\star}$ minimizes $E_{\mathbf{r}}(\mathbf{f}) \stackrel{\text{def}}{=} \sum_e r_e f_e^2$ among all unit flows $\mathbf{f}$ from s to t. The same also holds for flows of magnitude F. Thus, any s,t flow that respects capacity and pumps the same flow as the electrical flow from s to t has at least the energy of the corresponding electrical flow. Let $\mathbf{g}$ be such a combinatorial flow, i.e., $|g_e| \leq 1$. Then we have that

$$E_{\mathbf{r}}(\mathbf{f}^{\star}) \leq \sum_e r_e g_e^2 \leq \sum_e r_e. \tag{12.1}$$

The last inequality follows from the fact that $|g_e| \leq c_e$. Thus, if $E_{\mathbf{r}}(\mathbf{f}^\star) > \sum_e r_e$, we are certain there is no s,t-max-flow of value corresponding to that of $\mathbf{f}^\star$. We use this observation crucially in the algorithm.

12.3 s,t-MaxFlow

In this section we prove Theorem 12.1. Let $F^\star$ be the value of the s,t-max-flow in G. We present an algorithm that finds a $(1-\varepsilon)F^\star$ s,t-flow in time $\tilde{O}(m^{3/2}\text{poly}(1/\varepsilon))$. For convenience, we assume that $c_e = 1$, $\forall e$. Refer to Section 12.3.1 for general c_e.

It suffices to present an algorithm which, given a value F, either outputs an s,t-flow of value at least $(1-O(\varepsilon))F$ satisfying the capacity constraints or certifies that $F > F^\star$. The algorithm is iterative and appeals to the multiplicative weight update (MWU) method. The MWU technique is very general and here we present it only from the point of view of our application. At any given time t, the resistances for the edges are given by a vector $\mathbf{r}^t$. The corresponding $m \times m$ diagonal matrix is denoted by R_t and the Laplacian $L_t \stackrel{\text{def}}{=} B^\top R_t^{-1} B$. The algorithm ElecFlow appears below.

At any time t, the algorithm maintains weights w_e^t which are positive numbers. To start with, w_e^0 are set to one for all e. These weights are used to compute the resistances r_e^t at time t. The natural choice for r_e^t is w_e^t itself. Unfortunately, we do not know how to show that using these resistances, the number of iterations is bounded via the MWU method. The issue is that one needs to be able to control the *width* in the MWU method. In our case the width is the maximum flow across an edge at any time. Instead, we consider the update where the resistances are set using the following equation:

$$r_e^t = w_e^t + \frac{\varepsilon}{3m}\sum_e w_e^t.$$

Thus, the resistance of an edge at time t has a *local* component (w_e^t) and a *global* component (roughly ε times the average weight). Intuitively, the global component helps reduce the gap between the local resistance and the average global resistance, thus, reducing the possibility of a large current flowing one edge compared to the rest of the

Algorithm 12.1 ElecFlow

Input: $G(V,E)$, source s, sink t, a target flow value F and $0 < \varepsilon < 1$

Output: Either an s,t-flow $\mathbf{f}$ of value at least $(1 - O(\varepsilon))F$ or FAIL indicating that $F > F^\star$

$w_e^0 \leftarrow 1$ for all $e \in E$
$\rho \leftarrow 2\sqrt{\frac{m}{\varepsilon}}$
$T \leftarrow \frac{\rho \log m}{\varepsilon^2}$
for $t = 0 \to T - 1$ **do**
  $\forall e \in E$, $r_e^t \leftarrow w_e^t + \frac{\varepsilon}{3m}\sum_e w_e^t$
  $\mathbf{f}^t \stackrel{\text{def}}{=} R_t^{-1} B L_t^+ F(\mathbf{e}_s - \mathbf{e}_t)$
  if $E_{\mathbf{r}^t}(\mathbf{f}^t) > (1 + \frac{\varepsilon}{3})\sum_e r_e^t$ **then**
    return FAIL
  else
    $\forall e \in E$, $w_e^{t+1} \leftarrow w_e^t(1 + \frac{\varepsilon |f_e^t|}{\rho})$
  end if
end for
return $\mathbf{f} \stackrel{\text{def}}{=} \frac{(1-\varepsilon)}{(1+2\varepsilon)} \cdot \frac{1}{T} \cdot \sum_{t=0}^{T-1} \mathbf{f}^t$

graph. This allows us to bound the width to $\rho \stackrel{\text{def}}{=} O(\sqrt{m/\varepsilon})$ and the average congestion w.r.t. w_e^ts of the flow to $(1+\varepsilon)$. Hence, the number of iterations is $\tilde{O}(\sqrt{m}/\varepsilon^{5/2})$. Concretely, the upper bound on the width allows one to almost reverse the inequality $1 + x \leq e^x$, and its use is demonstrated in Lemma 12.4. The key to this algorithm is the update step and the width calculation. Once the resistances are set, the corresponding electrical flow $\mathbf{f}^t$ is computed and the weights then increase by a multiplicative factor of $(1 + \varepsilon|f_e^t|/\rho)$. Thus, the higher the congestion, the higher the increase in resistance. Finally, the flow $\mathbf{f}^t$ is computed using the Laplacian solver, see Theorem 11.1. It is an approximate flow, but we ignore this issue in the interest of readability and leave it as exercise to adapt the proof in the approximate setting. Assuming we can do so in $\tilde{O}(m)$ time, the overall running time is $\tilde{O}(m^{3/2}/\varepsilon^{5/2})$. We also show in Section 12.4 that if the algorithm outputs FAIL, we can recover an s,t cut of value at most F in additional linear time.

12.3.1 General Capacities

For general c_es, the r_e^t update rule is changed to $r_e^t \leftarrow \frac{1}{c_e^2}(w_e^t + \frac{\varepsilon}{3m}\sum_e w_e^t)$ and $|f_e^t|$ is replaced by $|f_e^t|/c_e$. Note that the new bound for the energy is $E_{\mathbf{r}}(\mathbf{f}) \leq \sum_e r_e f_e^2/c_e^2$. It is easy to verify that these changes make the algorithm work for general capacities.

12.3.2 Analysis

Since we do not know $F^\star$, the maximum flow from s to t, we first need to show that the number of Fs we may need to run is bounded by $\text{poly}(\log m, 1/\varepsilon)$. This can be done by binary search and is left as an exercise. Next, we need to show the following for any F for which the algorithm terminates but does not output FAIL:

- The flow value from s to t is at least $(1 - O(\varepsilon))F$; and
- the output flow does not violate any capacity constraints, i.e., $\forall e, |f_e| \leq \frac{(1-\varepsilon)}{(1+2\varepsilon)} \cdot \frac{1}{T} \cdot \sum_{t=0}^{T-1} |f_e^t| \leq c_e = 1$.

The first is easily seen as the flow output by the algorithm is essentially an average of T flows, each pushing F units of flow from s to t. Recall that we showed in Equation (12.1) that when the algorithm finds a flow $\mathbf{f}^t$ such that $E_{\mathbf{r}^t}(\mathbf{f}^t) > (1 + \frac{\varepsilon}{3})\sum_e r_e^t$, it implies that the s,t-max-flow is less than F. Hence, when ElecFlow returns FAIL, $F^\star < F$. On the other hand, when $E_{\mathbf{r}^t}(\mathbf{f}^t) \leq (1 + \frac{\varepsilon}{3})\sum_e r_e^t$ we can show that, in $\mathbf{f}^t$, the maximum congestion on an edge is bounded above and average congestion is small.

Lemma 12.3. If $E_{\mathbf{r}^t}(\mathbf{f}^t) \leq (1 + \frac{\varepsilon}{3})\sum_e r_e^t$, then

- $\max_e |f_e^t| \leq 2\sqrt{\frac{m}{\varepsilon}}$, and
- $\frac{\sum_e w_e^t |f_e^t|}{\sum_e w_e^t} \leq 1 + \varepsilon$.

Proof. For convenience, we drop the superscript t in this proof. The hypothesis then implies that

$$E_{\mathbf{r}}(\mathbf{f}) = \sum_e r_e f_e^2 \leq \left(1 + \frac{\varepsilon}{3}\right)\sum_e r_e.$$

Hence, using the update rule for r_e from the algorithm, we get that $\sum_e r_e f_e^2$ is at most

$$\left(1+\frac{\varepsilon}{3}\right)\sum_e\left(w_e+\frac{\varepsilon}{3m}\sum_e w_e\right)=\left(1+\frac{\varepsilon}{3}\right)^2\sum_e w_e\le(1+\varepsilon)\sum_e w_e,$$

where the last inequality follows from the fact that $\varepsilon<1$. On the other hand,

$$\sum_e r_e f_e^2=\sum_e\left(w_e+\frac{\varepsilon}{3m}\sum_e w_e\right)f_e^2. \tag{12.2}$$

Combining the two, we obtain that

$$\forall e,\quad \frac{\varepsilon f_e^2}{3m}\sum_e w_e\le\left(1+\frac{\varepsilon}{3}\right)\sum_e w_e,$$

which implies that $|f_e|\le\sqrt{\frac{3(1+\varepsilon/3)m}{\varepsilon}}\le 2\sqrt{m/\varepsilon}$ for all e. Using the Cauchy–Schwarz inequality on $\sqrt{w}_e$ and $\sqrt{w}_e|f_e|$, we get

$$\left(\sum_e w_e|f_e|\right)^2\le\left(\sum_e w_e\right)\left(\sum_e w_e f_e^2\right),$$

which implies that

$$\frac{(\sum_e w_e|f_e|)^2}{\sum_e w_e}\le\sum_e w_e f_e^2\le(1+\varepsilon)\sum_e w_e.$$

The last inequality follows from Equation (12.2) by taking the first term of summation. Hence,

$$\frac{\sum_e w_e|f_e|}{\sum_e w_e}\le\sqrt{1+\varepsilon}\le 1+\varepsilon. \qquad \square$$

To track progress, we use the potential function $\Phi_t\stackrel{\text{def}}{=}\sum_e w_e^t$. Clearly, $\Phi_0=m$ and

$$\Phi_{t+1}=\sum_e w_e^{t+1}=\sum_e w_e^t\left(1+\frac{\varepsilon|f_e^t|}{\rho}\right)=\sum_e w_e^t+\frac{\varepsilon}{\rho}\sum_e w_e^t|f_e^t|,$$

where $\rho = 2\sqrt{m/\varepsilon}$. Thus, using Lemma 12.3 we get that Φ_{t+1} is at most

$$\sum_e w_e^t + \frac{\varepsilon(1+\varepsilon)\sum_e w_e^t}{\rho} = \Phi_t\left(1 + \frac{\varepsilon(1+\varepsilon)}{\rho}\right) \leq m\left(1 + \frac{\varepsilon(1+\varepsilon)}{\rho}\right)^{t+1},$$

by repeating this argument. Hence, we obtain the following lemma.

Lemma 12.4. $\Phi_{T-1} \leq m\left(1 + \frac{\varepsilon(1+\varepsilon)}{\rho}\right)^{T-1} \leq me^{\frac{\varepsilon(1+\varepsilon)T}{\rho}}$.

In the other direction,

$$\forall e,\ w_e^{T-1} = \Pi_{t=0}^{T-1}\left(1 + \frac{\varepsilon|f_e^t|}{\rho}\right) \leq \sum_e w_e^{T-1} = \Phi_{T-1}.$$

Using the inequality $1 + \varepsilon x \geq e^{\varepsilon(1-\varepsilon)x}$ for all $x \in [0,1]$, and $0 < \varepsilon < 1$, we get that

$$\Pi_{t=0}^{T-1}\left(1 + \frac{\varepsilon|f_e^t|}{\rho}\right) \geq \Pi_{t=0}^{T-1} e^{\frac{(1-\varepsilon)\varepsilon|f_e^t|}{\rho}}.$$

Combining the last two inequalities we obtain that for all e,

$$\Phi_{T-1} \geq w_e^{T-1} \geq \Pi_{t=0}^{T-1} e^{\frac{(1-\varepsilon)\varepsilon|f_e^t|}{\rho}} = e^{\frac{(1-\varepsilon)\varepsilon\sum_t|f_e^t|}{\rho}}.$$

Now, using Lemma 12.4, we get that

$$e^{\frac{(1-\varepsilon)\varepsilon\sum_t|f_e^t|}{\rho}} \leq me^{\frac{\varepsilon(1+\varepsilon)T}{\rho}} = e^{\frac{\varepsilon(1+\varepsilon)T}{\rho} + \log m},$$

which implies that

$$\frac{(1-\varepsilon)\varepsilon\sum_t f_e^t}{\rho} \leq \frac{\varepsilon(1+\varepsilon)T}{\rho} + \log m.$$

Dividing by $T\varepsilon$ and multiplying by ρ, we get

$$\frac{(1-\varepsilon)\sum_t|f_e^t|}{T-1} \leq (1+\varepsilon) + \frac{\rho\log m}{T\varepsilon},$$

and by setting $\frac{\rho \log m}{T\varepsilon} \leq \varepsilon$, we get that for $T \geq \frac{\rho \log m}{\varepsilon^2}$,

$$\frac{(1-\varepsilon)\sum_t |f_e^t|}{(1+2\varepsilon)T} \leq 1.$$

Therefore, $\frac{\rho \log m}{\varepsilon^2}$ iterations suffice. This concludes the proof of Theorem 12.1.

Note that the number of iterations of the algorithm depends on the value of ρ. An immediate question is: Can it be decreased? The following example shows that the width $\rho = \sqrt{m}$ is tight.

Example 12.5. Consider a graph composed of $k+1$ disjoint paths between s and t; k paths of length k and one path of length 1. Hence, the total number of edges is $m = k^2 + 1$ and the max-flow in this graph is $k+1$. The electrical flow of value $k+1$ sends $\frac{k+1}{2}$ flow through the edge of the length 1 path, which is approximately $\sqrt{m}$.

12.4 s,t-Min Cut

In this section we show how to complete the proof of Theorem 12.2. The algorithm is the same as ElecFlow except now if a valid flow is returned for some F, we know that the minimum cut separating s and t is more than F. On the other hand, if for some value F ElecFlow returns FAIL, we show how to recover an s,t-cut of value $(1+O(\varepsilon))F$, thus completing the proof of Theorem 12.2.

Note that the vector of potentials used for setting $\mathbf{r}$ when ElecFlow outputs FAIL is given by $\mathbf{v} \stackrel{\text{def}}{=} L^+ F(\mathbf{e}_s - \mathbf{v}_t)$. Here, L^+ corresponds to the resistances r_e. We again ignore the issue that $\mathbf{v}$ is known only approximately. Hence, the cut we recover is in fact less than F. The vector $\mathbf{v}$ gives an embedding of the vertex set V on the real line. First, note that s and t are the two endpoints of this embedding.

Proposition 12.6. Given the embedding of V into $\mathbb{R}$ given by $\mathbf{v}$ as above, $\max_i v_i = v_s$ and $\min_i v_i = v_t$.

This is an easy exercise and can be proved using the fact that a harmonic function achieves its maxima and minima at the boundary. If we

pick a value uniformly at random from the interval $[v_t, v_s]$, then the probability that an edge $e = ij$ is cut is $\frac{|v_i - v_j|}{|v_s - v_t|}$. Let X_e be indicator random variable that is 1 if e is cut, and 0 otherwise. The expectation of X_e is $\mathbb{E}[X_e] = \mathbb{P}[X_e = 1] = \frac{|v_i - v_j|}{|v_s - v_t|}$. Using linearity of expectation, we get that

$$\mathbb{E}\left[\sum_e X_e\right] = \sum_e \mathbb{E}[X_e] = \frac{\sum_{e=ij} |v_i - v_j|}{|v_s - v_t|}.$$

Thus, using Cauchy–Schwarz inequality on $\frac{|v_i - v_j|}{\sqrt{r}_e}$ and $\sqrt{r}_e$, we get

$$\frac{\sum_{e=ij} |v_i - v_j|}{|v_s - v_t|} \leq \frac{1}{|v_s - v_t|} \sqrt{\sum_{e=ij} \frac{(v_i - v_j)^2}{r_e}} \sqrt{\sum_e r_e}.$$

Now using the fact that $E_{\mathbf{r}}(\mathbf{f}) = \sum_{e=ij} \frac{(v_i - v_j)^2}{r_e}$ and also $E_{\mathbf{r}}(\mathbf{f}) = F|v_s - v_t|$, we get that the above is at most

$$\frac{F}{E_{\mathbf{r}}(\mathbf{f})} \sqrt{E_{\mathbf{r}}(\mathbf{f})} \sqrt{\sum_e r_e} = F \sqrt{\frac{\sum_e r_e}{E_{\mathbf{r}}(\mathbf{f})}}.$$

Since $\mathbf{f}$ is such that $E_{\mathbf{r}}(\mathbf{f}) > \sum_e r_e$, finally we get $\mathbb{E}[\sum_e X_e] < F$; i.e., the expected value of cut is at most F. Therefore, one of the sweep cuts that separates s and t has to be less than the expected value. Given the vector $\mathbf{v}$, the running time of this algorithm is $\tilde{O}(m)$. Thus, for a given value of F, either ElecFlow returns a valid s,t-flow of value at least $(1 - O(\varepsilon))F$ or the algorithm in this section finds a cut of value at most F. The running time is dominated by that of ElecFlow. This completes the proof of Theorem 12.2.

Notes

The max-flow/min-cut theorem dates back to at least [26, 29], and a proof of it can be found in any text book on algorithms, see [5].

Theorems 12.1 and 12.2 are from [22]. In fact, there the power in the running time is 4/3 as opposed to the 3/2 presented in this section. Using Laplacian solvers to speed up interior point methods to solve the linear program for the max-flow problem, Daitch and Spielman [24] obtained

the $3/2$ result, which preceded the work of Christiano et al. [22]. The techniques in this section have been extended by Kelner et al. [46] to obtain algorithms for multicommodity flows for a constant number of commodities.

Two challenging problems remain: To decrease the dependence on the error ε from poly $1/\varepsilon$ to poly $\log 1/\varepsilon$; and to improve the running time to $\tilde{O}(m)$ with possibly worse dependence on ε.

The multiplicative weight update paradigm has had a remarkable success in a wide variety of areas. A recent survey of Arora et al. [11] should give the interested reader a good idea of its usefulness.

