Algorithm Design Using Spectral Graph Theory

Richard Peng

CMU-CS-13-121
August 2013

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Gary L. Miller, Chair
Guy E. Blelloch
Alan Frieze
Daniel A. Spielman, Yale University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported in part by the National Science Foundation under grant number CCF-1018463, by a Microsoft Research PhD Fellowship, by the National Eye Institute under grant numbers R01-EY01317-08 and RO1-EY11289-22, and by the Natural Sciences and Engineering Research Council of Canada under grant numbers M-377343-2009 and D-390928-2010. Parts of this work were done while at Microsoft Research New England. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the funding parties.

Keywords: Combinatorial Preconditioning, Linear System Solvers, Spectral Graph Theory, Parallel Algorithms, Low Stretch Embeddings, Image Processing

To my parents and grandparents

Abstract

Spectral graph theory is the interplay between linear algebra and combinatorial graph theory. Laplace's equation and its discrete form, the Laplacian matrix, appear ubiquitously in mathematical physics. Due to the recent discovery of very fast solvers for these equations, they are also becoming increasingly useful in combinatorial optimization, computer vision, computer graphics, and machine learning.

In this thesis, we develop highly efficient and parallelizable algorithms for solving linear systems involving graph Laplacian matrices. These solvers can also be extended to symmetric diagonally dominant matrices and M-matrices, both of which are closely related to graph Laplacians. Our algorithms build upon two decades of progress on combinatorial preconditioning, which connects numerical and combinatorial algorithms through spectral graph theory. They in turn rely on tools from numerical analysis, metric embeddings, and random matrix theory.

We give two solver algorithms that take diametrically opposite approaches. The first is motivated by combinatorial algorithms, and aims to gradually break the problem into several smaller ones. It represents major simplifications over previous solver constructions, and has theoretical running time comparable to sorting. The second is motivated by numerical analysis, and aims to rapidly improve the algebraic connectivity of the graph. It is the first highly efficient solver for Laplacian linear systems that parallelizes almost completely.

Our results improve the performances of applications of fast linear system solvers ranging from scientific computing to algorithmic graph theory. We also show that these solvers can be used to address broad classes of image processing tasks, and give some preliminary experimental results.

Acknowledgments

This thesis is written with the help of many people. First and foremost I would like to thank my advisor, Gary Miller, who introduced me to most of the topics discussed here and helped me throughout my studies at CMU. My thesis committee members, Guy Blelloch, Alan Frieze, and Daniel Spielman, provided me with invaluable advice during the dissertation process. I am also grateful towards Ian Munro and Daniel Sleator for their constant guidance.

I am indebted to the co-authors who I had the fortune to work with during my graduate studies. Many of the ideas and results in this thesis are due to collaborations with Yiannis Koutis, Kanat Tangwongsan, Hui Han Chin, and Shen Chen Xu. Michael Cohen, Jakub Pachocki, and Shen Chen Xu also provided very helpful comments and feedback on earlier versions of this document. I was fortunate to be hosted by Aleksander Mądry and Madhu Sudan while interning at Microsoft Research New England. While there, I also had many enlightening discussions with Jonathan Kelner and Alex Levin.

I would like to thank my parents and grandparents who encouraged and cultivated my interests; and friends who helped me throughout graduate school: Bessie, Eric, Hui Han, Mark, Neal, Tom, and Travis. Finally, I want to thank the CNOI, CEMC, IOI, USACO, and SGU for developing my problem solving abilities, stretching my imagination, and always giving me something to look forward to.

Contents

List of Figures

Chapter 1

Introduction

Solving a system of linear equations is one of the oldest and possibly most studied computational problems. There is evidence that humans have been solving linear systems to facilitate economic activities over two thousand years ago. Over the past two hundred years, advances in physical sciences, engineering, and statistics made linear system solvers a central topic of applied mathematics. This popularity of solving linear systems as a subroutine is partly due to the existence of efficient algorithms: small systems involving tens of variables can be solved reasonably fast even by hand.

Over the last two decades, the digital revolution led to a wide variety of new applications for linear system solvers. At the same time, improvements in sensor capabilities, storage devices, and networks led to sharp increases in data sizes. Methods suitable for systems with a small number of variables have proven to be challenging to scale to the large, sparse instances common in these applications. As a result, the design of efficient linear system solvers remains a crucial algorithmic problem.

Many of the newer applications model entities and their relationships as networks, also known as graphs, and solve linear systems based on these networks. The resulting matrices are called graph Laplacians, which we will formally define in the next section. This approach of extracting information using graph Laplacians has been referred to as the Laplacian Paradigm [Ten10]. It has been successfully applied to a growing number of areas including clustering, machine learning, computer vision, and algorithmic graph theory. The main focus of this dissertation is the routine at the heart of this paradigm: solvers for linear systems involving graph Laplacians.

1.1 Graphs and Linear Systems

One area where graph-based linear systems arise is the analysis of social networks. A social network can be represented as a set of links connecting pairs of people; an example is shown in Figure 1.1. In graph theoretic terms, the people correspond to vertices while the links correspond to edges. Given such a network, a natural question to ask is how closely related two people are. Purely graph-based methods may measure either the length of the shortest path or the maximum number of disjoint paths between the two corresponding vertices. One way to take both measures into account is to view the network as an electric circuit with each connection corresponding to

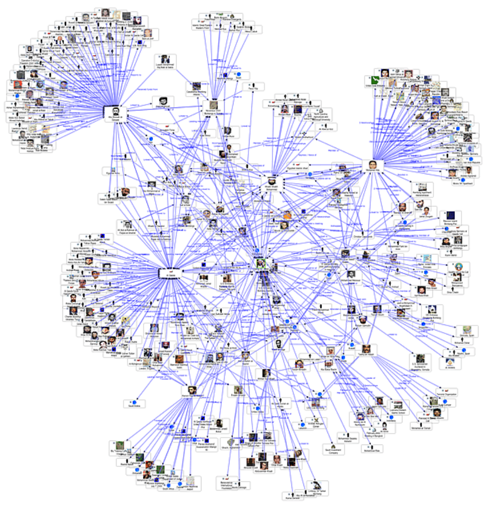


Figure 1.1: Representing a social network as a graph.

an electrical wire (resistor). The electrical resistance between two vertices then gives a quantity known as the effective resistance.

Intuition from electrical circuits suggests that shortening wires and duplicating wires can both reduce resistance. Therefore effective resistance and its variants can be used as measures of connectivity on networks, with lower resistance corresponding to being more closely connected. Similar methods are applicable to a wide range of problems such as measuring the importance of specific proteins in protein-protein interaction networks [MLZ$^+$09] and the link prediction problem in social networks [LNK07]. However, as the electrical network is not physically available, we can't measure the effective resistance experimentally. We can, however, compute it by solving a linear system.

To compute effective resistance, let us consider the more general problem of finding the voltages at all the vertices given that 1 unit of current passes between two specific vertices. For notational purposes it is more convenient to describe each resistor by its conductance, which is the inverse of its resistive value and analogous to edge weight in the corresponding graph. The setting of voltages at the vertices induces an electrical flow through the edges. There are two fundamental principles governing this voltage setting and electrical flow:

- Kirchhoff's law: With the exception of the vertices attached to the power source, the net amount of current entering and leaving each vertex is 0.
- Ohm's law: The current on an edge equals to the voltage difference between its endpoints times the conductance of the resistor.

As an example, consider the resistive network with 3 vertices depicted in Figure 1.2. Suppose we set the voltages at the three vertices to be x_1, x_2 and x_3 respectively. By Ohm's law we get that the currents along the edges $1 \to 2$ and $1 \to 3$ are $1 \cdot (x_1 - x_2)$ and $2 \cdot (x_1 - x_3)$ respectively.