Part III

Tools

13

Cholesky Decomposition Based Linear Solvers

The remainder of this monograph is concerned with methods to solve a system of linear equations, $A\mathbf{x} = \mathbf{b}$ where A is an $n \times n$ matrix and $\mathbf{b}$ is a vector in the column space of A. This section reviews an exact method for solving a linear system of equations based on Cholesky decomposition. Subsequently, the Cholesky-based method is employed to present an $O(n)$ time algorithm for solving a linear system of equations $L\mathbf{x} = \mathbf{b}$ when L is a Laplacian of a tree.

13.1 Cholesky Decomposition

We are concerned with matrices which are symmetric and PSD. A matrix A is lower (respectively, upper) triangular if $A_{ij} = 0$ whenever $i < j$ (respectively, $i > j$). Observe that if A is either lower or upper triangular, then $A\mathbf{x} = \mathbf{b}$ can be solved by back-substitution with $O(m)$ arithmetic operations, where m is the number of nonzero entries of A. Equivalently, $A^{+}\mathbf{b}$ can be evaluated using $O(m)$ arithmetic operations. The undergraduate textbook algorithm to solve a system of linear equations is Gaussian elimination which performs row operations on the matrix A to convert it to a lower or upper triangular matrix. When A

is symmetric and PSD, a more systematic way of solving $A\mathbf{x} = \mathbf{b}$ is via its *Cholesky decomposition*, which is a modification of Gaussian elimination. We present an algorithm to compute the Cholesky decomposition of A when A is positive definite. There are appropriate generalizations when A is not invertible. However, we will not cover these generalizations since in order to solve a Laplacian system of a connected graph $L\mathbf{x} = \mathbf{b}$, the vector $\mathbf{b}$ is orthogonal to the all-ones vector. In this setting $L^+ \succ 0$.

Theorem 13.1. If $A \succ 0$, then one can write $A = \Lambda\Lambda^\top$ where Λ is a lower triangular matrix. Such a decomposition is called the Cholesky decomposition of A.

Before we proceed with the proof of this theorem, we need the following lemma, often referred to as Schur's lemma.

Lemma 13.2. $A = \begin{pmatrix} d_1 & \mathbf{u}_1^\top \\ \mathbf{u}_1 & B_1 \end{pmatrix} \succ 0$ iff

$$d_1 > 0 \quad \text{and} \quad B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1 \succ 0.$$

Proof. Since $A \succ 0$, $d_1 > 0$. Consider minimizing the quadratic form $z^2 d_1 + 2z\mathbf{u}_1^\top\mathbf{y} + \mathbf{y}^\top B_1\mathbf{y}$ over the variable z for a fixed $\mathbf{y}$. The solution can be seen, by differentiating, to be $z = -\mathbf{u}_1^\top\mathbf{y}/d_1$. The minimum value then is $\mathbf{y}^\top(B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1)\mathbf{y}$. Hence, if $A \succ 0$, then for all $\mathbf{y}$, $\mathbf{y}^\top(B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1)\mathbf{y} > 0$. This implies that $B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1 \succ 0$. The other direction is straightforward. □

Proof. [of Theorem 13.1] Since $A \succ 0$, it suffices to express $A = \Lambda\Delta\Lambda^\top$ for a diagonal matrix Δ. Positive definiteness implies that $\Delta_{ii} > 0$ and thus the decomposition in the theorem is obtained via $A = (\Lambda\Delta^{1/2})(\Lambda\Delta^{1/2})^\top$. For a symmetric, positive-definite matrix A, we write it as

$$\begin{pmatrix} d_1 & \mathbf{u}_1^\top \\ \mathbf{u}_1 & B_1 \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{0}^\top \\ \mathbf{u}_1/d_1 & I_{n-1} \end{pmatrix} \begin{pmatrix} d_1 & \mathbf{0}^\top \\ \mathbf{0} & B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1 \end{pmatrix} \begin{pmatrix} 1 & \mathbf{u}_1^\top/d_1 \\ \mathbf{0} & I_{n-1} \end{pmatrix}.$$

Let the matrix in the middle of the r.h.s. be called A_1. Note that the first matrix is a lower triangular matrix Λ_1 and the third is $\Lambda_1^\top$. Lemma 13.2 implies that $B \stackrel{\text{def}}{=} B_1 - \mathbf{u}_1\mathbf{u}_1^\top/d_1 \succ 0$ which gives $A_1 \succ 0$. Recursively, we can write $B = \Lambda'\Delta'\Lambda'^\top$ as the product of $(n-1)\times(n-1)$ matrices, and

$$A_1 = \begin{pmatrix} 1 & \mathbf{0}^\top \\ \mathbf{0} & \Lambda' \end{pmatrix} \begin{pmatrix} d_1 & \mathbf{0}^\top \\ \mathbf{0} & \Delta' \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0}^\top \\ \mathbf{0} & \Lambda'^\top \end{pmatrix} \stackrel{\text{def}}{=} \Lambda''\Delta\Lambda''^\top.$$

Thus, $A = \Lambda_1\Lambda''\Delta\Lambda''^\top\Lambda_1^\top$. Since product of lower triangular matrices is lower triangular, the theorem follows. □

Given the Cholesky decomposition, one can solve $A\mathbf{x} = \mathbf{b}$ easily. Indeed, one first evaluates $\mathbf{b}' = \Lambda^+\mathbf{b}$ and then evaluates $\mathbf{x} = (\Lambda^\top)^+\mathbf{b}'$. Note that the number of such operations is of the order of nonzero entries in Λ. In the method to compute the Cholesky decomposition of A in the proof above, the choice of the first row was arbitrary. In fact, any ordering of the rows (and the same ordering of the columns) of A can be used to recover a decomposition which results in a faster algorithm to solve $A\mathbf{x} = \mathbf{b}$. Formally, the Cholesky decomposition of A and $Q^\top AQ$, (i.e., A with its rows and columns permuted by the same permutation matrix Q,) can be very different.[1] Therefore, the number of nonzero entries in the Cholesky decomposition of a matrix, called the *fill in*, depends on the permutation of rows and columns. Finding the permutation which leads to the minimum fill in is known to be NP-hard.

13.2 Fast Solvers for Tree Systems

Now we demonstrate how to use the combinatorial structure of the linear system to get a good Cholesky decomposition. This results in a fast solver for $L_T\mathbf{x} = \mathbf{b}$ when T is a tree. Any symmetric matrix A can be associated with an n vertex graph (with self-loops) where we have an edge ij of weight $A(i,j)$ between vertices i and j. The number of edges in the graph is precisely the number of nonzero entries in A. Let us now view the Cholesky decomposition as described in the proof of

[1] An $n \times n$ matrix Q is said to be a permutation matrix if all its entries are either 0 or 1 and there is exactly one 1 in each row and in each column.

Theorem 13.1 as a process which modifies the graph: After the row i has been processed, the resulting graph (which corresponds to A_1) has the following modifications:

(1) All edges ij with $j \neq i$ are deleted; this corresponds to setting $A_1(i,j) = 0$; and
(2) for every pair jk (j could be equal to k) neighboring to i a (potentially new) edge is modified; this corresponds to setting $A_1(j,k) = B_1(j,k) - \frac{A(i,j)A(i,k)}{A(i,i)}$.

Now suppose the graph corresponding to the linear system as described above is a tree and potentially self-loops. Then in every iteration of the Cholesky decomposition, we can choose a leaf node i. Since there is a single node adjacent to i, the graph associated with the matrix A_1 is a tree as well (potentially with an extra self-loop). In particular, this implies that we can write A as $\Lambda\Delta\Lambda^\top$ where $\Lambda = \Lambda_1\Lambda_2\cdots\Lambda_n$ and each Λ_i is a lower triangular matrix with at most one nonzero off-diagonal entry. This gives a Cholesky decomposition with at most $O(n)$ nonzero entries. Moreover, in the computation of the Cholesky decomposition, in each iteration, at most $O(1)$ operations are done. Therefore, the decomposition can be computed in $O(n)$ time. Thus, we have proved the following theorem.

Theorem 13.3. Given a symmetric, PSD matrix A and a vector $\mathbf{b}$ such that the graph of A corresponds to a tree, one can find in $O(n)$ time a permutation matrix Q such that the Cholesky decomposition of $Q^\top AQ$ has at most $O(n)$ nonzero entries.

This immediately implies the following corollary which will be used in Section 17.

Corollary 13.4. If L_T is the Laplacian of a tree T and $\mathbf{b}$ is a vector such that $\langle \mathbf{b}, \mathbf{1} \rangle = 0$, then the solution of $L_T\mathbf{x} = \mathbf{b}$ can found in $O(n)$ time.

Proof. Note that the graph associated with the Laplacian of a tree is the tree itself. Therefore, using Theorem 13.3, we can find the Cholesky

decomposition of the permuted Laplacian to get $\Lambda\Lambda^\top \mathbf{y} = \mathbf{b}'$ in $O(n)$ time. Here, $\mathbf{y} \stackrel{\text{def}}{=} Q^\top \mathbf{x}$ and $\mathbf{b}' \stackrel{\text{def}}{=} Q^\top \mathbf{b}$. Note that Λ is not full rank since L_T is not, however, $\langle \mathbf{b}, \mathbf{1} \rangle = 0$ implies that $\mathbf{b}$ (and thus $\mathbf{b}'$) is in the column space of $L_T Q$. Therefore, we are guaranteed a solution $\mathbf{x}$ and this can be calculated in the number of nonzero entries of L_T, which is also $O(n)$. □

Notes

More on Cholesky decomposition of factorization can be found in the book [35]. George [33] pioneered the use of finding the right ordering in the Cholesky decomposition for special graphs, e.g., square grids. He showed that any PSD matrix whose graph is an $n \times n$ grid can be solved in $O(n^{1.5})$ time. This is the *nested dissection* algorithm. Lipton et al. [52] generalized this result to planar graphs by showing the existence of $O(\sqrt{n})$-sized separators in planar graphs. This gives an algorithm that runs in $\tilde{O}(n^{1.5})$ time and, in general, gives an algorithm that runs roughly in time $n^{1+\varepsilon}$ for a family of graphs which have separators of size n^ε and are closed under edge removal.

14

Iterative Linear Solvers I
The Kaczmarz Method

An old and simple iterative method to solve $A\mathbf{x} = \mathbf{b}$, due to Kaczmarz, starts with an arbitrary point, finds an equation that is not satisfied, forces the current solution to satisfy it, and repeats. In this section, the convergence of a randomized variant of the Kaczmarz is established. Subsequently, it is shown how this method has been recently employed to develop a new and simpler algorithm to solve Laplacian systems in approximately $\tilde{O}(m)$ time.

14.1 A Randomized Kaczmarz Method

Let $A\mathbf{x} = \mathbf{b}$ be a system of equations where A is an $m \times n$ matrix and $\mathbf{x}^\star$ is one of its solutions. It is convenient to think of the solution $\mathbf{x}^\star$ as lying on all the hyperplanes

$$\langle \mathbf{a}_i, \mathbf{x}^\star \rangle = b_i,$$

where $\mathbf{a}_i$ is the i-th row of A and b_i is the corresponding entry of $\mathbf{b}$. Thus, starting with an arbitrary initial point $\mathbf{x}_0$, consider the following

simple iterative algorithm to compute an approximation to $\mathbf{x}^\star$: Given $\mathbf{x}_t$ such that $A\mathbf{x}_t \neq \mathbf{b}$, choose any hyperplane $\langle \mathbf{a}_i, \mathbf{x} \rangle = b_i$ which does not contain $\mathbf{x}_t$ and let $\mathbf{x}_{t+1}$ be the orthogonal projection of $\mathbf{x}_t$ onto this hyperplane. It can be shown that this method will converge to $\mathbf{x}^\star$ as long as every hyperplane is considered an infinite number of times. For instance, one could cycle over the hyperplanes in a fixed order. However, the convergence to $\mathbf{x}^\star$ can be very slow and bounds on the rate of convergence for general A exist for any deterministic hyperplane selection rule. This motivates the use of randomization and we consider a simple randomized rule called the Randomized Kaczmarz method: Set $\mathbf{x}_0 = \mathbf{0}$. For $t \geq 0$, given $\mathbf{x}_t$ such that $A\mathbf{x}_t \neq \mathbf{b}$, choose a hyperplane $\langle \mathbf{a}_i, \mathbf{x} \rangle = b_i$ with probability proportional to $\|\mathbf{a}_i\|^2$ and let $\mathbf{x}_{t+1}$ be the orthogonal projection of $\mathbf{x}_t$ onto this hyperplane. The rate of convergence of this algorithm turns out to be related to a notion of an average condition number defined as follows.

Definition 14.1. Given an $m \times n$ matrix A, let $\|A^+\|$ be defined as follows

$$\|A^+\|^2 \stackrel{\text{def}}{=} \sup_{\mathbf{z} \in \mathbb{R}^n,\, A\mathbf{z} \neq \mathbf{0}} \frac{\|\mathbf{z}\|^2}{\|A\mathbf{z}\|^2}.$$

Recall that $\|A\|_F^2$ is the squared-Frobenius norm of A which is defined to be the sum of squares of all the entries of A. The average condition number is defined to be

$$\tilde{\kappa}(A) \stackrel{\text{def}}{=} \|A\|_F \|A^+\|.$$

Theorem 14.1. The Randomized Kaczmarz method presented above, starting with $\mathbf{x}_0 = \mathbf{0}$, satisfies

$$\|\mathbf{x}_\tau - \mathbf{x}^\star\|^2 \leq \varepsilon^2 \|\mathbf{x}^\star\|^2$$

for $\tau = O(\tilde{\kappa}^2(A) \log 1/\varepsilon)$ with probability at least 0.9.

14.2 Convergence in Terms of Average Condition Number

The proof of Theorem 14.1 follows from the following lemma.

Lemma 14.2. Fix the choice of the hyperplanes up to t and let $\langle \mathbf{a}_i, \mathbf{x} \rangle = b_i$ be the random hyperplane selected with probability $\|\mathbf{a}_i\|^2 / \|A\|_F^2$. If $\mathbf{x}_{t+1}$ is the orthogonal projection of $\mathbf{x}_t$ on to this hyperplane,

$$\mathbb{E}\,\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \leq (1 - 1/\tilde{\kappa}^2(A))\,\|\mathbf{x}_t - \mathbf{x}^\star\|^2.$$

As a first step in the proof of this lemma, let us calculate $\mathbf{x}_{t+1}$ in terms of the hyperplane and $\mathbf{x}_t$. The unit normal of the hyperplane $\langle \mathbf{a}_i, \mathbf{x} \rangle = b_i$ is $\hat{\mathbf{a}}_i \stackrel{\text{def}}{=} \frac{\mathbf{a}_i}{\|\mathbf{a}_i\|}$. Since $\mathbf{x}_{t+1}$ is an orthogonal projection of $\mathbf{x}_t$ onto this hyperplane, there is some $\gamma \geq 0$ such that

$$\mathbf{x}_t - \mathbf{x}_{t+1} = \gamma \cdot \hat{\mathbf{a}}_i.$$

Since $\mathbf{x}_{t+1}$ lies on the hyperplane, we know that

$$\langle \mathbf{a}_i, \mathbf{x}_{t+1} \rangle = b_i = \langle \mathbf{a}_i, \mathbf{x}^\star \rangle.$$

Thus, $\gamma = \frac{\langle \mathbf{a}_i, \mathbf{x}_t - \mathbf{x}^\star \rangle}{\|\mathbf{a}_i\|}$, implying that

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\langle \mathbf{a}_i, \mathbf{x}_t - \mathbf{x}^\star \rangle}{\|\mathbf{a}_i\|^2} \mathbf{a}_i. \tag{14.1}$$

Since $\mathbf{x}_{t+1}$ and $\mathbf{x}^\star$ lie in the hyperplane, $\langle \mathbf{a}_i, \mathbf{x}_{t+1} - \mathbf{x}^\star \rangle = 0$, which implies that $\mathbf{x}_{t+1} - \mathbf{x}^\star$ is orthogonal to $\mathbf{x}_{t+1} - \mathbf{x}_t$. Thus, by Pythagoras Theorem,

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 = \|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2. \tag{14.2}$$

Since the hyperplane $\langle \mathbf{a}_i, \mathbf{x} \rangle = b_i$ was chosen with probability $\frac{\|\mathbf{a}_i\|^2}{\|A\|_F^2}$, combining Equations (14.1) and (14.2), we obtain that

$$\begin{aligned}
\mathbb{E}\,\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 &= \|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \sum_i \frac{\|\mathbf{a}_i\|^2}{\|A\|_F^2} \frac{|\langle \mathbf{a}_i, \mathbf{x}_t - \mathbf{x}^\star \rangle|^2}{\|\mathbf{a}_i\|^4} \|\mathbf{a}_i\|^2 \\
&= \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(1 - \frac{1}{\|A\|_F^2} \sum_i \frac{|\langle \mathbf{a}_i, \mathbf{x}_t - \mathbf{x}^\star \rangle|^2}{\|\mathbf{x}_t - \mathbf{x}^\star\|^2} \right)
\end{aligned}$$

$$
\begin{aligned}
&= \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(1 - \frac{1}{\|A\|_F^2} \frac{\|A(\mathbf{x}_t - \mathbf{x}^\star)\|^2}{\|\mathbf{x}_t - \mathbf{x}^\star\|^2}\right) \\
&\overset{\text{Defn.14.1}}{\leq} \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(1 - \frac{1}{\|A\|_F^2 \|A^+\|^2}\right) \\
&= \|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(1 - \frac{1}{\tilde{\kappa}^2(A)}\right).
\end{aligned}
$$

This completes the proof of the Lemma 14.2. By repeated application of this lemma we obtain that for $t \geq 0$,

$$\mathbb{E}\,\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \leq (1 - 1/\tilde{\kappa}^2(A))^t \|\mathbf{x}_0 - \mathbf{x}^\star\|^2 = (1 - 1/\tilde{\kappa}^2(A))^t \|\mathbf{x}^\star\|^2.$$

Thus, from Markov's Inequality, with probability at least $1 - 1/10$,

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \leq 10(1 - 1/\tilde{\kappa}^2(A))^t \|\mathbf{x}^\star\|^2. \tag{14.3}$$

Hence, by selecting $\tau = O(\tilde{\kappa}^2(A) \log 1/\varepsilon)$, we obtain that

$$\|\mathbf{x}_\tau - \mathbf{x}^\star\|^2 \leq \varepsilon^2 \|\mathbf{x}^\star\|^2$$

with probability at least 0.9, completing the proof of Theorem 14.1.