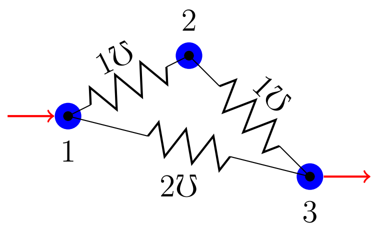


Figure 1.2: A resistive electric network with resistors labeled by their conductances (℧)

Therefore the amount of current that we need to inject into vertex 1 to maintain these voltages is:

$$1 \cdot (x_1 - x_2) + 2 \cdot (x_1 - x_3) = 3x_1 - x_2 - 2x_3$$

Identities for the required current entering/leaving vertices 2 and 3 can also be derived similarly. Therefore, if we want one unit of current to enter at vertex 1 and leave at vertex 3, the voltages need to satisfy the following system of linear equations:

$$\begin{aligned} 3x_1 - 1x_2 - 2x_3 &= 1 \\ -1x_1 + 2x_2 - 1x_3 &= 0 \\ -2x_1 - 1x_2 + 3x_3 &= -1 \end{aligned}$$

Ohm's law gives that the voltage drop required to send 1 unit of current is the resistance. Therefore, once we obtain the voltages, we can also compute the effective resistance. In our example, a solution for the voltages is $x_1 = 2/5$, $x_2 = 1/5$, and $x_3 = 0$, so the effective resistance between vertices 1 and 3 is $x_1 - x_3 = 2/5$.

This model of the graph as a circuit is very much simplified from the point of view of circuit theory. However, it demonstrates that problems on graphs can be closely associated with linear systems. More generally, linear systems assume the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an $n \times n$ matrix containing real numbers, $\mathbf{b}$ is an $n \times 1$ column vector of real numbers, and $\mathbf{x}$ is an $n \times 1$ column vector of unknowns, also called variables. For example, the above linear system can be expressed in matrix form as:

$$\begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

Note that each off-diagonal entry is the negation of the conductance between its two vertices, and each diagonal entry is the sum of the conductances of all resistors incident to that vertex. Since resistive networks are essentially undirected graphs with edge weights equaling conductance, this type of matrix is known as a graph Laplacian. It is the fundamental object studied in spectral graph theory, and can be defined as follows:

Definition 1.1.1 *The graph Laplacian $\mathbf{L}_G$ of a weighted graph $G = (V, E, \boldsymbol{w})$ with n vertices is* *an $n \times n$ matrix whose entries are:*

$$\begin{aligned} \forall u \neq v: \quad \boldsymbol{L}_{G,uv} &= -\boldsymbol{w}_{uv} \\ \forall u: \quad \boldsymbol{L}_{G,uu} &= \sum_{v \neq u} \boldsymbol{w}_{uv} \end{aligned}$$

In numerical analysis, graph Laplacians belong to the category of Symmetric Diagonally Dominant (SDD) matrices. These are a further generalization of graph Laplacians where the off-diagonal entries can be positive. Also, the diagonal entry can exceed the total weight of the off-diagonal entries, instead of having to be equal to it.

Definition 1.1.2 *A symmetric matrix **M** is symmetric diagonally dominant (SDD) if:*

$$\forall i: \quad \boldsymbol{M}_{ii} \geq \sum_{j \neq i} |\boldsymbol{M}_{ij}|$$

Gremban and Miller [Gre96] showed that solving a symmetric diagonally dominant linear system reduces to solving a graph Laplacian system with twice the size. This reduction is also well-understood for approximate solvers [ST06], and in the presence of fixed point round-off errors [KOSZ13]. A more intricate extension that solves systems involving M-matrices using Laplacian solvers was also shown by Daitch and Spielman [DS07]. These reductions allow us to focus on solving systems involving graph Laplacians for the rest of this thesis.

1.2 Related Work

Despite its long history, the problem of constructing fast linear system solvers is considered far from being solved. The speed of algorithms is commonly measured in terms of the input size. In the case of general linear systems on n variables, the matrix is of size n^2. However, the sizes of sparse matrices are better measured by the number of non-zero entries, denoted by m. The best case scenario, which remains entirely consistent with our current understanding, is that any linear system can be solved with $O(m)$[1] operations.

Linear system solvers can be classified into two main categories: direct and iterative methods. Direct methods aim to modify the matrix to become the identity matrix using a sequence of reversible operations. Inverses of these operations are then composed to form the inverse of the matrix.

It's fair to say that Gaussian elimination is the most well-known method for solving linear systems. It runs in $O(n^3)$ time on an $n \times n$ matrix, and is a direct method. Strassen [Str69] first demonstrated that this exponent of 3 is not tight, giving an algorithm that runs in $O(n^{\log_2 7})$ time. Subsequently this exponent has been further decreased, with $O(n^{2.3754})$ by Coppersmith and

[1]We use $f(n) = O(g(n))$ to denote $f(n) \leq c \cdot g(n)$ when $n \geq n_0$ for some constants c and n_0.

Winograd [CW90] holding as the record for twenty years. Recently, further improvements were made by Stothers [Sto10] and Vassilevska Williams [Wil12], with the latter taking $O(n^{2.3727})$ time. On a dense matrix with $m \approx n^2$, these algorithms can be viewed as taking between $O(m^{3/2})$ and $O(m^{1.18})$ operations. However, for sparse matrices with n in the millions, a more crucial limiting factor for direct algorithms is the need to compute and store a dense, $\Omega(n^2)$ sized, inverse.

Iterative methods, on the other hand, generate a sequence of solution vectors that converge to the solution. Most iterative methods only maintain a small number of length n vectors per step, and perform matrix-vector multiplications in each iteration. This makes them more suitable for solving the large sparse linear systems that are common in practice.

One of the most important iterative methods the conjugate gradient algorithm due to Lanczos [Lan52], Hestenes, and Stiefel [HS52]. Assuming exact real arithmetic, this method takes $O(n)$ iterations to obtain an exact answer, giving a total cost of $O(nm)$. However, this bound is fragile in the presence of floating point round-off errors, and preconditioners are needed to drastically improve the performances of the conjugate algorithm. A more detailed treatment of iterative methods can be found in books such as [Axe94].

Combinatorial preconditioning is a way to combine the best from both direct and iterative methods. The combinatorial structure of graph Laplacians allows one to analyze iterative methods using graph theory. The first algorithms by Vaidya [Vai91] constructed preconditioners by adding edges to a maximum weight spanning tree. These methods led to an $O((dn)^{1.75})$ time algorithm for graphs with maximum degree d, and an $O((dn)^{1.2})$ time algorithm when the graph is planar [Jos97]. Subsequently, methods of constructing preconditioners using Steiner trees have also been investigated [GMZ94].

Subsequently, Boman and Hendrickson [BH01] made the crucial observation that it is better to add edges to a different tree known as the low-stretch spanning tree. Such trees were first introduced by Alon, Karp, Peleg, and West [AKPW95], and have been well-studied in the setting of online algorithms and metric embeddings. Exploring this connection further led to the result by Spielman and Teng [ST03], which runs in $O(m^{1.31})$ time for general graphs. They also identified spectral sparsification, the problem of finding a sparse equivalent of a dense graph, as the key subroutine needed for a better algorithm. Subsequently, they gave a spectral sparsification algorithm that produces a nearly-linear sized sparsifiers in nearly linear time [ST08, ST06]. Incorporating it into the previous solver framework led to the the first algorithm for solving Laplacian systems that run in nearly-linear, or $O(m \log^c n)$, time. Our sequential algorithm in Chapter 2 also builds upon this framework.

Most direct methods can be parallelized [JáJ92], while the matrix-vector multiplications performed in each step of iterative methods are also inherently parallel. The first parallel algorithm for solving graph Laplacians with non-trivial parallel speedups was obtained in the setting of planar graphs by Koutis and Miller [KM07]. It parallelizes the framework from the Spielman and Teng algorithm, and was extended to general graphs by Blelloch et al. [BGK$^+$13]. Our parallel algorithm in Chapter 3 uses a different approach based primarily on spectral sparsification. Due to the intricacies of measuring performances of parallel algorithms, we defer a more detailed discussion of the related works on parallel algorithms to Chapter 3.