The Kaczmarz method restricted to a subspace. Note that the Randomized Kaczmarz method and Theorem 14.1 above can be easily extended to the case when the optimal solution $\mathbf{x}^\star$ is known to satisfy $U\mathbf{x}^\star = \mathbf{v}$ for some U and $\mathbf{v}$, and A is such that $UA^\top = \mathbf{0}$. This will be useful in the next section. In this setting, the Randomized Kaczmarz method maintains that $U\mathbf{x}_t = \mathbf{v}$ for all t. Thus, we first need an initial choice $\mathbf{x}_0$ s.t. $U\mathbf{x}_0 = \mathbf{v}$. As a consequence, $\mathbf{x}_0$ may no longer be $\mathbf{0}$ and, hence, in Equation (14.3) we can no longer replace $\|\mathbf{x}_0 - \mathbf{x}^\star\|$ by $\|\mathbf{x}^\star\|$. Further, $U\mathbf{x}_0 - U\mathbf{x}_t = \mathbf{0}$ for all t. Thus, a careful look at the proof of Theorem 14.1 suggests that the following definition of $\|A^+\|$ suffices.

$$\|A^+\|^2 \overset{\text{def}}{=} \sup_{\mathbf{z} \in \mathbb{R}^n,\, A\mathbf{z} \neq \mathbf{0},\, U\mathbf{z} = \mathbf{0}} \frac{\|\mathbf{z}\|^2}{\|A\mathbf{z}\|^2}.$$

This can be smaller, giving rise to the possibility that $\tilde{\kappa}(A)$ in this setting is smaller, which in turn could mean a reduced number of iterations. Without any additional effort, an analysis identical to that of Theorem 14.1 shows that after $\tau \sim \tilde{\kappa}^2(A) \log 1/\varepsilon$ iterations, w.h.p. $\|\mathbf{x}_\tau - \mathbf{x}^\star\|^2 \leq \varepsilon^2 \|\mathbf{x}_0 - \mathbf{x}^\star\|^2$.

14.3 Toward an $\tilde{O}(m)$-Time Laplacian Solver

In this section we illustrate how the Randomized Karcmarz method can be applied to a Laplacian system. We focus on presenting the algorithm and derive a bound on the number of iterations. This approach has recently led to a new algorithm and a simpler proof of the main theorem (Theorem 3.1) in this monograph.

For this section let us focus on the case of computing the flow through each edge when one unit of current is pumped from s to t and the edges have unit resistances. Such a computational primitive was required in Section 12. It is an easy exercise to extend this argument to solve $L\mathbf{x} = \mathbf{b}$ when $\mathbf{b}$ is not necessarily $\mathbf{e}_s - \mathbf{e}_t$. To set things up, let $G = (V, E)$ be an undirected, unweighted graph on n vertices and m edges, and let $B \in \{-1, 0, 1\}^{n \times m}$ be an edge-vertex incidence matrix corresponding to a fixed orientation of the edges. Recall from Section 4 the unit s, t flow vector $\mathbf{i} = B^\top L^+(\mathbf{e}_s - \mathbf{e}_t)$. Given $\mathbf{i}$, it is easy to compute $L^+(\mathbf{e}_s - \mathbf{e}_t)$ in $O(m)$ time. Thus, let us focus on computing an approximation to $\mathbf{i}$.

For an undirected cycle C in G, let $\mathbf{1}_C \in \{-1, 0, 1\}^m$ be a vector supported exactly on the edges of C that satisfies $B\mathbf{1}_C = 0$. Since the sum of voltages across any cycle sum up to zero if all resistances are one, $\langle \mathbf{1}_C, \mathbf{i} \rangle = 0$ for all cycles C in G. Fix a spanning tree T in G and, for every $e \notin T$, let C_e be the unique cycle formed by adding e to T. The indicator vectors for these $m - n + 1$ cycles form a basis of the space of cycle vectors and can be used to generate the indicator vector of any cycle in G. Hence, to find $\mathbf{i}$ it is sufficient to solve the following system of equations

$$\forall e \notin T \quad \langle \mathbf{1}_{C_e}, \mathbf{f} \rangle = 0 \quad \text{s.t. } B\mathbf{f} = \mathbf{e}_s - \mathbf{e}_t.$$

Let A be an $m - n + 1 \times m$ matrix whose rows are indexed by edges $e \notin T$ and are equal to $\mathbf{1}_{C_e}$. Then the problem reduces to solving $A\mathbf{f} = \mathbf{0}$ subject to $B\mathbf{f} = \mathbf{e}_s - \mathbf{e}_t$. We will apply the Randomized Karcmarz method to the system $A\mathbf{f} = \mathbf{0}$ while working in the space $B\mathbf{f} = \mathbf{e}_s - \mathbf{e}_t$: Initially choose $\mathbf{f}_0$ such that $B\mathbf{f}_0 = \mathbf{e}_s - \mathbf{e}_t$. One such choice is to put a unit flow from s to t on the unique path from s to t in T. At time t, given $\mathbf{f}_t$, compute $\mathbf{f}_{t+1}$ by orthogonally projecting $\mathbf{f}_t$ onto the hyperplane

$\langle \mathbf{1}_C, \mathbf{f} \rangle = 0$, where C is chosen from one of the $m - n + 1$ cycles with probability proportional to $\|\mathbf{1}_C\|^2 = |C|$. The number of iterations necessary to get within ε of the optimal solution, from Theorem 14.1, is $O(\tilde{\kappa}^2(A) \log 1/\varepsilon)$.

In the remainder of the section we calculate $\tilde{\kappa}^2(A)$. Recall that $\tilde{\kappa}^2(A) = \|A\|_F^2 \|A^+\|^2$. Fixing some T, $\|A\|_F^2$ equals

$$\sum_{e \notin T} \|\mathbf{1}_{C_e}\|^2 = \sum_{e \notin T} |C| = m - n + 1 + \sum_{e \notin T} \mathrm{str}_T(e),$$

where $\mathrm{str}_T(e)$ is the *stretch* of e in T defined as the length of the unique path in T between the endpoints of e.[1] On the other hand,

$$\|A^+\|^2 = \sup_{\mathbf{f},\, A\mathbf{f} \neq \mathbf{0},\, B\mathbf{f} = \mathbf{0}} \frac{\|\mathbf{f}\|^2}{\|A\mathbf{f}\|^2}.$$

We will now show that this is at most 1.

Let $\mathbf{f}_T$ denote the entries of $\mathbf{f}$ corresponding to the tree edges and $\mathbf{f}_N$ denote those corresponding to the non-tree edges. Similarly, we can split the columns of A into those corresponding to tree edges and non-tree edges. It follows that one can write $A = [A_T\ I_N]$, where I_N is a diagonal matrix on the non-tree edges, if the direction of each cycle is chosen to coincide with the edge that was added to form it from T. Suppose $\mathbf{f}$ is supported only on one of the cycles determined by T, then $A_T^\top \mathbf{f}_N = \mathbf{f}_T$. Thus, by linearity, the following holds for any vector $\mathbf{f}$ such that $B\mathbf{f} = \mathbf{0}$,

$$A_T^\top \mathbf{f}_N = \mathbf{f}_T. \tag{14.4}$$

Note that $\|A\mathbf{f}\|^2 = \|A_T \mathbf{f}_T\|^2 + 2\mathbf{f}_T^\top A_T^\top \mathbf{f}_N + \|\mathbf{f}_N\|^2$. Thus, by (14.4),

$$\|A\mathbf{f}\|^2 = \|A_T \mathbf{f}_T\|^2 + \|\mathbf{f}_T\|^2 + \|\mathbf{f}_N\|^2 \geq \|\mathbf{f}_T\|^2 + \|\mathbf{f}_N\|^2 = \|\mathbf{f}\|^2.$$

This proves the claim that $\tilde{\kappa}^2(A) \leq m - n + 1 + \sum_{e \notin T} \mathrm{str}_T(e)$. We will see in Theorem 17.3 (without proof) that one can construct, in $\tilde{O}(m)$ time, trees with $\sum_{e \notin T} \mathrm{str}_T(e) = \tilde{O}(m)$. Since, $\|\mathbf{f}_0\|^2 \leq n$ as the initial flow is just on a path from s to t, applying Theorem 14.1 in the subspace restricted setting of the previous section, the number of iterations

[1] This notion of stretch will play a central role in the proof of Theorem 3.1 presented in this monograph and more on this appears in Section 17.

required by the Randomized Kaczmarz method to get within ε can be seen to be bounded by $\tilde{O}(m \log 1/\varepsilon)$

In order to complete the proof of Theorem 3.1 the key issues that remain are, given T, how to choose an edge not in T with probability proportional to the length of the cycle it forms with T and how to update $\mathbf{f}_{t+1}$ from $\mathbf{f}_t$. The former is a simple exercise. For the latter, note that the updates involve simply adding or removing flow along cycles, which can be implemented in logarithmic time using a data structure called *link-cut trees*. The details of this data structure and how it is employed are beyond the scope of this monograph.

Notes

The Kaczmarz method appears in [43]. The Randomized Kaczmarz procedure and its analysis appear in [83]. This analysis is tied to the *alternating projections* framework [19], which also has many other algorithmic applications. Section 14.3 is based on a recent result due to [47] where details about the *link-cut tree* data structures can be found. It is also shown in [47] that the inverter derived from the Randomized Kaczmarz method is linear in the sense of Section 3.4.

15

Iterative Linear Solvers II
The Gradient Method

This section phrases the problem of solving a system of linear equations as an optimization problem and uses the technique of gradient descent to develop an iterative linear solver. Roughly, iterative methods to solve a system $A\mathbf{x} = \mathbf{b}$ maintain a guess $\mathbf{x}_t$ for the solution at an iteration t, and update to a new guess $\mathbf{x}_{t+1}$ through feedback about how good their guess was. The update typically uses a simple vector–matrix product and is often very fast and popular in practice due to its small space complexity.

15.1 Optimization View of Equation Solving

In this section we assume that A is symmetric and positive definite and, as usual, we let the eigenvalues of A be $0 < \lambda_1 \leq \cdots \leq \lambda_n$. To start, observe that solving $A\mathbf{x} = \mathbf{b}$ is equivalent to finding the minimum of $f(\mathbf{x})$ where f is defined to be

$$f(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}\mathbf{x}.$$

Observe that when A is positive definite, $\nabla^2 f = A \succ 0$, so f is strictly convex and, thus, has a unique minimum $\mathbf{x}^\star$. This $\mathbf{x}^\star$ must satisfy $\nabla f(\mathbf{x}^\star) = A\mathbf{x}^\star - \mathbf{b} = 0$.

Since f is a convex function, we can use the gradient descent approach popular in convex optimization to solve for $\mathbf{x}^\star$. A typical gradient descent algorithm starts with an initial vector $\mathbf{x}_0$, and at iteration t moves to a new point $\mathbf{x}_{t+1}$ in the direction opposite to the gradient of f at $\mathbf{x}_t$. We would like to move in this direction because f decreases along it. Note that $\nabla f(\mathbf{x}_t) = A\mathbf{x}_t - \mathbf{b}$ which means that the time it takes to compute the gradient requires a single multiplication of a vector with a matrix A. We use t_A to denote the time it takes to multiply an $n \times n$ matrix A with a vector.[1] The hope is that in a small number of iterations, each of which is geared to reducing the function value, the process will converge to $\mathbf{x}^\star$. We prove the following theorem.

Theorem 15.1. There is an algorithm GDSolve that, given an $n \times n$ symmetric matrix $A \succ 0$, a vector $\mathbf{b}$, and $\varepsilon > 0$, finds a vector $\mathbf{x}$ such that

$$\|\mathbf{x} - A^+\mathbf{b}\|_A \leq \varepsilon \|A^+\mathbf{b}\|_A$$

in time $O(t_A \cdot \kappa(A) \log 1/\varepsilon)$. Here the condition number of A is defined to be $\kappa(A) \stackrel{\text{def}}{=} \lambda_n(A)/\lambda_1(A)$ and, for a vector $\mathbf{v}$, $\|\mathbf{v}\|_A \stackrel{\text{def}}{=} \sqrt{\mathbf{v}^\top A \mathbf{v}}$.

15.2 The Gradient Descent-Based Solver

Before we describe the algorithm GDSolve formally and prove Theorem 15.1, we provide a bit more intuition. First, let us fix some notation. For $t \geq 0$, let $\mathbf{d}_t \stackrel{\text{def}}{=} \mathbf{x}^\star - \mathbf{x}_t$ be a vector which tells us where the current solution $\mathbf{x}_t$ is w.r.t. $\mathbf{x}^\star$, and let $\mathbf{r}_t \stackrel{\text{def}}{=} A\mathbf{d}_t = \mathbf{b} - A\mathbf{x}_t$. Observe that $\mathbf{r}_t = -\nabla f(\mathbf{x}_t)$. Think of $\mathbf{r}_t$ as a crude approximation to $\mathbf{d}_t$. At iteration t, there is a parameter η_t which determines how much to move in the direction of $\mathbf{r}_t$. We will discuss the choice of η_t shortly, but for now, the new point is

$$\mathbf{x}_{t+1} \stackrel{\text{def}}{=} \mathbf{x}_t + \eta_t \mathbf{r}_t. \tag{15.1}$$

[1] t_A is at least n and, even if A has m nonzero entries, can be much less than m, e.g., if $A = \mathbf{v}\mathbf{v}^\top$ and is specified in this form.

How does one choose η_t? One way is to choose it greedily: Given $\mathbf{x}_t$ (and therefore $\mathbf{r}_t$) choose η_t which minimizes $f(\mathbf{x}_{t+1})$. Towards this end, consider the function

$$g(\eta) \stackrel{\text{def}}{=} f(\mathbf{x}_t + \eta \mathbf{r}_t) = \frac{1}{2}(\mathbf{x}_t + \eta \mathbf{r}_t)^\top A(\mathbf{x}_t + \eta \mathbf{r}_t) - \mathbf{b}^\top(\mathbf{x}_t + \eta \mathbf{r}_t).$$

The minimum is attained when the gradient of g w.r.t. η is 0. This can be calculated easily:

$$g'(\eta) = \mathbf{r}_t^\top A\mathbf{x}_t + \eta \mathbf{r}_t^\top A\mathbf{r}_t - \mathbf{r}_t^\top \mathbf{b},$$

which implies

$$\eta_t = \frac{\mathbf{r}_t^\top(\mathbf{b} - A\mathbf{x}_t)}{\mathbf{r}_t^\top A\mathbf{r}_t} = \frac{\mathbf{r}_t^\top \mathbf{r}_t}{\mathbf{r}_t^\top A\mathbf{r}_t}.$$

GDSolve is described formally in Algorithm 15.1.

Algorithm 15.1 GDSolve

Input: Symmetric, positive-definite matrix $A \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$ and T
Output: $\mathbf{x}_T \in \mathbb{R}^n$

1: $\mathbf{x}_0 \leftarrow \mathbf{0}$
2: **for** $t = 0 \rightarrow T - 1$ **do**
3: Set $\mathbf{r}_t = \mathbf{b} - A\mathbf{x}_t$
4: Set $\eta_t = \frac{\mathbf{r}_t^\top \mathbf{r}_t}{\mathbf{r}_t^\top A\mathbf{r}_t}$
5: Set $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t \mathbf{r}_t$
6: **end for**
7: **return** $\mathbf{x}_T$

Thus, to prove Theorem 15.1 the only thing one needs to show is that for $T = O(\kappa(A) \log 1/\varepsilon)$, $\|\mathbf{x}_T - A^+\mathbf{b}\|_A \leq \varepsilon \|A^+\mathbf{b}\|_A$. Towards this, we prove Lemma 15.2 which shows how the A-norm of the error vector $\mathbf{d}_t = \mathbf{x}_t - \mathbf{x}^\star$ drops with t.

Lemma 15.2. $\|\mathbf{d}_{t+1}\|_A^2 \leq (1 - 1/\kappa(A)) \cdot \|\mathbf{d}_t\|_A^2$

Proof. We start with the following observation: From the update rule we get $\mathbf{d}_{t+1} = \mathbf{d}_t - \eta_t \mathbf{r}_t$ and $\mathbf{r}_{t+1} = \mathbf{r}_t - \eta_t A\mathbf{r}_t$. This gives, $\mathbf{r}_t^\top \mathbf{r}_{t+1} = \mathbf{r}_t^\top \mathbf{r}_t - \frac{\mathbf{r}_t^\top \mathbf{r}_t}{\mathbf{r}_t^\top A\mathbf{r}_t} \mathbf{r}_t^\top A\mathbf{r}_t = 0$. Therefore, two consecutive update directions

$\mathbf{r}_t$ and $\mathbf{r}_{t+1}$ are orthonormal. This is expected since from $\mathbf{x}_t$ we move in the direction $\mathbf{r}_t$ as far as we can to minimize f. Now,

$$\|\mathbf{d}_{t+1}\|_A^2 = \mathbf{d}_{t+1}^\top A\mathbf{d}_{t+1} = \mathbf{d}_{t+1}^\top \mathbf{r}_{t+1} = (\mathbf{d}_t - \eta_t \mathbf{r}_t)^\top \mathbf{r}_{t+1} = \mathbf{d}_t^\top \mathbf{r}_{t+1}.$$

The last equality used the orthonormality of $\mathbf{r}_t$ and $\mathbf{r}_{t+1}$. Thus,

$$\|\mathbf{d}_{t+1}\|_A^2 = \mathbf{d}_t^\top \mathbf{r}_{t+1} = \mathbf{d}_t^\top A\mathbf{d}_{t+1} = \mathbf{d}_t^\top A(\mathbf{d}_t - \eta_t \mathbf{r}_t).$$

This is the same as

$$\|\mathbf{d}_t\|_A^2 \cdot \left(1 - \eta_t \frac{\mathbf{d}_t^\top A\mathbf{r}_t}{\mathbf{d}_t^\top A\mathbf{d}_t}\right) = \|\mathbf{d}_t\|_A^2 \cdot \left(1 - \frac{\mathbf{r}_t^\top \mathbf{r}_t}{\mathbf{r}_t^\top A\mathbf{r}_t} \cdot \frac{\mathbf{d}_t^\top A^2\mathbf{d}_t}{\mathbf{d}_t^\top A\mathbf{d}_t}\right).$$

In the last inequality, we use $\mathbf{r}_t = A\mathbf{d}_t$. Now, the first fraction is at least $1/\lambda_n$, while the second is at least λ_1 since

$$\frac{\mathbf{d}_t^\top A^2\mathbf{d}_t}{\mathbf{d}_t^\top A\mathbf{d}_t} = \frac{(A^{1/2}\mathbf{d}_t)^\top A(A^{1/2}\mathbf{d}_t)}{(A^{1/2}\mathbf{d}_t)^\top (A^{1/2}\mathbf{d}_t)} \geq \min_{\mathbf{x}\neq\mathbf{0}} \frac{\mathbf{x}^\top A\mathbf{x}}{\mathbf{x}^\top \mathbf{x}} = \lambda_1.$$

Here we is where we have used that $A \succ 0$. This gives us

$$\|\mathbf{d}_{t+1}\|_A^2 \leq (1 - 1/\kappa(A))\|\mathbf{d}_t\|_A^2,$$

which completes the proof of the lemma. □

As a corollary we get that in $T = 2\kappa(A)\log 1/\varepsilon$ steps, $\|\mathbf{x}_T - \mathbf{x}^\star\|_A \leq \varepsilon\|\mathbf{x}_0 - \mathbf{x}^\star\|_A$; in fact, with a better analysis the factor 2 can be removed. This completes the proof of Theorem 15.1.