Recently, an alternate approach for solvers was developed by Kelner et al. [KOSZ13]. They replace the sparsification and recursive preconditioning framework with data structures, but retain the low-stretch spanning trees. The non-recursive structure of this algorithm leads to a much more direct analysis of its numerical stability in the unit cost RAM model. Subsequent improvements by Lee and Sidford [LS13] led to a running time of about $O(m \log^{3/2} n)$.

1.3 Structure of This Thesis

This thesis presents two highly efficient algorithms for solving linear systems involving graph Laplacians, and consequently SDD matrices. Both of these algorithms run in nearly-linear time. That is, as the size of the input doubles, the running times of the algorithms increase by factors that approach 2. These algorithms take diametrically opposite approaches, and can be read in any order. However, both rely heavily on the connection between graph Laplacians and graphs. Readers more familiar with combinatorial or graph theoretic algorithms might find the sequential algorithm in Chapter 2 more natural, while the approach taken for the parallel algorithm in Chapter 3 is more motivated by numerical analysis. As matrices are multi-dimensional generalizations of scalars, we give a brief overview of our algorithms in the 1-dimensional case in Section 1.5, and introduce the relevant linear algebraic tools in Section 1.6.

1.3.1 Sequential SDD Linear System Solver

Chapter 2 gives an algorithm that solves a SDD linear system with m non-zero entries to a relative error of ϵ in $O(m \log n \log(1/\epsilon))$ time after spending $O(m \log n \log \log n)$ time preprocessing it. It builds upon the recursive preconditioning framework by Spielman and Teng [ST06], which reduces solving a linear system to solving several smaller systems. Our algorithm gives both an improved routine for generating these smaller systems, and a more holistic view of the framework that leads to a faster running time. It is based on two papers joint with Ioannis Koutis and Gary Miller [KMP10, KMP11], as well as subsequent unpublished improvements joint with Gary Miller.

1.3.2 Stability of Recursive Preconditioning Under Round-off Errors

A common assumption made in most analyses of combinatorial preconditioning algorithms to date is that all calculations are performed exactly. Actual computation involve numbers of bounded length, which leads to round-off errors. For the recursive preconditioning framework, which our algorithm in Chapter 2 is based on, an earlier analysis by Spielman and Teng [ST06] needed to increase the lengths of numbers by a factor of $\Theta(\log n)$ to guarantee convergence. This in part motivated the algorithms by Kelner et al. [KOSZ13] and Lee and Sidford [LS13], which are more numerically stable. In Section 2.6, we give a round-off error analysis of the recursive preconditioning framework which incorporates many of our other improvements. We show that the $O(m \log n \log(1/\epsilon))$ time algorithm is numerically stable under similar assumptions: a constant factor increase in the lengths of numbers is sufficient for convergence under fixed-point arithmetic.

1.3.3 Parallel SDD Linear System Solver

Chapter 3 focuses on constructing nearly-linear work solver algorithms with better parallel performances. It constructs a small number of sparse matrices whose product approximates the inverse. As matrix-vector multiplications are readily parallelizable, evaluating such a representation leads to a parallel solver algorithm. For a graph Laplacian with edge weights in the range $[1, U]$, the algorithm computes a solution with a relative error of ϵ in $O\left(\log^c(nU)\log(1/\epsilon)\right)$ depth $O\left(m\log^c(nU)\log(1/\epsilon)\right)$ work[2] for some constant c. This result is joint with Dan Spielman and has yet to be published.

Due to the relatively large logarithmic factors in the performances of this algorithm, we omit analyzing its numerical stability in the belief that significant improvements and simplifications are likely. Some possible directions for improvements are discussed in Chapter 6.

1.3.4 Graph Embeddings

At the core of our solver algorithms are routines that construct graphs that are similar to the input graph. We will formalize this similarity, and describe relevant tools and algorithms in Section 1.6. In our algorithms, such routines are often used along with an embedding of the graph into a tree. Some modifications of known results are needed to obtain the strongest versions of our results in Chapters 2 and 3. These modifications are presented in Chapter 4.

As the development of better tree embeddings is an active area of research, the presentation aims for simplicity. It is based on two papers joint with Guy Blelloch, Anupam Gupta, Ioannis Koutis, Gary Miller, Kanat Tangwongsan, and Shen Chen Xu [BGK$^+$13, MPX13], as well as unpublished improvements joint with Gary Miller and Shen Chen Xu.

1.3.5 Image Processing

The speed of SDD linear system solvers both in theory and in practice has led to many applications using solvers as a subroutine. A short survey of these applications is in Section 1.4. In Chapter 5, we focus on an area where combinatorial preconditioners are on the verge of entering practice: image processing. We show that recent advances in graph theoretic algorithms can be used to optimize a broad class of image processing objectives.

We formulate an optimization objective which we term the the grouped least squares objective that's applicable to many problems in computer vision. It also represents a smooth interpolation between quadratic and linear minimization in undirected graphs. This connection allows us to draw upon the approximate minimum cut algorithm by Christiano et al. [CKM$^+$11] and design an algorithm for minimizing the grouped least squares objective. We also demonstrate the practical feasibility of our approach by performing preliminary experiments on well-known image processing examples. It is based on a paper joint with Hui Han Chin, Aleksander Mądry, and Gary Miller [CMMP13].

[2] Depth/work is a standard measure of the efficiency of parallel algorithms, with depth corresponding to the longest chain of sequential dependencies and work corresponding to the number of operations performed.

1.4 Applications

Because of the central role played by graph Laplacians in spectral graph theory and mathematical physics, algorithmic applications of graph Laplacians predated, and in some sense motivated, the design of more efficient solver algorithms. One of the earlier such results was by Tutte [Tut63]. He embeds a 3-connected planar graph in the 2-D plane without edge crossings by solving two graph Laplacian linear systems.

A more powerful object derived from the graph Laplacian is the principle eigenvector. Also known as the Fiedler vector, its properties has been extensively studied in spectral graph theory [Fie73, ST96, Chu97]. Along with the solver algorithm [ST06], Spielman and Teng showed that a small number of calls to the solver allows one to compute a good approximation to this vector. An improved algorithm using matrix exponentials, which in turn relies on solving SDD linear systems, was given by Orecchia et al. [OSV12]. Alon and Millman [AM85] showed that a discrete variant of an inequality by Cheeger [Che70] gives bounds on the quality of the cut returned from scanning this eigenvector. Such algorithms were introduced in the setting of image processing by Shi and Malik [SM97], and is now one of the standard tools for clustering. It has a wide range of applications in statistics, computer vision, and machine learning.

In numerical analysis, graph Laplacians have direct connections with elliptic finite element methods. They approximate the solution to a variety of differential equations on meshes used in applications such as collision modeling. These connections were made explicit by Boman et al. [BHV08], who showed that when the underlying discretization of space (mesh) is well structured, the resulting linear system is algebraically close to a graph Laplacian, and can therefore be solved using a few invocations to a solver for systems involving graph Laplacians.

The discovery of theoretically fast solvers for SDD linear systems [ST06] allowed their use as primitives in designing graph algorithms. These solvers are the key subroutines of the fastest known algorithms for a multitude of graph problems including:

- Computing balanced separators of graphs using matrix exponentials and heat-kernel random walks by Orecchia et al. [OSV12].
- Generating spectral sparsifiers, which also act as cut-preserving sparsifiers by Spielman and Srivastava [SS08]. Improvements to this algorithm for the case of dense graphs were also given by Koutis et al. [KLP12].
- Finding minimum cost flows and generalized lossy flow by Daitch and Spielman [DS08].
- Computing maximum bipartite matchings (or more generally maximum flow in unit capacity graphs) by Mądry [Mąd13].
- Approximating maximum flows in undirected graphs by Christiano et al. [CKM$^+$11] and Lee et al.[LRS13].
- Generating random spanning trees by Kelner and Mądry [KM09a].