To effectively use Theorem 15.1 one needs a more tractable condition to check when to terminate. It is easy to check that $\|A\mathbf{x}_t - \mathbf{b}\| \leq \delta$. For this termination condition to imply the termination condition in the theorem, one has to choose δ small enough; this seems to require an upper bound on the condition number of A. We leave it as an exercise to check that for graph Laplacians, an easy estimate can be obtained in terms of the ratio of the largest to the smallest edge weight. In the next section we will see how to reduce the number of iterations to about $\sqrt{\kappa(A)}$ when A is PSD. This will require a deeper look into GDSolve.

Notes

The method GDSolve suggested here is well known and bears similarity with Richardson's iterative method [64], see also [36, 37]. More on gradient descent can be found in the book [19].

16

Iterative Linear Solvers III
The Conjugate Gradient Method

This section presents a sophisticated iterative technique called the conjugate gradient method which is able to approximately solve a linear system of equations in time that depends on the square root of the condition number. It introduces the notion of Krylov subspace and shows how Chebyshev polynomials play a crucial role in obtaining this square root saving. Finally, the Chebyshev iteration method is presented which is used in Section 18 to construct linear Laplacian solvers.

16.1 Krylov Subspace and A-Orthonormality

Given a symmetric and positive-definite matrix A, we start where we left off in the previous section. Recall that for a vector $\mathbf{b}$, we set up the optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x}.$$

We presented an algorithm GDSolve which, in its t-th iteration, generates $\mathbf{x}_t$ where

$$\mathbf{x}_1 = \mathbf{x}_0 + \eta_0 \mathbf{r}_0,$$
$$\mathbf{x}_2 = \mathbf{x}_1 + \eta_1 \mathbf{r}_1 = \mathbf{x}_1 + \eta_1(\mathbf{r}_0 - \eta_0 A\mathbf{r}_0) = \mathbf{x}_0 + \eta_1 \mathbf{r}_0 - \eta_1\eta_0 A\mathbf{r}_0,$$

and so on. Here $\mathbf{r}_0 = \mathbf{b}$ and $\mathbf{r}_i = \mathbf{b} - A\mathbf{x}_i$. Thus, it follows by induction that $\mathbf{x}_t$ lies in $\mathbf{x}_0 + \mathcal{K}_t$ where $\mathcal{K}_t$ is the subspace spanned by $\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{t-1}\mathbf{r}_0\} = \{\mathbf{b}, A\mathbf{b}, \ldots, A^{t-1}\mathbf{b}\}$. $\mathcal{K}_t$ is called the *Krylov subspace* of order t generated by A and $\mathbf{b}$.

In GDSolve, although at each iteration t we move greedily along the direction $\mathbf{r}_{t-1}$, the resulting point $\mathbf{x}_t$ which lies in $\mathbf{x}_0 + \mathcal{K}_t$ is not guaranteed to be a minimizer of f over this subspace. That is, there could be a $\mathbf{y} \in \mathbf{x}_0 + \mathcal{K}_t$ such that $f(\mathbf{y}) < f(\mathbf{x}_t)$. The main idea of the conjugate gradient method can be summarized in one sentence:

At step t, move to the f-minimizer over $\mathbf{x}_0 + \mathcal{K}_t$.

Note that the problem of finding $\mathbf{x}^\star$ is precisely that of finding the f-minimizer over $\mathbf{x}_0 + \mathcal{K}_n$. In this section we will see how to make this idea work and will prove the following theorem.

Theorem 16.1. There is an algorithm CGSolve that, given an $n \times n$ symmetric matrix $A \succ 0$, a vector $\mathbf{b}$, and $\varepsilon > 0$, finds a vector $\mathbf{x}$ such that

$$\|\mathbf{x} - A^+\mathbf{b}\|_A \leq \varepsilon \|A^+\mathbf{b}\|_A$$

in time $O(t_A \cdot \sqrt{\kappa(A)} \log 1/\varepsilon)$.

The first question we would like to ask is: How can we find the f-minimizer over $\mathbf{x}_0 + \mathcal{K}_t$ *quickly*? Suppose we have access to a basis $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{t-1}\}$ of $\mathcal{K}_t$ such that for any scalars $\beta_0, \ldots, \beta_{t-1}$, we have the following *decomposition*:

$$f\left(\mathbf{x}_0 + \sum_{i=0}^{t-1} \beta_i \mathbf{p}_i\right) - f(\mathbf{x}_0) = \sum_{i=0}^{t-1} (f(\mathbf{x}_0 + \beta_i \mathbf{p}_i) - f(\mathbf{x}_0)), \qquad (16.1)$$

i.e., the optimization problem becomes *separable* in the variables β_i. Then, it is sufficient to find, for all i, α_i which is the α that minimizes $f(\mathbf{x}_0 + \alpha \mathbf{p}_i) - f(\mathbf{x}_0)$: Observe that $\mathbf{x}_t \stackrel{\text{def}}{=} \mathbf{x}_0 + \sum_{i=1}^{t-1} \alpha_i \mathbf{p}_i$ minimizes $f(\mathbf{y})$ for all $\mathbf{y} \in \mathbf{x}_0 + \mathcal{K}_t$. Therefore, if we have access to a basis as described above, we are able to move to a minimizer over $\mathbf{x}_0 + \mathcal{K}_t$ iteratively.

Consider Equation (16.1). If f were a linear function, then equality clearly holds. Thus, we can just look at the quadratic portion of f to decide when this equality holds. For brevity, let $\mathbf{v}_i \stackrel{\text{def}}{=} \beta_i \mathbf{p}_i$. Evaluating the l.h.s. of Equation (16.1) gives

$$\begin{aligned}\frac{1}{2}\left(\mathbf{x} + \sum_i \mathbf{v}_i\right)^\top A \left(\mathbf{x} + \sum_i \mathbf{v}_i\right) - \frac{1}{2}\mathbf{x}^\top A\mathbf{x} \\ = \mathbf{x}^\top A\left(\sum_i \mathbf{v}_i\right) + \left(\sum_i \mathbf{v}_i\right)^\top A\left(\sum_i \mathbf{v}_i\right).\end{aligned}$$

Evaluating the r.h.s. gives $\frac{1}{2}\sum_i(\mathbf{x}^\top A\mathbf{v}_i + \mathbf{v}_i^\top A\mathbf{v}_i)$. The above equations are equal if the *cross terms* $\mathbf{v}_i^\top A\mathbf{v}_j$ are equal to 0. This is precisely the definition of A-orthonormal vectors.

Definition 16.1. Given a symmetric matrix A, a set of vectors $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_t$ are A-orthonormal iff for all $i \neq j$, $\mathbf{p}_i^\top A\mathbf{p}_j = 0$.

Now we are armed to give an overview of the conjugate gradient algorithm. Let $\mathbf{x}_0$ be the initial vector, and let $\mathbf{r}_0 \stackrel{\text{def}}{=} A(\mathbf{x}^\star - \mathbf{x}_0)$. Let $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_t$ be a set of A-orthonormal vectors spanning $\mathcal{K}_t = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{t-1}\mathbf{r}_0\}$. In the next section we will see how to compute this A-orthonormal basis for $\mathcal{K}_t$. In fact, we will compute the vector $\mathbf{p}_t$ itself in the $(t+1)$-st iteration taking $O(1)$ extra matrix–vector computations; for the time being, suppose they are given.

Let α_t be scalars that minimize $f(\mathbf{x}_0 + \alpha\mathbf{p}_t) - f(\mathbf{x}_0)$; by a simple calculation, we get $\alpha_t = \frac{\mathbf{p}_t^\top \mathbf{r}_0}{\mathbf{p}_t^\top A\mathbf{p}_t}$. The conjugate gradient algorithm updates the vectors as follows

$$\mathbf{x}_{t+1} \stackrel{\text{def}}{=} \mathbf{x}_t + \alpha_t\mathbf{p}_t \quad \text{and} \quad \alpha_t \stackrel{\text{def}}{=} \frac{\mathbf{p}_t^\top \mathbf{r}_0}{\mathbf{p}_t^\top A\mathbf{p}_t}. \tag{16.2}$$

16.2 Computing the A-Orthonormal Basis

We now show how to compute an A-orthonormal basis of $\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{t-1}\mathbf{r}_0\}$. One could use Gram–Schmidt orthonormalization which

iteratively computes $\mathbf{p}_0, \ldots, \mathbf{p}_t$, by setting

$$\mathbf{p}_{t+1} \stackrel{\text{def}}{=} \mathbf{v} - \sum_{i \leq t} \frac{\mathbf{v}^\top A \mathbf{p}_i}{\mathbf{p}_i^\top A \mathbf{p}_i} \mathbf{p}_i,$$

where $\mathbf{v}$ is the new vector which one is trying to A-orthonormalize. Given that $\mathbf{p}_j^\top A \mathbf{p}_i = 0$ for all $i \neq j \leq t$, this ensures that $\mathbf{p}_{t+1}^\top A \mathbf{p}_i = 0$ for all $i \leq t$. Note that the above can take up to $O(t)$ matrix–vector calculations to compute $\mathbf{p}_{t+1}$. We now show that if one is trying to A-orthonormalize the Krylov subspace generated by A, one can get away by performing only $O(1)$ matrix–vector computations. This relies on the fact that A is symmetric.

Let $\mathbf{p}_0 = \mathbf{r}_0$. Suppose we have constructed vectors $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_t\}$ which form an A-orthonormal basis for $\mathcal{K}_{t+1}$. Inductively assume that the vectors satisfy

$$\mathcal{K}_{i+1} = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^i \mathbf{r}_0\} = \text{span}\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_i\} \tag{16.3}$$

and that $A\mathbf{p}_i \in \mathcal{K}_{i+2}$ for all $i \leq t-1$. Note that this is true when $i = 0$ as $\mathbf{p}_0 = \mathbf{r}_0$. Let us consider the vector $A\mathbf{p}_t$. If $A\mathbf{p}_t \in \mathcal{K}_{t+1}$, $\mathcal{K}_j = \mathcal{K}_{t+1}$ for all $j > t+1$ and we can stop. On the other hand, if $A\mathbf{p}_t \notin \mathcal{K}_{t+1}$, we construct $\mathbf{p}_{t+1}$ by A-orthonormalizing it w.r.t. $\mathbf{p}_i$ for all $i \leq t$. Thus,

$$\mathbf{p}_{t+1} \stackrel{\text{def}}{=} A\mathbf{p}_t - \sum_{i \leq t} \frac{(A\mathbf{p}_t)^\top A \mathbf{p}_i}{\mathbf{p}_i^\top A \mathbf{p}_i} \mathbf{p}_i. \tag{16.4}$$

This way of picking $\mathbf{p}_{t+1}$ implies, from our assumption in Equation (16.3), that

$$\mathcal{K}_{t+2} = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{t+1}\mathbf{r}_0\} = \text{span}\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{t+1}\}.$$

It also ensures that $A\mathbf{p}_t$ can be written as a linear combination of $\mathbf{p}_i$s for $i \leq t+1$. Hence, $A\mathbf{p}_t \in \mathcal{K}_{t+2}$, proving our induction hypothesis. Thus, for every $i \leq t$, there are constants c_js such that

$$(A\mathbf{p}_t)^\top (A\mathbf{p}_i) = \mathbf{p}_t^\top A^\top (A\mathbf{p}_i) = \mathbf{p}_t^\top A (A\mathbf{p}_i) = \sum_{j \leq i+1} c_j \mathbf{p}_t^\top A \mathbf{p}_j$$

Hence,

$$(A\mathbf{p}_t)^\top (A\mathbf{p}_i) = 0$$

for all $i < t-1$. Hence, Equation (16.4) simplifies to

$$\mathbf{p}_{t+1} = A\mathbf{p}_t - \frac{\mathbf{p}_t^\top A^2 \mathbf{p}_t}{\mathbf{p}_t^\top A \mathbf{p}_t}\mathbf{p}_t - \frac{\mathbf{p}_t^\top A^2 \mathbf{p}_{t-1}}{\mathbf{p}_{t-1}^\top A \mathbf{p}_{t-1}}\mathbf{p}_{t-1}.$$

Thus, to compute $\mathbf{p}_{t+1}$ we need only $O(1)$ matrix–vector multiplications with A, as promised. This completes the description of the conjugate gradient algorithm which appears formally below.

Algorithm 16.1 CGSolve

Input: Symmetric, positive-definite matrix $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$ and T
Output: $\mathbf{x}_T \in \mathbb{R}^n$

1: $\mathbf{x}_0 \leftarrow \mathbf{0}$
2: $\mathbf{r}_0 \leftarrow \mathbf{b}$
3: $\mathbf{p}_0 = \mathbf{r}_0$
4: **for** $t = 0 \to T-1$ **do**
5: Set $\alpha_t = \frac{\mathbf{p}_t^\top \mathbf{r}_0}{\mathbf{p}_t^\top A \mathbf{p}_t}$
6: Set $\mathbf{r}_t = \mathbf{b} - A\mathbf{x}_t$
7: Set $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t$
8: Set $\mathbf{p}_{t+1} = A\mathbf{p}_t - \frac{\mathbf{p}_t^\top A^2 \mathbf{p}_t}{\mathbf{p}_t^\top A \mathbf{p}_t}\mathbf{p}_t - \frac{\mathbf{p}_t^\top A^2 \mathbf{p}_{t-1}}{\mathbf{p}_{t-1}^\top A \mathbf{p}_{t-1}}\mathbf{p}_{t-1}$.
9: **end for**
10: **return** $\mathbf{x}_T$

Since each step of the conjugate gradient algorithm requires $O(1)$ matrix–vector multiplications, to analyze its running time, it suffices to bound the number of iterations. In the next section, we analyze the convergence time of the conjugate gradient algorithm. In particular, we call a point $\mathbf{x}_t$ an ε-approximate solution if $f(\mathbf{x}_t) - f(\mathbf{x}^\star) \le \varepsilon(f(\mathbf{x}_0) - f(\mathbf{x}^\star))$. We wish to find how many iterations the algorithm needs to get to an ε-approximate solution. We end this section with the following observation: In n steps, the conjugate gradient method returns $\mathbf{x}^\star$ exactly. This is because $\mathbf{x}^\star \in \mathbf{x}_0 + \mathcal{K}_n$.

16.3 Analysis via Polynomial Minimization

We now utilize the fact that $\mathbf{x}_t$ minimizes f over the subspace $\mathbf{x}_0 + \mathcal{K}_t$ to prove an upper bound on $f(\mathbf{x}_t) - f(\mathbf{x}^\star)$. The following is easily seen.

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) = \frac{1}{2}(\mathbf{x}_t - \mathbf{x}^\star)^\top A(\mathbf{x}_t - \mathbf{x}^\star) = \frac{1}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|_A^2.$$

Since $\mathbf{x}_t$ lies in $\mathbf{x}_0 + \mathcal{K}_t$, we can write $\mathbf{x}_t = \mathbf{x}_0 + \sum_{i=0}^{t-1} \gamma_i A^i \mathbf{r}_0$ for some scalars γ_i. Let $p(x)$ be the polynomial defined to be $\sum_{i=0}^{t-1} \gamma_i x^i$. Note that there is a *one-to-one* correspondence between points in $\mathbf{x}_0 + \mathcal{K}_t$ and degree $t-1$ polynomials in one variable. Then, we get $\mathbf{x}_t = \mathbf{x}_0 + p(A)\mathbf{r}_0 = \mathbf{x}_0 + p(A)A(\mathbf{x}^\star - \mathbf{x}_0)$. Therefore, we get

$$\mathbf{x}_t - \mathbf{x}^\star = (I - p(A)A)(\mathbf{x}_0 - \mathbf{x}^\star) = q(A)(\mathbf{x}_0 - \mathbf{x}^\star),$$

where $q(x) \stackrel{\text{def}}{=} 1 - xp(x)$. Now, note that there is a *one-to-one* correspondence between degree $t-1$ polynomials and degree t polynomials that evaluate to 1 at 0. Let this latter set of polynomials be $\mathcal{Q}_t$. Since $\mathbf{x}_t$ minimizes $\|\mathbf{x}_t - \mathbf{x}^\star\|_A^2$ over $\mathcal{K}_t$, we get that $f(\mathbf{x}_t) - f(\mathbf{x}^\star)$ equals

$$\frac{1}{2}\|\mathbf{x}_t - \mathbf{x}^\star\|_A^2 = \min_{q \in \mathcal{Q}_t} \left(q(A)(\mathbf{x}_0 - \mathbf{x}^\star)\right)^\top A(q(A)(\mathbf{x}_0 - \mathbf{x}^\star)). \tag{16.5}$$

Before we proceed with the proof of Theorem 16.1, we need the following lemma.

Lemma 16.2. Let A be a symmetric matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$. Then, for any polynomial $p(\cdot)$ and vector $\mathbf{v}$,

$$(p(A)\mathbf{v})^\top A(p(A)\mathbf{v}) \leq \mathbf{v}^\top A\mathbf{v} \cdot \max_{i=1}^{n} |p(\lambda_i)|^2.$$

Proof. Write $A = U\Gamma U^\top$ where Γ is the diagonal matrix consisting of eigenvalues and the columns of U are the corresponding orthonormal eigenvectors. Note that $p(A) = Up(\Gamma)U^\top$. Now for any vector $\mathbf{v}$, we get

$$\mathbf{v}^\top p(A)^\top A p(A)\mathbf{v} = \mathbf{v}^\top U p(\Gamma) U^\top U \Gamma U^\top U p(\Gamma) U^\top \mathbf{v} = \mathbf{v}^\top U \Gamma p^2(\Gamma) U^\top \mathbf{v}.$$

Thus, if $\mathbf{v} = \sum_j \zeta_j \mathbf{u}_j$ where $\mathbf{u}_j$ is the eigenvector of A corresponding to λ_j, we get that the l.h.s. of the above equality is $\sum_j \zeta_j^2 \lambda_j p^2(\lambda_j)$. Similarly, $\mathbf{v}^\top A\mathbf{v} = \sum_j \zeta_j^2 \lambda_j$. The lemma follows. □

Plugging the above claim into Equation (16.5) we get the following upper bound on $f(\mathbf{x}_t)$ which is crux in the analysis of the conjugate gradient algorithm:

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \min_{q \in \mathcal{Q}_t} \max_{i=1}^{n} |q(\lambda_i)|^2 \cdot (f(\mathbf{x}_0) - f(\mathbf{x}^\star)). \tag{16.6}$$

Note that, since the eigenvalues of A satisfy $\lambda_1 \leq \cdots \leq \lambda_n$, the r.h.s. of Equation (16.6) can be further approximated by

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \min_{q \in \mathcal{Q}_t} \max_{x \in [\lambda_1, \lambda_n]} |q(x)|^2 \cdot (f(\mathbf{x}_0) - f(\mathbf{x}^\star)). \tag{16.7}$$

Thus, since $f(\mathbf{x}_0) = f(\mathbf{0}) = 0$ and $f(\mathbf{x}^\star) = -1/2 \cdot \|\mathbf{x}^\star\|_A^2$, we have proved the following lemma about CGSolve.

Lemma 16.3. Let $A \succ 0$, λ_1, λ_n be the smallest and the largest eigenvalues of A, respectively, and let $\mathcal{Q}_t$ be the set of polynomials of degree at most t which take the value 1 at 0. Then,

$$\|\mathbf{x}_t - \mathbf{x}^\star\|_A^2 \leq \|\mathbf{x}^\star\|_A^2 \cdot \min_{q \in \mathcal{Q}_t} \max_{x \in [\lambda_1, \lambda_n]} |q(x)|^2.$$

Therefore, *any* polynomial in $\mathcal{Q}_t$ gives an upper bound on $f(\mathbf{x}_t)$. In particular, the polynomial $q(x) \stackrel{\text{def}}{=} (1 - \frac{2x}{\lambda_1 + \lambda_n})^t$ gives

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^t (f(\mathbf{x}_0) - f(\mathbf{x}^\star)),$$

where $\kappa = \lambda_n/\lambda_1$. This is slightly better than what we obtained in Lemma 15.2. As a corollary of the discussion above, we again obtain that after n iterations the solution computed is exact.

Corollary 16.4. After n steps of CGSolve, $\mathbf{x}_n = \mathbf{x}^\star$. Hence, in $O(t_A n)$ time, one can compute $\mathbf{x}^\star = A^+\mathbf{b}$ for a positive definite A.

Proof. Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of A. Consider the polynomial $q(x) \stackrel{\text{def}}{=} \prod_{i=1}^n (1 - x/\lambda_i)$. Note that $q(0) = 1$, and $q(\lambda_i) = 0$ for all eigenvalues of A. From Lemma 16.3,

$$\|\mathbf{x}_n - \mathbf{x}^\star\|_A^2 \leq \|\mathbf{x}^\star\|_A^2 \cdot \max_{i=1}^n |q(\lambda_i)|^2 = 0. \qquad \square$$

16.4 Chebyshev Polynomials — Why Conjugate Gradient Works

In the previous section we reduced the problem of bounding the error after t iterations to a problem about polynomials. In particular, the goal reduced to finding a polynomial $q(\cdot)$ of degree at most t which takes the value 1 at 0 and minimizes $\max_{i=1}^{n} |q(\lambda_i)|$ where λ_is are eigenvalues of A. In this section we present a family of polynomials, called Chebyshev polynomials, and use them to prove Theorem 16.1.

For a nonnegative integer t, we will let $T_t(x)$ denote the degree t Chebyshev polynomial of the first kind, which are defined recursively as follows: $T_0(x) \stackrel{\text{def}}{=} 1, T_1(x) \stackrel{\text{def}}{=} x$, and for $t \geq 2$,

$$T_t(x) \stackrel{\text{def}}{=} 2xT_{t-1}(x) - T_{t-2}(x).$$

The following is a simple consequence of the recursive definition above.

Proposition 16.5. If $x \in [-1, 1]$, then we can define $\theta \stackrel{\text{def}}{=} \arccos(x)$, and the degree t Chebyshev polynomial is given by $T_t(\cos\theta) = \cos(t\theta)$. Hence, $T_t(x) \in [-1, 1]$ for all $x \in [-1, 1]$.

Proof. First, note that $T_0(\cos\theta) = \cos 0 = 1$ and $T_1(\cos\theta) = \cos\theta = x$. Additionally, $\cos((t+1)\theta) = \cos(t\theta)\cos(\theta) - \sin(t\theta)\sin(\theta)$ and similarly $\cos((t-1)\theta) = \cos(t\theta)\cos(\theta) - \sin(t\theta)\sin(\theta)$. Hence, it is clear that the recursive definition for Chebyshev polynomials, namely $\cos((t+1)\theta) = 2\cos\theta\cos(t\theta) + \cos((t-1)\theta)$, applies. □

Thus, letting $x = \cos\theta$ and using de Moivre's formula, $T_t(x) = \cos(t\theta) = \frac{1}{2}(\exp(it\theta) + \exp(-it\theta))$ can be written as

$$T_t(x) = \frac{1}{2}((x + \sqrt{x^2 - 1})^t + (x - \sqrt{x^2 - 1})^t),$$

which is a polynomial of degree t. For $0 < a < b$, we define the polynomial $Q_{a,b,t}$ as follows:

$$Q_{a,b,t}(x) \stackrel{\text{def}}{=} \frac{T_t\left(\frac{a+b-2x}{b-a}\right)}{T_t\left(\frac{a+b}{b-a}\right)}.$$

Note that $Q_{a,b,t}$ is of degree t and evaluates to 1 at $x = 0$ and hence lies in $\mathcal{Q}_t$. Furthermore, for $x \in [a, b]$, the numerator takes a value of at most 1 (by the definition of T_t). Therefore, if we plug in $a = \lambda_1$ and $b = \lambda_n$, the smallest and the largest eigenvalues of A, respectively, we obtain

$$\forall x \in [\lambda_1, \lambda_n], \quad Q_{\lambda_1,\lambda_n,t}(x) \leq T_t\left(\frac{\lambda_1 + \lambda_n}{\lambda_n - \lambda_1}\right)^{-1} = T_t\left(\frac{\lambda_n/\lambda_1 + 1}{\lambda_n/\lambda_1 - 1}\right)^{-1}.$$

Since $\kappa(A) = \lambda_n/\lambda_1$, and $T_t(\frac{\kappa+1}{\kappa-1}) = \frac{1}{2}((\frac{\sqrt{k}+1}{\sqrt{k}-1})^t + (\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^t)$, we get

$$\forall x \in [\lambda_1, \lambda_n], \quad Q_{\lambda_1,\lambda_n,t}(x) \leq 2\left(\frac{\sqrt{\lambda_n/\lambda_1} - 1}{\sqrt{\lambda_n/\lambda_1} + 1}\right)^t. \tag{16.8}$$

Thus, by Lemma 16.3, for any $t \geq \Omega\ (\sqrt{\kappa(A)}\log 1/\varepsilon)$, after t steps of CGSolve we get $\mathbf{x}_t$ satisfying $f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \varepsilon^2 \cdot (f(\mathbf{x}_0) - f(\mathbf{x}^\star))$. Thus, for this value of t,

$$\|\mathbf{x}_t - A^+\mathbf{b}\|_A \leq \varepsilon \|A^+\mathbf{b}\|_A.$$

This completes the proof of Theorem 16.1.

16.5 The Chebyshev Iteration

In this section we consider the possibility of attaining a bound of $\sqrt{\kappa(A)}\log 1/\varepsilon$ iterations when $\mathbf{x}_t = p_t(A)\mathbf{b}$ where p_t is a sequence of polynomials. When compared with CGSolve, this will have the advantage that $\mathbf{x}_t$ is a linear operator applied to $\mathbf{b}$. As in the proof of Lemma 16.3, it is sufficient to define p_ts such that

$$\max_{x \in [\lambda_1, \lambda_n]} |xp_t(x) - 1| \leq O(1 - \sqrt{\lambda_1/\lambda_n})^t, \tag{16.9}$$

which gives us

$$\|\mathbf{x}_t - A^+\mathbf{b}\|_A^2 \leq O(1 - \sqrt{\lambda_1/\lambda_n})^t \cdot \|A^+\mathbf{b}\|_A^2.$$

Thus, in $O(\sqrt{\kappa(A)}\log 1/\varepsilon)$ iterations $\mathbf{x}_t$ is ε close to $A^+\mathbf{b}$. Note that Equation (16.9) implies that, if we let $Z_t \stackrel{\text{def}}{=} p_t(A)$, then for all $\mathbf{b}$,

$$\|Z_t\mathbf{b} - A^+\mathbf{b}\|_A \leq O(1 - \sqrt{\lambda_1/\lambda_n})^t.$$

Hence, if $\|Z_t\mathbf{b} - A^+\mathbf{b}\|_A \leq \delta$, then $\|Z_t - A\| \leq \delta \cdot \|A^+\|^{1/2}$. In fact, we can set p_t to be $Q_{\lambda_1,\lambda_n,t}$, defined using the Chebyshev polynomials in the previous section, and use Equation (16.8) to obtain the bound in Equation (16.9). The only issue that remains is how to compute $Q_{\lambda_1,\lambda_n,t}(A)\mathbf{b}$ in $\tilde{O}(t_A t)$-time. This can be done using the recursive definition of the Chebyshev polynomials in the previous section and is left as an exercise. The update rule starts by setting

$$\mathbf{x}_0 \stackrel{\text{def}}{=} \mathbf{0} \quad \text{and} \quad \mathbf{x}_1 \stackrel{\text{def}}{=} \mathbf{b},$$

and for $t \geq 2$,

$$\mathbf{x}_t \stackrel{\text{def}}{=} \alpha_2 A\mathbf{x}_{t-1} + \alpha_1 \mathbf{x}_{t-2} + \alpha_0 \mathbf{b}$$

for fixed scalars $\alpha_0, \alpha_1, \alpha_2$ which depend on λ_1 and λ_n. This is called the Chebyshev iteration and, unlike CGSolve, requires the knowledge of the smallest and the largest eigenvalues of A.[1] We summarize the discussion in this section in the following theorem.

Theorem 16.6. There is an algorithm which given an $n \times n$ symmetric positive-definite matrix A, a vector $\mathbf{b}$, numbers $\lambda_l \leq \lambda_1(A)$ and $\lambda_u \geq \lambda_n(A)$ and an error parameter $\varepsilon > 0$, returns an $\mathbf{x}$ such that

(1) $\|\mathbf{x} - A^+\mathbf{b}\|_A \leq \varepsilon \cdot \|A^+\mathbf{b}\|_A$,
(2) $\mathbf{x} = Z\mathbf{b}$ where Z depends only on A and ε, and
(3) $\|Z - A^+\| \leq \varepsilon$,

where we have absorbed the $\|A^+\|^{1/2}$ term in the error. The algorithm runs in $(t_A \cdot \sqrt{\lambda_u/\lambda_l} \log 1/(\varepsilon\lambda_l))$ time.

16.6 Matrices with Clustered Eigenvalues

As another corollary of the characterization of Lemma 16.3 we show that the rate of convergence of CGSlove is better if the eigenvalues of A are *clustered*. This partly explains why CGSolve is attractive in practice; it is used in Section 17 to construct fast Laplacian solvers.

[1] For our application in Section 18, we will be able to provide estimates of these eigenvalues.

Corollary 16.7. For a matrix A, suppose all but c eigenvalues are contained in a range $[a, b]$. Then, after $t \geq c + O(\sqrt{b/a} \log 1/\varepsilon)$ iterations,

$$\|\mathbf{x}_t - \mathbf{x}^\star\|_A \leq \varepsilon \|\mathbf{x}^\star\|_A.$$

Proof. Let $q(x) \stackrel{\text{def}}{=} Q_{a,b,i}(x) \cdot \Pi_{i=1}^{c}(1 - x/\lambda_i)$ where $\lambda_1, \ldots, \lambda_c$ are the c eigenvalues outside of the interval $[a, b]$, and $Q_{a,b,i}$ defined earlier. From Lemma 16.3, since the value of $q(\lambda_i) = 0$ for all $i = 1, \ldots, c$, we only need to consider the maximum value of $|q(x)|^2$ in the range $[a, b]$. By the properties of $Q_{a,b,i}$ described above, if we pick $i = \Theta(\sqrt{b/a} \log 1/\varepsilon)$, for all $x \in [a, b]$, $|q(x)|^2 \leq \varepsilon$. Therefore, CGSolve returns an ε-approximation in $c + O(\sqrt{b/a} \log 1/\varepsilon)$ steps; note that this could be much better than $\sqrt{\kappa(A)}$ since no restriction is put on the c eigenvalues. □

Notes

The conjugate gradient method was first proposed in [39]. There is a vast amount of literature on this algorithm and the reader is directed to the work in [36, 37, 68, 72]. More on Chebyshev polynomials and their centrality in approximation theory can be found in the books [21, 65]. An interested reader can check that a Chebyshev-like iterative method with similar convergence guarantees can also be derived by applying Nesterov's accelerated gradient descent method to the convex function in the beginning of this section.

17

Preconditioning for Laplacian Systems

In this section, the notion of preconditioning is introduced. Instead of solving $A\mathbf{x} = \mathbf{b}$, here one tries to solve $PA\mathbf{x} = P\mathbf{b}$ for a matrix P such that $\kappa(PA) \ll \kappa(A)$ where it is not much slower to compute $P\mathbf{v}$ than $A\mathbf{v}$, thus speeding up iterative methods such as the conjugate gradient method. As an application, preconditioners are constructed for Laplacian systems from low-stretch spanning trees that result in a $\tilde{O}(m^{4/3})$ time Laplacian solver. Preconditioning plays an important role in the proof of Theorem 3.1 presented in Section 18. Low-stretch spanning trees have also been used in Section 4.

17.1 Preconditioning

We saw in the previous section that using the conjugate gradient method for a symmetric matrix $A \succ 0$, one can solve a system of equations $A\mathbf{x} = \mathbf{b}$ in time $O(t_A \cdot \sqrt{\kappa(A)} \log 1/\varepsilon)$ up to an error of ε. Let us focus on the two quantities in this running time bound: t_A which is the time it takes to multiply a vector with A, and $\kappa(A)$, the condition number of A. Note that the algorithm requires only restricted access to A: Given a vector $\mathbf{v}$, output $A\mathbf{v}$. Thus, one strategy to reduce this running

time is to find a matrix P s.t. $\kappa(PA) \ll \kappa(A)$ and $t_P \sim t_A$. Indeed, if we had access to such a matrix, we could solve the equivalent system of equations $PA\mathbf{x} = P\mathbf{b}$.[1] Such a matrix P is called a *preconditioner* for A. One choice for P is A^+. This reduces the condition number to 1 but reduces the problem of computing $A^+\mathbf{b}$ to itself, rendering it useless. Surprisingly, as we will see in this section, for a Laplacian system one can often find preconditioners by using the graph structure, thereby reducing the running time significantly. The following theorem is the main result of this section.

Theorem 17.1. For any undirected, unweighted graph G with m edges, a vector $\mathbf{b}$ with $\langle \mathbf{b}, \mathbf{1} \rangle = 0$, and $\varepsilon > 0$, one can find an $\mathbf{x}$ such that $\|\mathbf{x} - L_G^+\mathbf{b}\|_{L_G} \leq \varepsilon \|L_G^+\mathbf{b}\|_{L_G}$ in $\tilde{O}(m^{4/3} \log 1/\varepsilon)$ time.

There are two crucial ingredients to the proof. The first is the following simple property of CGSolve.

Lemma 17.2. Suppose A is a symmetric positive-definite matrix with minimum eigenvalue $\lambda_1 \geq 1$ and trace $\mathrm{Tr}(A) \leq \tau$. Then CGSolve converges to an ε-approximation in $O(\tau^{1/3} \log 1/\varepsilon)$ iterations.

Proof. Let Λ be the set of eigenvalues larger than τ/γ, where γ is a parameter to be set later. Note that $|\Lambda| \leq \gamma$. Therefore, apart from these γ eigenvalues, all the rest lie in the range $[1, \tau/\gamma]$. From Corollary 16.7, we get that CGSolve finds an ε-approximation in $\gamma + O(\sqrt{\tau/\gamma} \log 1/\varepsilon)$ iterations. Choosing $\gamma \stackrel{\text{def}}{=} \tau^{1/3}$ completes the proof of the lemma. □

The second ingredient, and the focus of this section, is a construction of a combinatorial preconditioner for L_G. The choice of preconditioner is L_T^+ where T is a spanning tree of G. Note that from Corollary 13.4, this preconditioner satisfies the property that $L_T^+\mathbf{v}$ can be computed in $O(n)$ time. Thus, the thing we need to worry about is how to construct a spanning tree T of G such that $\kappa(L_T^+ L_G)$ is as small as possible.

---

[1] One might worry that the matrix PA may not be symmetric; one normally gets around this by preconditioning by $P^{1/2}AP^{1/2}$. This requires $P \succ 0$.

17.2 Combinatorial Preconditioning via Trees

Let G be an unweighted graph with an arbitrary orientation fixed for the edges giving rise to the vectors $\mathbf{b}_e$ which are the rows of the corresponding incidence matrix. We start by trying to understand why, for a spanning tree T of a graph G, L_T^+ might be a natural candidate for preconditioning L_G. Note that

$$L_T = \sum_{e \in T} \mathbf{b}_e \mathbf{b}_e^\top \preceq \sum_{e \in G} \mathbf{b}_e \mathbf{b}_e^\top = L_G.$$

Thus, $I \prec L_T^+ L_G$, which implies that $\lambda_1(L_T^+ L_G) \geq 1$. Thus, to bound the condition number of $\kappa(L_T^+ L_G)$, it suffices to bound $\lambda_n(L_T^+ L_G)$. Unfortunately, there is no easy way to bound this. Since $L_T^+ L_G$ is PSD, an upper bound on its largest eigenvalue is its trace, $\mathrm{Tr}(L_T^+ L_G)$. If this upper bound is τ, Lemma 17.2 would imply that the conjugate gradient method applied to $L_T^+ L_G$ takes time approximately $\tau^{1/3} t_{L_T^+ L_G} \sim O(\tau^{1/3}(m+n))$. Thus, even though it sounds wasteful to bound the trace by τ rather than by the largest eigenvalue by τ, Lemma 17.2 allows us to improve the dependency on τ from $\tau^{1/2}$ to $\tau^{1/3}$. We proceed to bound the trace of $L_T^+ L_G$:

$$\mathrm{Tr}(L_T^+ L_G) = \mathrm{Tr}\left(L_T^+ \sum_{e \in G} \mathbf{b}_e \mathbf{b}_e^\top \right) = \sum_e \mathbf{b}_e^\top L_T^+ \mathbf{b}_e,$$

where we use $\mathrm{Tr}(A+B) = \mathrm{Tr}(A) + \mathrm{Tr}(B)$ and $\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB)$. Note that $\mathbf{b}_e^\top L_T^+ \mathbf{b}_e$ is a scalar and is precisely the effective resistance across the endpoints of e in the tree T where each edge of G has a unit resistance. The effective resistance across two nodes i, j in a tree is the sum of effective resistances along the unique path $P(i,j)$ on the tree. Thus, we get

$$\mathrm{Tr}(L_T^+ L_G) = \sum_{e \in G} |P(e)|.$$

A trivial upper bound on $\mathrm{Tr}(L_T^+ L_G)$, thus, is nm. This can also be shown to hold when G is weighted. This leads us to the following definition.

Definition 17.1. For an unweighted graph G, the *stretch* of a spanning T is defined to be $\mathrm{str}_T(G) \stackrel{\text{def}}{=} \mathrm{Tr}(L_T^+ L_G)$.

Thus, to obtain the best possible bound on the number of iterations of CGSolve, we would like a spanning tree T of G which minimizes the average length of the path an edge of G has to travel in T. The time it takes to construct T is also important.

17.3 An $\tilde{O}(m^{4/3})$-Time Laplacian Solver

In this section we complete the proof of Theorem 17.1. We start by stating the following nontrivial structural result about the existence and construction of low-stretch spanning trees whose proof is graph-theoretic and outside the scope of this monograph. The theorem applies to weighted graphs as well.

Theorem 17.3. For any undirected graph G, a spanning tree T can be constructed in $\tilde{O}(m \log n + n \log n \log\log n)$ time such that $\mathrm{str}_T(G) = \tilde{O}(m \log n)$. Here $\tilde{O}(\cdot)$ hides $\log\log n$ factors.

This immediately allows us to conclude the proof of Theorem 17.1 using Lemma 17.2 and the discussion in the previous section. The only thing that remains is to address the issue that the matrix $L_T^+ L_G$ is symmetric. The following trick is used to circumvent this difficulty. Recall from Theorem 13.3 that the Cholesky decomposition of a tree can be done in $O(n)$ time. In particular, let $L_T = EE^\top$, where E is a lower triangular matrix with at most $O(n)$ nonzero entries. The idea is to look at the system of equations $E^+ L_G {E^+}^\top \mathbf{y} = E^+ \mathbf{b}$ instead of $L_G \mathbf{x} = \mathbf{b}$. If we can solve for $\mathbf{y}$ then we can find $\mathbf{x} = {E^+}^\top \mathbf{y}$, which is computationally fast since E is lower triangular with at most $O(n)$ nonzero entries. Also, for the same reason, $E^+ \mathbf{b}$ can be computed quickly.

Now we are in good shape. Let $A \stackrel{\text{def}}{=} E^+ L_G {E^+}^\top$ and $\mathbf{b}' \stackrel{\text{def}}{=} E^+ \mathbf{b}$; note that A is symmetric. Also, the eigenvalues of A are the same as those of ${E^\top}^+ A E^\top = L_T^+ L_G$. Thus, the minimum eigenvalue of A is 1

and the trace is $\tilde{O}(m)$ by Theorem 17.3. Thus, using the conjugate gradient method, we find an ε-approximate solution to $A\mathbf{y} = \mathbf{b}'$ in $\tilde{O}(m^{1/3})$ iterations.

In each iteration, we do $O(1)$ matrix–vector multiplications. Note that for any vector $\mathbf{v}$, $A\mathbf{v}$ can be computed in $O(n + m)$ operations since $E^+\mathbf{v}$ takes $O(n)$ operations (since E is a lower triangular matrix with at most $O(n)$ nonzero entries) and $L_G\mathbf{v}'$ takes $O(m)$ operations (since L_G has at most $O(m)$ nonzero entries).

Notes

While preconditioning is a general technique for Laplacian systems, its use originates in the work of Vaidya [86]. Using preconditioners, Vaidya obtained an $\tilde{O}((\Delta n)^{1.75} \log 1/\varepsilon)$ time Laplacian solver for graphs with maximum degree Δ. Vaidya's paper was never published and his ideas and their derivatives appear in the thesis [42]. Boman and Hendrickson [18] use low-stretch spanning trees constructed by Alon et al. [6] to obtain ε-approximate solutions to Laplacian systems roughly in $m^{3/2} \log 1/\varepsilon$ time. The main result of this section, Theorem 17.1, is from [81]. The first $\operatorname{polylog} n$ stretch trees in near-linear-time were constructed by Elkin et al. [27]. Theorem 17.3 is originally from [1], and improvements can be found in [2]. An important open problem is to find a simple proof of Theorem 17.3 (potentially with worse log factors).

18

Solving a Laplacian System in $\tilde{O}(m)$ Time

Building on the techniques developed hitherto, this section presents an algorithm which solves $L\mathbf{x} = \mathbf{b}$ in $\tilde{O}(m)$ time.

18.1 Main Result and Overview

In Section 17 we saw how preconditioning a symmetric PSD matrix A by another symmetric PSD matrix P reduces the running time of computing $A^{+}\mathbf{b}$ for a vector $\mathbf{b}$ to about $O((t_A + t_P)\sqrt{\kappa(PA)})$, where t_A and t_P are the times required to compute matrix–vector product with A and P, respectively. We saw that if $A = L_G$ for a graph G, then a natural choice for P is L_T^{+} where T is a spanning tree of G with small total stretch.[1] For a graph G with edge weights given by w_G and a tree T, the stretch of an edge $e \in E(G)$ in T is defined to be

$$\mathrm{str}_T(e) \stackrel{\text{def}}{=} w_G(e) \sum_{f \in P_e} w_G^{-1}(f),$$

where P_e is the set of edges on the unique path that connects the endpoints of e in T. Thus, if $e \in T$, then $\mathrm{str}_T(e) = 1$. We define

[1] Unless specified, when we talk about a spanning tree T of a weighted graph, the edges of T inherit the weights from G.

$\mathrm{str}_T(G) \stackrel{\text{def}}{=} \sum_{e \in E} \mathrm{str}_T(e)$. With this definition, it is easy to see that

$$\mathrm{str}_T(G) = \mathrm{Tr}(L_T^+ L_G).$$

This is the same as the definition in Section 17 where Theorem 17.3 asserted a remarkable result: A spanning tree T such that $\mathrm{str}_T(G) = \tilde{O}(m)$ can be constructed in time $\tilde{O}(m)$. Note that choosing a spanning tree is convenient since we can compute $L_T^+ \mathbf{v}$ exactly in $O(n)$ time using Cholesky decomposition, see Theorem 13.3. If T is such a spanning tree and L_T is its Laplacian, then, using L_T^+ as a preconditioner, we showed in Theorem 17.1 how the conjugate gradient method can be used to compute a vector $\mathbf{x}$ such that $\|\mathbf{x} - L_G^+ \mathbf{b}\|_{L_G} \leq \varepsilon \|L_G^+ \mathbf{b}\|_{L_G}$ in $\tilde{O}(m^{4/3} \log 1/\varepsilon)$ time. In this section we improve this result and prove the following theorem; the key result of this monograph.

Theorem 18.1. For any undirected, unweighted graph G with m edges, a vector $\mathbf{b}$ with $\langle \mathbf{b}, \mathbf{1} \rangle = 0$, and $\varepsilon > 0$, one can find an $\mathbf{x}$ such that $\|\mathbf{x} - L_G^+ \mathbf{b}\|_{L_G} \leq \varepsilon \|L_G^+ \mathbf{b}\|_{L_G}$ in $\tilde{O}(m \log 1/\varepsilon)$ time.

This theorem is a restatement of Theorem 3.1 where the algorithm which achieves this bound is referred to as LSolve. In this section we present LSolve and show that it runs in $\tilde{O}(m \log 1/\varepsilon)$ time. Toward the end we discuss the linearity of LSolve: Namely, the output $\mathbf{x}$ of LSolve on input $L_G, \mathbf{b}$ and ε is a vector $Z\mathbf{x}$, where Z is a $n \times n$ matrix that depends only on G and ε, and satisfies $\|Z - L^+\| \leq \varepsilon$.

Proof Overview

Unlike the proof of Theorem 17.1, it is not clear how to prove Theorem 18.1 by restricting to tree preconditioners. The reason is that, while for a tree T of G we can compute $L_T^+ \mathbf{v}$ in $O(n)$ time, we do not know how to improve upon the upper bound on $\kappa(L_T^+ L_G)$ beyond what was presented in Section 17. A natural question is whether we can reduce the condition number by allowing more edges than those contained in a tree. After all, the Cholesky decomposition-based algorithm to solve tree systems in Section 13 works if, at every step during the elimination process, there is always a degree 1 *or* a degree 2 vertex.

The first observation we make, proved in Section 18.2, is that if we have a graph on n vertices and $n-1+k$ edges (i.e., k more edges than in a tree), after we are done eliminating all degree 1 and 2 vertices, we are left with a graph that has at most $2(k-1)$ vertices and $3(k-1)$ edges. Thus, if k is not too large, there is a serious reduction in the size of the linear system that is left to solve. Now, can we find a subgraph H of G (whose edges are allowed to be weighted) that has at most $n-1+k$ edges and the condition number of $L_H^+ L_G$ is much smaller than n? The answer turns out to be yes. We achieve this by a combination of the spectral sparsification technique from Section 10 with low-stretch spanning trees. Specifically, for a graph G with n vertices and m edges, in $\tilde{O}(m)$ time we can construct a graph H with $n-1+k$ edges such that $\kappa(L_H^+ L_G) \leq O(^{(m\log^2 n)}/_k)$. Thus, if we choose $k = {}^m/_{(100\log^2 n)}$, apply crude sparsification followed by eliminating all degree 1 and 2 vertices, we are left with a Laplacian system of size $O(^m/_{(100\log^2 n)})$, call it $\tilde{G}$. The details of how to find such an H appear in Section 18.3.

We now need to make about $\sqrt{\kappa(L_H^+ L_G)} \leq 10\log^2 n$ calls to the conjugate gradient method to solve for $L_G^+ \mathbf{b}$. This immediately leads us to the problem of computing approximately $10\log^2 n$ products of the form $L_H^+ \mathbf{v}$ for some vector $\mathbf{v}$. This, in turn, requires us to compute the same number of $L_{\tilde{G}}^+ \mathbf{u}$ products. $\tilde{G}$ is neither a tree nor does it contain any degree 1 or 2 vertices. Thus, to proceed, we recurse on $\tilde{G}$. Fortunately, the choice of parameters ensures that $\sqrt{\kappa(L_H^+ L_G)}/_{(100\log^2 n)} \leq {}^1/_{10}$. This *shrinkage* ensures that the work done at any level remains bounded by $\tilde{O}(m)$. We stop the recursion when the size of the instance becomes $\operatorname{polylog} n$. This means that the depth of the recursion is bounded by $O(^{\log n}/_{\log\log n})$. The details of how the recursion is employed and how the parameters are chosen to ensure that the total work remains $\tilde{O}(m)$ are presented in Section 18.4.

There is an important issue regarding the fact that the conjugate gradient is error-prone. While in practice this may be fine, in theory, this can cause serious problems. To control the error, one has to replace the conjugate gradient with its linear version, Chebyshev iteration, presented in Section 16.5. This results in an algorithm that makes

LSolve a linear operator as claimed above. The details are presented in Section 18.5 and can be omitted.

If we were to present all the details that go into the proof of Theorem 18.1, including a precise description of LSolve, it would be overwhelming to the reader and make the presentation quite long. Instead, we bring out the salient features and important ideas in the algorithm and proof, and show how it marries ideas between graph theory and linear algebra that we have already seen in this monograph. The goal is not to convince the reader about the proof of Theorem 18.1 to the last constants, but to give enough detail to allow a keen reader to reconstruct the full argument and adapt it to their application. Finally, in what follows, we will ignore $\text{poly}(\log\log n)$ factors.

18.2 Eliminating Degree 1, 2 Vertices

Suppose H is a graph with n' vertices and $m' = n' - 1 + k$ edges. As in Section 13.2, we greedily eliminate all degree 1 and 2 vertices to obtain a graph G'. If $m' \leq n' - 1$, then we can compute $L_H^+ \mathbf{v}$ in $O(n')$ time via Theorem 13.3. Hence, we may assume that $k \geq 1$. This reduces the computation of $L_H^+ \mathbf{v}$ to that of computing $L_{G'}^+ \mathbf{v}$ in $O(m' + n')$ time.

How many edges and vertices does G' have? Let k_1 and k_2 be the number of vertices of degree 1 and 2 eliminated from H, respectively. Then $|V(G')| = n' - k_1 - k_2$ and $|E(G')| \leq m' - k_1 - k_2$. The latter occurs because we removed two edges adjacent to a degree 2 vertex and added one edge between its neighbors. In G', however, every vertex has degree at least 3. Hence,

$$3(n' - k_1 - k_2) \leq 2(m' - k_1 - k_2) = 2(n' - 1 + k - k_1 - k_2).$$

Hence, $|V(G')| = n' - k_1 - k_2 \leq 2(k-1)$, and consequently,

$$|E(G')| \leq m' - k_1 - k_2 \leq 2(k-1) + k - 1 = 3(k-1).$$

Thus, we have proved the following lemma.

Lemma 18.2. Given a graph H with n' vertices and $m' = n' - 1 + k$ edges for $k \geq 1$, the computation of $L_H^+ \mathbf{v}$ can be reduced to computing $L_{G'}^+ \mathbf{v}$ in time $O(n' + m')$ where G' has at most $2(k-1)$ vertices and at most $3(k-1)$ edges.

18.3 Crude Sparsification Using Low-Stretch Spanning Trees

We now present the following crude spectral sparsification theorem, which is a rephrasing of Theorem 10.4.

Theorem 18.3. There is an algorithm that, given a graph G with edge weights w_G, $\gamma > 0$, and numbers q_e such that $q_e \geq w_G(e)R_e$ for all e, outputs a weighted graph H s.t.

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

and $O(W \log W \log 1/\gamma)$ edges. The algorithm succeeds with probability at least $1 - \gamma$ and runs in $\tilde{O}(W \log W \log 1/\gamma)$ time. Here $W \stackrel{\text{def}}{=} \sum_e q_e$ and R_e is the effective resistance of e.

We show how one can compute q_es that meet the condition of this theorem and have small $\sum_e q_e$ in time $O(m \log n + n \log^2 n)$. Thus, we do *not* rely on the Laplacian solver as in the proof of Theorem 10.2. First notice that if T is a spanning tree of G with weight function w_G and $e \in G$, then $R_e \leq \sum_{f \in P_e} w_G^{-1}(f)$, where P_e is the path in T with endpoints the same as e. This is because adding more edges only reduces the effective resistance between the endpoints of e. Hence, $w_e R_e \leq \mathrm{str}_T(e)$. Thus, $\mathrm{str}_T(e)$ satisfies one property required by q_e in the theorem above. Moreover, we leave it as an exercise to show that, given T, one can compute $\mathrm{str}_T(e)$ for all $e \in G$ in $O(m \log n)$ time using elementary methods. The thing we need to worry about is how to quickly compute a T with small $\sum_e \mathrm{str}_T(e)$. This is exactly where the low-stretch spanning trees from Theorem 17.3, and mentioned in the introduction to this section, come into play. Recall that, for a graph G with weight function w_G, the tree T from Theorem 17.3 satisfies $\sum_{e \in G} \mathrm{str}_T(e) = O(m \log n)$ and can be computed in time $O(n \log^2 n + m \log n)$. Hence, the number of edges in the crude sparsifier from Theorem 18.3 is $O(m \log^2 n)$ if we set $\gamma = 1/\log n$. This is not good as we have *increased* the number of edges in the sampled graph.

There is a trick to get around this. Let $1 \ll \kappa < m$ be an additional parameter and let T be the low-stretch spanning tree from Theorem 17.3 for the graph G. Consider the graph $\hat{G} = (V, E)$ with

weight function $w_{\hat{G}}(e) \stackrel{\text{def}}{=} \kappa w_G(e)$ if $e \in T$ and $w_{\hat{G}}(e) \stackrel{\text{def}}{=} w_G(e)$ if $e \notin T$. Thus, in $\hat{G}$ we have scaled the edges of T by a factor of κ. This trivially implies that

$$L_G \preceq L_{\hat{G}} \preceq \kappa L_G.$$

We now let $q_e \stackrel{\text{def}}{=} \text{str}_T(e) = 1$ if $e \in T$ and $q_e \stackrel{\text{def}}{=} 1/\kappa \cdot \text{str}_T(e)$ if $e \notin T$. Thus, again it can be checked that the q_es satisfy the conditions for Theorem 18.3 for $\hat{G}$ with weights $w_{\hat{G}}$. First note that

$$W = \sum_e q_e = \sum_{e \in T} 1 + \sum_{e \notin T} \frac{\text{str}_T(e)}{\kappa} = n - 1 + O\left(m \log n / \kappa\right)$$

as $\text{str}_T(G) = O(m \log n)$. Now, let us estimate the number of edges we obtain when we sample from the distribution $\{q_e\}_{e \in E}$ approximately $W \log W$ times as in Theorem 10.4. Note that even if an edge is chosen multiple times in the sampling process, there is exactly one edge in H corresponding to it with a weight summed up over multiple selections. Thus, in H, we account for edges from T separately. These are exactly $n - 1$. On the other hand, if an edge $e \notin T$, a simple Chernoff bound argument implies that with probability at least $1 - 1/n^{0.1} \gg 1 - 1/\log n$, the number of such edges chosen is at most $O(m \log^2 n / \kappa)$. Since it takes $O(n \log^2 n + m \log n)$ time to find T and approximately $W \log W$ additional time to sample H, in

$$O(n \log^2 n + m \log n + m \log^2 n / \kappa)$$

time, with probability at least $1 - 1/\log^2 n$, we obtain a graph H such that

$$L_G \preceq 2L_H \preceq 3\kappa L_G.$$

Moreover, the number of edges in H is $n - 1 + O(m \log^2 n / \kappa)$. Importantly, $\kappa(L_H^+ L_G) = O(\kappa)$. Thus, to compute $L_G^+ \mathbf{v}$, we need about $O(\sqrt{\kappa})$ computations of the form $L_H^+ \mathbf{u}$ if we deploy the conjugate gradient method. To summarize, we have proved the following lemma.

Lemma 18.4. There is an algorithm that, given a graph G on n vertices with m edges, a vector $\mathbf{v}$, an $\varepsilon > 0$ and a parameter κ, constructs

a graph H on n vertices such that, with probability at least $1 - 1/\log n$,

(1) $\kappa(L_H^+ L_G) = O(\kappa)$,
(2) the number of edges in H is $n - 1 + O(m \log^2 n/\kappa)$, and
(3) the time it takes to construct H is $O(n \log^2 n + m \log n + m \log^2 n/\kappa)$.

Now, if we apply the greedy degree $1,2$ elimination procedure on H, we obtain G' with $O(m \log^2 n/\kappa)$ edges and vertices. Thus, we have reduced our problem to computing $L_{G'}^+ \mathbf{u}$ for vectors $\mathbf{u}$ where the size of G' has gone down by a factor of κ from that of G.

18.4 Recursive Preconditioning — Proof of the Main Theorem

Now we show how the ideas we developed in the last two sections can be used recursively. Our starting graph $G = G_1$ has $m_1 = m$ edges and $n_1 = n$ vertices. We use a parameter $\kappa < m$ which will be determined later. We then crudely sparsify G_1 using Lemma 18.4 to obtain H_1 such that, with probability $1 - 1/\log n$, (this probability is fixed for all iterations), the number of edges in H_1 is at most $n_1 - 1 + O(m_1 \log^2 n_1/\kappa)$ and $\kappa(L_{H_1}^+ L_{G_1}) = O(\kappa)$. If H_1 does not satisfy these properties, then we abort and restart from G_1. Otherwise, we apply greedy elimination, see Lemma 18.2, to H_1 and obtain G_2; this is a deterministic procedure. If n_2, m_2 are the number of vertices and edges of G_2, then we know that $m_2 = O(m_1 \log^2 n_1/\kappa)$ and $n_2 = O(m_1 \log^2 n_1/\kappa)$. If $n_2 \leq n_f$ (for some n_f to be determined later), then we stop, otherwise we iterate and crude sparsify G_2. We follow this with greedy elimination to get H_2 and proceed to find G_3 and so on. Once we terminate, we have a sequence

$$G = G_1, H_1, G_2, H_2, \ldots, H_{d-1}, G_d,$$

such that

(1) $n_d = O(n_f)$,
(2) For all $1 \leq i \leq d-1$, $\kappa(L_{H_i}^+ L_{G_i}) = O(\kappa)$, and
(3) For all $2 \leq i \leq d-1$, $n_i, m_i \leq \frac{m_{i-1} \log^2 n_{i-1}}{\kappa}$.

Note that n_i, m_i are the number of vertices and edges of G_i, and not H_i. The probability that this process is never restarted is at least $1 - d/\log n$. For this discussion we assume that we use the conjugate gradient method which, when given H and G, can compute $L_G^+ \mathbf{u}$ for a given vector $\mathbf{u}$ with about $\sqrt{\kappa(L_H^+ L_G)}$ computations of the form $L_H^+ \mathbf{v}$ for some vector $\mathbf{v}$. The (incorrect) assumption here is that there is no error in computation and we address this in the next section. The depth of the recursion is

$$d \stackrel{\text{def}}{=} \frac{\log(n/n_f)}{\log(\kappa/\log^2 n)}.$$

Now to compute $L_{G_1}^+ \mathbf{u}$ for a given vector $\mathbf{u}$, the computation tree is of degree $O(\sqrt{\kappa})$ and depth d. At the bottom, we have to compute roughly $(\sqrt{\kappa})^d$ problems of the type $L_{G_d}^+ \mathbf{v}$. We can do this exactly in time $O(n_f^3)$ using Gaussian elimination. Hence, the total work done at the last level is

$$O(n_f^3) \cdot (\sqrt{\kappa})^d.$$

At level i, the amount of work is

$$(\sqrt{k})^i \cdot \tilde{O}(m_i + n_i).$$

We want the sum over all except the top level to be $O(m)$. This is achieved if

$$\sqrt{\kappa} \cdot \frac{\log^2 n}{\kappa} \ll 1.$$

Finally, note that the work done in building the sequence of preconditioners is

$$\sum_i O(m_i \log n_i + n_i \log^2 n_i + m_i \log^2 n_i/\kappa) = O(n \log^2 n + m \log n)$$

if $\kappa \geq \log^3 n$. The depth of the recursion is $d = \frac{\log n/n_f}{\log \kappa/\log^2 n}$. Hence, the choice of parameters is dictated by the following constraints:

(1) $O(n_f^3) \cdot (\sqrt{\kappa})^d = O(m \log^2 n)$,
(2) $O(\sqrt{\kappa} \cdot \frac{\log^2 n}{\kappa}) \leq \frac{1}{2}$, and
(3) $\kappa \geq \log^3 n$.

Under these conditions, the running time of the algorithm is $\tilde{O}(m\sqrt{\kappa})$ at the top level and $O(m\log^2 n)$ in the remaining levels. We set $\kappa \stackrel{\text{def}}{=} 100\log^4 n$ and $n_f \stackrel{\text{def}}{=} \log n$. This choice of κ and n_f also ensures that d is at most $\log n/(2\log\log n)$ and, hence, condition (2) above is also satisfied. Finally, the probability that we never restart is at least $1 - d/\log n \gg 1 - 1/100$ for large enough n. The total running time is bounded by $O((m + n)\log^2 n)$. In the calculations above, note that as long as there is an algorithm that reduces computing $A^+\mathbf{b}$ to computing at most $\kappa(A)^{1-\varepsilon}$ products $A\mathbf{u}$, one can obtain an algorithm that runs in $\tilde{O}(m\log^{O(1/\varepsilon)} n)$ time. This concludes the proof of Theorem 18.1 assuming that there is no error in the application of the conjugate gradient method. In the following section we the intuition behind managing this error.

18.5 Error Analysis and Linearity of the Inverse

As discussed above, the conjugate gradient is not error-free. Since our algorithm uses conjugate gradient recursively, the error could cascade in a complicated manner and it is not clear how to analyze this effect. Here, we use the Chebyshev iteration, see Section 16.5, which achieves the same running time as the conjugate gradient method but has the additional property that on input a matrix A, a vector $\mathbf{b}$, and an $\varepsilon > 0$, outputs $\mathbf{x} = Z\mathbf{b}$ where Z is a matrix such that $(1 - \varepsilon)Z \preceq A^+ \preceq (1 + \varepsilon)Z$. This extra property comes at a price; it requires the knowledge of the smallest and the largest eigenvalues of A. A bit more formally, if we are given a lower bound λ_l on the smallest eigenvalue and an upper bound λ_u on the largest eigenvalue of A, then, after $O(\sqrt{\lambda_u/\lambda_l}\log 1/\varepsilon)$ iterations, the Chebyshev iteration-based method outputs an $\mathbf{x} = Z\mathbf{b}$ such that $(1 - \varepsilon)Z \preceq A^+ \preceq (1 + \varepsilon)Z$.

Let us see how to use this for our application. To illustrate the main idea, consider the case when the chain consists of

$$G = G_1, H_1, G_2, H_2, G_3,$$

where we compute $L^+_{G_3}\mathbf{u}$ exactly. In addition, we know that all the eigenvalues of $L^+_{H_i}L_{G_i}$ lie in the interval $[1, \kappa]$, where we fixed $\kappa \sim \log^4 n$. Given $L^+_{G_3}$, it is easy to obtain $L^+_{H_2}$ since G_3 is obtained by eliminating variables from H_2; hence there is no error. Thus, if we want

to compute $L_{G_2}^{+}\mathbf{u}$, we use the Chebyshev iteration-based solver from Theorem 16.6 with error ε and the guarantee that all eigenvalues of $L_{H_2}^{+}L_{G_2}$ lie in the interval $[1,\kappa]$. Thus, the linear operator that tries to approximate $L_{G_2}^{+}$, Z_2, is such that $(1-\varepsilon)Z_2 \preceq L_{G_2}^{+} \preceq (1+\varepsilon)Z_2$. This is where the error creeps in and linearity is used. From Z_2 we can easily construct $\tilde{Z}_1$ which is supposed to approximate $L_{H_1}^{+}$; namely, $(1-\varepsilon)\tilde{Z}_1 \preceq L_{H_1}^{+} \preceq (1+\varepsilon)\tilde{Z}_1$. This implies that the eigenvalues for our approximator $\tilde{Z}_1 L_{G_1}$ of $L_{H_1}^{+}L_{G_1}$ which were supposed to lie in the interval $[1,\kappa]$ may spill out. However, if we enlarge the interval to $[1/1+\delta,(1+\delta)\kappa]$ for a small constant δ sufficiently bigger than ε, (at the expense of increasing the number of iterations by a factor of $1+\delta$) we ensure that $(1-\varepsilon)Z_1 \preceq L_{G_1}^{+} \preceq (1+\varepsilon)Z_1$. This argument can be made formal via induction by noting that κ, ε, and δ remain fixed through out.

Notes

Theorem 18.1 was first proved by Spielman and Teng [77, 78, 79, 80] and the proof presented in this section, using crude sparsifiers, draws significantly from [49, 50]. The idea of eliminating degree 1 and 2 vertices was implicit in [86]. The idea of recursive preconditioning appears in the thesis [42]. Theorem 18.3 is in [49].

19

Beyond $A\mathbf{x} = \mathbf{b}$
The Lanczos Method

This section looks beyond solving linear equations and considers the more general problem of computing $f(A)\mathbf{v}$ for a given PSD matrix A, vector $\mathbf{v}$ and specified function $f(\cdot)$. A variant of the conjugate gradient method, called the Lanczos method, is introduced which uses Krylov subspace techniques to approximately compute $f(A)\mathbf{v}$. The results of this section can also be used to give an alternative proof of the main result of Section 9 on computing the matrix exponential.

19.1 From Scalars to Matrices

Suppose one is given a symmetric PSD matrix A and a function $f : \mathbb{R} \mapsto \mathbb{R}$. Then one can define $f(A)$ as follows: Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be eigenvectors of A with eigenvalues $\lambda_1, \ldots, \lambda_n$, then $f(A) \stackrel{\text{def}}{=} \sum_i f(\lambda_i)\mathbf{u}_i\mathbf{u}_i^\top$. Given a vector $\mathbf{v}$, we wish to compute $f(A)\mathbf{v}$. One way to do this exactly is to implement the definition of $f(A)$ as above. This requires the diagonalization of A, which is costly. Hence, in the interest of speed, we are happy with an approximation to $f(A)\mathbf{v}$. The need for such primitive often arises in theory and practice. While the conjugate gradient method allows us to do this for $f(x) = 1/x$, it seems specific to this function. For instance, it is not clear how to adapt the conjugate

gradient method to compute $\exp(A)\mathbf{v}$, a primitive central in several areas of optimization and mathematics. See Section 9 for the definition of matrix exponential. In this section we present a meta-method, called the Lanczos method, to compute approximations to $f(A)\mathbf{v}$. The method has a parameter k that provides better and better approximations to $f(A)\mathbf{v}$ as it increases: The error in the approximation after k rounds is bounded by the maximum distance between the best degree k polynomial and f in the interval corresponding to the smallest and the largest eigenvalues of A. The linear algebra problem is, thus, reduced to a problem in approximation theory. The following is the main result of this section.

Theorem 19.1. There is an algorithm that, given a symmetric PSD matrix A, a vector $\mathbf{v}$ with $\|\mathbf{v}\| = 1$, a function f, and a positive integer parameter k, computes a vector $\mathbf{u}$ such that,

$$\|f(A)\mathbf{v} - \mathbf{u}\| \leq 2 \cdot \min_{p_k \in \Sigma_k} \max_{\lambda \in \Lambda(A)} |f(\lambda) - p_k(\lambda)|.$$

Here Σ_k denotes the set of all degree k polynomials and $\Lambda(A)$ denotes the interval containing all the eigenvalues of of A. The time taken by the algorithm is $O((n + t_A)k + k^2)$.

We remark that this theorem can be readily applied to the computation of $\exp(A)\mathbf{v}$; see the notes.

19.2 Working with Krylov Subspace

For a given positive integer k, the Lanczos method looks for an approximation to $f(A)\mathbf{v}$ of the form $p(A)\mathbf{v}$ where p is a polynomial of degree k. Note that for any polynomial p of degree at most k, the vector $p(A)\mathbf{v}$ is a linear combination of the vectors $\{\mathbf{v}, A\mathbf{v}, \ldots, A^k\mathbf{v}\}$. The span of these vectors is referred to as the Krylov subspace of order k of A w.r.t. $\mathbf{v}$ and is defined below.

Definition 19.1. Given a matrix A and a vector $\mathbf{v}$, the Krylov subspace of order k, denoted by $\mathcal{K}_k$, is defined as the subspace that is spanned by the vectors $\{\mathbf{v}, A\mathbf{v}, \ldots, A^k\mathbf{v}\}$.

Since A and $\mathbf{v}$ are fixed, we denote this subspace by $\mathcal{K}_k$. (This definition differs slightly from the one in Section 16 where $\mathcal{K}_{k+1}$ was used to denote this subspace.) Note that any vector in $\mathcal{K}_k$ has to be of the form $p(A)\mathbf{v}$, where p is a degree k polynomial. The Lanczos method to compute $f(A)\mathbf{v}$ starts by generating an orthonormal basis for $\mathcal{K}_k$. Let $\mathbf{v}_0, \ldots, \mathbf{v}_k$ be any orthonormal basis for $\mathcal{K}_k$, and let V_k be the $n \times (k+1)$ matrix with $\{\mathbf{v}_i\}_{i=0}^k$ as its columns. Thus, $V_k^\top V_k = I_{k+1}$ and $V_k V_k^\top$ denotes the projection onto the subspace. Also, let T_k be the operator A in the basis $\{\mathbf{v}_i\}_{i=0}^k$ restricted to this subspace, i.e., $T_k \stackrel{\text{def}}{=} V_k^\top A V_k$. Since all the vectors $\mathbf{v}, A\mathbf{v}, \ldots, A^k\mathbf{v}$ are in the subspace, any of these vectors (or their linear combination) can be obtained by applying T_k to $\mathbf{v}$ (after a change of basis), instead of A. The following lemma states this formally.

Lemma 19.2. Let V_k be the orthonormal basis and T_k be the operator A restricted to $\mathcal{K}_k$ where $\|\mathbf{v}\| = 1$, i.e., $T_k = V_k^\top A V_k$. Let p be a polynomial of degree at most k. Then,

$$p(A)\mathbf{v} = V_k p(T_k) V_k^\top \mathbf{v}.$$

Proof. Recall that $V_k V_k^\top$ is the orthogonal projection onto the subspace $\mathcal{K}_k$. By linearity, it suffices to prove this for $p = x^t$ for all $t \leq k$. This is true for $t = 0$ since $V_k V_k^\top \mathbf{v} = \mathbf{v}$. For all $j \leq k$, $A^j\mathbf{v}$ lies in $\mathcal{K}_k$, thus, $V_k V_k^\top A^j \mathbf{v} = A^j \mathbf{v}$. Hence,

$$\begin{aligned} A^t\mathbf{v} &= (V_k V_k^\top) A (V_k V_k^\top) A \cdots A (V_k V_k^\top)\mathbf{v} \\ &= V_k (V_k^\top A V_k)(V_k^\top A V_k) \cdots (V_k^\top A V_k) V_k^\top \mathbf{v} = V_k T_k^t V_k^\top \mathbf{v}. \end{aligned}$$ □

The next lemma shows that $V_k f(T_k) V_k^\top \mathbf{v}$ approximates $f(A)\mathbf{v}$ as well as the *best* degree k polynomial that uniformly approximates f. The proof is based on the observation that if we express f as a sum of *any* degree k polynomial and an *error* function, the above lemma shows that the polynomial part is computed exactly in this approximation.

Lemma 19.3. Let V_k be the orthonormal basis, and T_k be the operator A restricted to $\mathcal{K}_k$ where $\|\mathbf{v}\| = 1$, i.e., $T_k = V_k^\top A V_k$. Let $f : \mathbb{R} \to \mathbb{R}$ be any function such that $f(A)$ and $f(T_k)$ are well defined. Then, $\left\| f(A)\mathbf{v} - V_k f(T_k) V_k^\top \mathbf{v} \right\|$ is at most

$$\min_{p_k \in \Sigma_k} \left(\max_{\lambda \in \Lambda(A)} |f(\lambda) - p_k(\lambda)| + \max_{\lambda \in \Lambda(T_k)} |f(\lambda) - p_k(\lambda)| \right).$$

Proof. Let p_k be any degree k polynomial. Let $r_k \stackrel{\text{def}}{=} f - p_k$. Then,

$$\begin{aligned}
\left\| f(A)\mathbf{v} - V_k f(T_k) V_k^\top \mathbf{v} \right\| &\leq \left\| p_k(A)\mathbf{v} - V_k p_k(T_k) V_k^\top \mathbf{v} \right\| \\
&\quad + \left\| r_k(A)\mathbf{v} - V_k r_k(T_k) V_k^\top \mathbf{v} \right\| \\
&\stackrel{\text{(Lemma 19.2)}}{\leq} 0 + \|r_k(A)\| + \left\| V_k r_k(T_k) V_k^\top \right\| \\
&= \max_{\lambda \in \Lambda(A)} |r_k(\lambda)| + \max_{\lambda \in \Lambda(T_k)} |r_k(\lambda)|.
\end{aligned}$$

Minimizing over p_k gives us our lemma. □

Observe that in order to compute this approximation, we do not need to know the polynomial explicitly. It suffices to prove that there exists a degree k polynomial that uniformly approximates f on the interval containing the spectrum of A and T_k (for exact computation, $\Lambda(T_k) = [\lambda_{\min}(T_k), \lambda_{\max}(T_k)] \subseteq [\lambda_{\min}(A), \lambda_{\max}(A)] = \Lambda(A)$.) Moreover, if $k \ll n$, the computation is reduced to a much smaller matrix. We now show that an orthonormal basis for the Krylov subspace, V_k, can be computed quickly and then describe the LANCZOS procedure which underlies Theorem 19.1.

19.3 Computing a Basis for the Krylov Subspace

In this section, we show that if we construct the basis $\{\mathbf{v}_i\}_{i=0}^k$ in a particular way, the matrix T_k has extra structure. In particular, if A is symmetric, we show that T_k must be *tridiagonal*. This helps us speed up the construction of the basis.

Algorithm 19.1 LANCZOS

Input: A symmetric, PSD matrix A, a vector $\mathbf{v}$ such that $\|\mathbf{v}\| = 1$, a positive integer k, and a function $f : \mathbb{R} \to \mathbb{R}$

Output: A vector $\mathbf{u}$ that is an approximation to $f(A)\mathbf{v}$

$\mathbf{v}_0 \leftarrow \mathbf{v}$
for $i = 0 \to k-1$ // *Construct an orthonormal basis to Krylov subspace of order k*
do
  if $i = 0$ ***then***
    $\mathbf{w}_0 \leftarrow A\mathbf{v}_0$
  else
    $\mathbf{w}_i \leftarrow A\mathbf{v}_i - \beta_i \mathbf{v}_{i-1}$ // *Orthogonalize w.r.t.* $\mathbf{v}_{i-1}$
  end if
  $\alpha_i \leftarrow \mathbf{v}_i^\top \mathbf{w}_i$
  $\mathbf{w}'_i \stackrel{\text{def}}{=} \mathbf{w}_i - \alpha_i \mathbf{v}_i$ // *Orthogonalize w.r.t.* $\mathbf{v}_i$
  // *If* $\mathbf{w}'_i = 0$*, compute the approximation with the matrices* T_{i-1} *and* V_{i-1}*, instead of* T_k *and* V_k*. The error bound still holds.*
  $\beta_{i+1} \leftarrow \|\mathbf{w}'_i\|$
  $\mathbf{v}_{i+1} \leftarrow \frac{\mathbf{w}'_i}{\beta_{i+1}}$ // *Scaling it to norm* 1
end for
Let V_k *be the* $n \times (k+1)$ *matrix whose columns are* $\mathbf{v}_0, \ldots, \mathbf{v}_k$
Let T_k *be the* $(k+1) \times (k+1)$ *matrix such that for all* i $(T_k)_{ii} = \mathbf{v}_i^\top A \mathbf{v}_i = \alpha_i, (T_k)_{i,i+1} = (T_k)_{i+1,i} = \mathbf{v}_{i+1}^\top A \mathbf{v}_i = \beta_{i+1}$ *and all other entries are 0* // *Compute* $T_k \stackrel{\text{def}}{=} V_k^\top A V_k$
Compute $\mathcal{A} \leftarrow f(T_k)$ exactly via eigendecomposition
return $V_k \mathcal{A} V_k^\top \mathbf{v}$

Suppose we compute the orthonormal basis $\{\mathbf{v}_i\}_{i=0}^k$ iteratively, starting from $\mathbf{v}_0 = \mathbf{v}$: For $i = 0, \ldots, k$, we compute $A\mathbf{v}_i$ and remove the components along the vectors $\{\mathbf{v}_0, \ldots, \mathbf{v}_i\}$ to obtain a new vector that is orthogonal to the previous vectors. This vector, scaled to norm 1, is defined to be $\mathbf{v}_{i+1}$. These vectors, by construction, are such that for all $i \le k$, $\text{Span}\{\mathbf{v}_0, \ldots, \mathbf{v}_i\} = \text{Span}\{\mathbf{v}, A\mathbf{v}, \ldots, A^k\mathbf{v}\}$. Note that $(T_k)_{ij} = \mathbf{v}_i^\top A \mathbf{v}_j$.

If we construct the basis iteratively as above, $A\mathbf{v}_j \in \text{Span}\{\mathbf{v}_0, \ldots, \mathbf{v}_{j+1}\}$ by construction, and if $i > j+1$, $\mathbf{v}_i$ is orthogonal to this subspace and hence $\mathbf{v}_i^\top (A\mathbf{v}_j) = 0$. Thus, T_k satisfies $(T_k)_{ij} = 0$ for $i > j+1$.

Moreover, if A is symmetric, $\mathbf{v}_j^\top (A\mathbf{v}_i) = \mathbf{v}_i^\top (A\mathbf{v}_j)$, and hence T_k is symmetric and tridiagonal. This means that at most three coefficients are nonzero in each row. Thus, while constructing the basis, at step $i+1$, we need to orthonormalize $A\mathbf{v}_i$ only w.r.t. $\mathbf{v}_{i-1}$ and $\mathbf{v}_i$. This fact is used for efficient computation of T_k. The algorithm LANCZOS appears in Figure 19.1.

Completing the Proof of Theorem 19.1

The algorithm LANCZOS implements the Lanczos method discussed in this section. The guarantee on u follows from Lemma 19.3 and the fact that $\Lambda(T_k) \subseteq \Lambda(A)$. We use the fact that $(T_k)_{ij} = \mathbf{v}_i^\top A\mathbf{v}_j$ and that T_k must be *tridiagonal* to reduce our work to just computing $O(k)$ entries in T_k. The total running time is dominated by k multiplications of A with a vector, $O(k)$ dot-products and the eigendecomposition of the tridiagonal matrix T_k to compute $f(T_k)$ (which can be done in $O(k^2)$ time), giving a total running time of $O((n + t_A)k + k^2)$.

Notes

The presentation in this section is a variation of the Lanczos method, a term often used specifically to denote the application of this meta-algorithm to computing eigenvalues and eigenvectors of matrices. It has been used to compute the matrix exponential, see [60, 67, 87]. The Lanczos method was combined with a semidefinite programming technique from [62], a rational approximation result from [70] and the Spielman–Teng Laplacian solver by Orecchia et al. [60] to obtain an $\tilde{O}(m)$ time algorithm for the BALANCED EDGE-SEPARATOR problem from Section 7. Specifically, since Theorem 19.1 does not require the explicit knowledge the best polynomial, it was used in [60] to compute an approximation to $\exp(-L)\mathbf{v}$ and, hence, give an alternative proof of Theorem 9.1 from Section 9. The eigendecomposition result

for tridiagonal matrices, referred to in the proof of Theorem 19.1, is from [63]. The reader is encouraged to compare the Lanczos method with the conjugate gradient method presented in Section 16. Concretely, it is a fruitful exercise to try to explore if the conjugate gradient method can be derived from the Lanczos method by plugging in $f(x) = x^{-1}$.

References

[1] I. Abraham, Y. Bartal, and O. Neiman, "Nearly tight low stretch spanning trees," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 781–790, 2008.

[2] I. Abraham and O. Neiman, "Using petal-decompositions to build a low stretch spanning tree," in *ACM Symposium on Theory of Computing (STOC)*, pp. 395–406, 2012.

[3] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," *Journal of Computer and Systems Sciences*, vol. 66, no. 4, pp. 671–687, 2003.

[4] R. Ahlswede and A. Winter, "Addendum to "Strong converse for identification via quantum channels"," *IEEE Transactions on Information Theory*, vol. 49, no. 1, p. 346, 2003.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[6] N. Alon, R. M. Karp, D. Peleg, and D. B. West, "A graph-theoretic game and its application to the k-server problem," *SIAM Journal on Computing*, vol. 24, no. 1, pp. 78–100, 1995.

[7] N. Alon and V. D. Milman, "λ_1, isoperimetric inequalities for graphs, and superconcentrators," *Journal of Combinational Theory, Series B*, vol. 38, no. 1, pp. 73–88, 1985.

[8] R. Andersen, F. R. K. Chung, and K. J. Lang, "Local graph partitioning using pagerank vectors," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 475–486, 2006.

[9] R. Andersen and Y. Peres, "Finding sparse cuts locally using evolving sets," in *ACM Symposium on Theory of Computing (STOC)*, pp. 235–244, 2009.

[10] S. Arora, E. Hazan, and S. Kale, "$O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 238–247, 2004.

[11] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: A meta-algorithm and applications," *Theory of Computing*, vol. 8, no. 1, pp. 121–164, 2012.

[12] S. Arora and S. Kale, "A combinatorial, primal-dual approach to semidefinite programs," in *ACM Symposium on Theory of Computing (STOC)*, pp. 227–236, 2007.

[13] S. Arora, S. Rao, and U. V. Vazirani, "Expander flows, geometric embeddings and graph partitioning," *Journal of the ACM*, vol. 56, no. 2, 2009.

[14] J. D. Batson, D. A. Spielman, and N. Srivastava, "Twice-Ramanujan sparsifiers," in *ACM Symposium on Theory of Computing (STOC)*, pp. 255–262, 2009.

[15] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *Proceedings of the Workshop on Computational Learning Theory (COLT)*, pp. 624–638, 2004.

[16] A. A. Benczúr and D. R. Karger, "Approximating s–t minimum cuts in $\tilde{O}(n^2)$ time," in *ACM Symposium on Theory of Computing (STOC)*, pp. 47–55, 1996.

[17] R. Bhatia, *Matrix Analysis (Graduate Texts in Mathematics)*. Springer, 1996.

[18] E. G. Boman and B. Hendrickson, "Support theory for preconditioning," *SIAM Journal on Matrix Analysis Applications*, vol. 25, no. 3, pp. 694–717, 2003.

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

[20] J. Cheeger, "A lower bound for the smallest eigenvalue of the Laplacian," *Problems in Analysis*, pp. 195–199, 1970.

[21] E. W. Cheney, *Introduction to Approximation Theory/E. W. Cheney*. New York: McGraw-Hill, 1966.

[22] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S. Teng, "Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs," in *ACM Symposium on Theory of Computing (STOC)*, pp. 273–282, 2011.

[23] F. R. K. Chung, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, 1997.

[24] S. I. Daitch and D. A. Spielman, "Faster approximate lossy generalized flow via interior point algorithms," in *ACM Symposium on Theory of Computing (STOC)*, pp. 451–460, 2008.

[25] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks*. Washington, DC: Mathematical Association of America, 1984.

[26] P. Elias, A. Feinstein, and C. Shannon, "A note on the maximum flow through a network," *IEEE Transactions on Information Theory*, vol. 2, no. 4, pp. 117–119, December 1956.

[27] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, "Lower-stretch spanning trees," *SIAM Journal on Computing*, vol. 38, no. 2, pp. 608–628, 2008.

[28] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, pp. 298–305, 1973.

[29] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1954.

[30] A. Frangioni and C. Gentile, "Prim-based support-graph preconditioners for min-cost flow problems," *Computational Optimization and Applications*, vol. 36, no. 2–3, pp. 271–287, April 2007.

[31] W. S. Fung, R. Hariharan, N. J. A. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *ACM Symposium on Theory of Computing (STOC)*, pp. 71–80, 2011.

[32] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[33] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, 1973.

[34] C. D. Godsil and G. Royle, *Algebraic Graph Theory*. Springer, 2001.

[35] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins Univ. Press, 1996.

[36] G. H. Golub and M. L. Overton, "The convergence of inexact Chebyshev and Richardson iterative methods for solving linear systems," Technical Report, Stanford University, Stanford, CA, USA, 1987.

[37] G. H. Golub and R. S. Varga, "Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods," *Numerische Mathematik*, vol. 3, pp. 157–168, 1961.

[38] K. Gremban, "Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems," PhD Thesis, Carnegie Mellon University, Pittsburgh, CMU CS Tech Report CMU-CS-96-123, October 1996.

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

[40] G. Iyengar, D. J. Phillips, and C. Stein, "Approximating semidefinite packing programs," *SIAM Journal on Optimization*, vol. 21, no. 1, pp. 231–268, 2011.

[41] G. Iyengar, D. J. Phillips, and C. Stein, "Approximation algorithms for semidefinite packing problems with applications to maxcut and graph coloring," in *IPCO'05: Proceedings of Conference on Integer Programming and Combinatorial Optimization*, pp. 152–166, 2005.

[42] A. Joshi, "Topics in optimization and sparse linear systems," PhD Thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, UMI Order No. GAX97-17289, 1997.

[43] S. Kaczmarz, "Ängenäherte Auflösung von Systemen linearer Gleichungen," *Bulletin International del' Académie Polonaise Sciences et des Lettres*, pp. 355–356, 1937.

[44] S. Kale, "Efficient algorithms using the multiplicative weights update method," PhD Thesis, Princeton University, Department of Computer Science, 2007.

[45] J. A. Kelner and A. Madry, "Faster generation of random spanning trees," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 13–21, 2009.

[46] J. A. Kelner, G. L. Miller, and R. Peng, "Faster approximate multicommodity flow using quadratically coupled flows," in *ACM Symposium on Theory of Computing (STOC)*, pp. 1–18, 2012.

[47] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *ACM Symposium on Theory of Computing (STOC)*, 2013.

[48] R. Khandekar, S. Rao, and U. V. Vazirani, "Graph partitioning using single commodity flows," *Journal of the ACM*, vol. 56, no. 4, 2009.

[49] I. Koutis, G. L. Miller, and R. Peng, "Approaching optimality for solving SDD linear systems," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 235–244, 2010.

[50] I. Koutis, G. L. Miller, and R. Peng, "A nearly-$m \log n$ time solver for SDD linear systems," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 590–598, 2011.

[51] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times*. American Mathematical Society, 2006.

[52] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM Journal on Numerical Analysis*, vol. 16, no. 2, pp. 346–358, 1979.

[53] L. Lovász and M. Simonovits, "Random walks in a convex body and an improved volume algorithm," *Random Structures & Algorithms*, vol. 4, no. 4, pp. 359–412, 1993.

[54] R. Lyons and Y. Peres, *Probability on Trees and Networks*. To Appear in Cambridge University Press, 2012.

[55] A. Madry, "Fast approximation algorithms for cut-based problems in undirected graphs," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 245–254, 2010.

[56] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi, "A spectral algorithm for improving graph partitions," *Journal of Machine Learning Research*, vol. 13, pp. 2339–2365, 2012.

[57] S. Maji, N. K. Vishnoi, and J. Malik, "Biased normalized cuts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2057–2064, 2011.

[58] M. Mihail, "Conductance and convergence of markov chains — a combinatorial treatment of expanders," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 526–531, 1989.

[59] L. Orecchia, "Fast approximation algorithms for graph partitioning using spectral and semidefinite-programming techniques," PhD Thesis, EECS Department, University of California, Berkeley, May 2011.

[60] L. Orecchia, S. Sachdeva, and N. K. Vishnoi, "Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator," in *ACM Symposium on Theory of Computing (STOC)*, pp. 1141–1160, 2012.

[61] L. Orecchia, L. J. Schulman, U. V. Vazirani, and N. K. Vishnoi, "On partitioning graphs via single commodity flows," in *ACM Symposium on Theory of Computing (STOC)*, pp. 461–470, 2008.

[62] L. Orecchia and N. K. Vishnoi, "Towards an SDP-based approach to spectral methods: A nearly-linear-time algorithm for graph partitioning and decomposition," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 532–545, 2011.

[63] V. Y. Pan and Z. Q. Chen, "The complexity of the matrix eigenproblem," in *ACM Symposium on Theory of Computing (STOC)*, pp. 507–516, 1999.

[64] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations with an application to the stresses in a masonry dam," *Transactions of the Royal Society of London*, vol. Series A, no. 210, pp. 307–357, 1910.

[65] T. J. Rivlin, *An Introduction to the Approximation of Functions*. Blaisdell book in numerical analysis and computer science. Blaisdell Pub. Co., 1969.

[66] M. Rudelson, "Random vectors in the isotropic position," *Journal of Functional Analysis*, vol. 164, no. 1, pp. 60–72, 1999.

[67] Y. Saad, "Analysis of some Krylov subspace approximations to the matrix exponential operator," *SIAM Journal on Numerical Analysis*, vol. 29, pp. 209–228, February 1992.

[68] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics. Philadelphia, PA, USA, 2nd Edition, 2003.

[69] S. Sachdeva and N. K. Vishnoi, "Inversion is as easy as exponentiation," *manuscript*, 2012.

[70] E. B. Saff, A. Schonhage, and R. S. Varga, "Geometric convergence to e^{-z} by rational functions with real poles," *Numerische Mathematik*, vol. 25, pp. 307–322, 1975.

[71] J. Sherman, "Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009.

[72] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Technical Report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[73] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888–905, 1997.

[74] D. B. Shmoys, *Cut Problems and Their Application to Divide-and-Conquer*. pp. 192–235, 1997.

[75] D. A. Spielman, "Algorithms, graph theory, and linear equations in Laplacian matrices," in *Proceedings of International Congress of Mathematicians (ICM'10)*, 2010.

[76] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011.

[77] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," in *ACM Symposium on Theory of Computing (STOC)*, pp. 81–90, New York, NY, USA, 2004.

[78] D. A. Spielman and S.-H. Teng, "Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," CoRR, abs/cs/0607105, 2006.

[79] D. A. Spielman and S.-H. Teng, "A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning," CoRR, abs/0809.3232, 2008.

[80] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 981–1025, 2011.

[81] D. A. Spielman and J. Woo, "A note on preconditioning by low-stretch spanning trees," CoRR, abs/0903.2816, 2009.

[82] G. Strang, *Linear Algebra and Its Applications.* Harcourt Brace Jonanovich. San Diego, 3rd Edition, 1988.

[83] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *Journal of Fourier Analysis and Applications*, vol. 15, pp. 262–278, 2009.

[84] S. Teng, "The Laplacian paradigm: Emerging algorithms for massive graphs," in *Proceedings of the Annual Conference on Theory and Applications of Models of Computation (TAMC)*, pp. 2–14, 2010.

[85] L. N. Trefethen and D. Bau, *Numerical Linear Algebra.* SIAM, 1997.

[86] P. M. Vaidya, "Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners," Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[87] J. vanden Eshof and M. Hochbruck, "Preconditioning Lanczos approximations to the matrix exponential," *SIAM Journal on Scientific Computing*, vol. 27, pp. 1438–1457, November 2005.

[88] J. H. Wilkinson, *The Algebraic Eigenvalue Problem (Monographs on Numerical Analysis).* USA: Oxford University Press, 1st Edition, April 1988.

[89] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 912–919, 2003.