Our solver algorithm from Chapter 2 leads to improved theoretical running times for all these applications. Some of these algorithms can also be combined with parallel solvers. Somewhat surprisingly, for graph optimization problems such as maximum flow, this leads to the best parallel algorithms whose operation count is comparable to the state-of-the-art sequential ones. This connection was first observed by Blelloch et al. [BGK$^+$13], and our algorithm from Chapter 3 also improves these parallel algorithms.

One area where combinatorial preconditioning has already been used in practice is computer vision. The lattice-like underlying connectivity structure allows for the construction of combinatorial preconditioners under local sparsity assumptions. The presence of weights and extra edges introduced by non-local methods also makes combinatorial preconditioning more suitable than methods that assume uniformity. A code package known as the Combinatorial Multigrid (CMG) Solver was shown to be applicable to a variety of image segmentation and denoising tasks by Koutis et al. [KMST09], and to gradient in-painting by McCann and Pollard [MP08]. Our image processing experiments in Chapter 5 were also performed using the CMG Solver package.

1.5 Solving Linear Systems

Many key ideas in our algorithms can be illustrated using the simplest SDD linear systems: the 1×1 matrix. Here we can omit linear algebraic notations and treat this entry as a scalar, ℓ. As ℓ^{-1} is equivalent to the inverse of this matrix, our goal is to find $\ell^{-1}b$ without dividing by ℓ. This is a reasonable simplification from higher dimensions since a matrix-vector multiplication by $\mathbf{A}$ is much less expensive than finding its inverse $\mathbf{A}^{-1}$. This goal can also be viewed as constructing a routine that divides by ℓ using only multiplications.

For simplicity, first consider the situation where ℓ is close to 1, e.g.:

$$\frac{1}{2} \leq \ell \leq \frac{3}{2}$$

We can invoke an identity about geometric series that tells us that when $0 < \ell < 2$, $1/\ell$ equals the following infinite sum known as a power series:

$$\begin{aligned} \ell^{-1} &= 1/(1-(1-\ell)) \\ &= 1 + (1-\ell) + (1-\ell)^2 + (1-\ell)^3 + \ldots \end{aligned} \tag{1.1}$$

Of course, computing an infinite sum is not possible. On the other hand, the first t terms forms a polynomial and evaluating it may lead to a reasonable approximation. This can be done by either computing $(1-\ell)^i$ in increasing order for all $0 \leq i < t$, or using Horner's rule (see e.g. Chapter 2 of [CSRL01]). In either case, t multiplications suffices to evaluate the first t terms of this power series.

This is the simplest example of an iterative method, which approximates the solution of a linear system using a series of matrix-vector multiplications. Iterative methods form the main framework of both of our sequential and parallel algorithms. This particular power series can also be interpreted as computing the stationary distribution of a random walk, and we will explore this connection in more detail in Section 3.1.

The convergence of this type of method depends heavily on the value of $1 - \ell$. The error of using the first t terms is given by the tail of infinite power series:

$$(1-\ell)^t + (1-\ell)^{t+1} + (1-\ell)^{t+2} + \dots \tag{1.2}$$
$$= (1-\ell)^t \left(1 + (1-\ell)^1 + (1-\ell)^2 + \dots\right) \tag{1.3}$$
$$= (1-\ell)^t \ell^{-1} \tag{1.4}$$

If ℓ is in the range $[1/2, 3/2]$, $1 - \ell$ is between $[-1/2, 1/2]$ and $(1-\ell)^t$ approaches 0 geometrically. Therefore evaluating a small number of terms of the power series given in Equation 1.1 leads to a good estimate. If we want an answer in the range $[(1-\epsilon)\,\ell^{-1}b, (1+\epsilon)\,\ell^{-1}b]$ for some error $\epsilon > 0$, it suffices to evaluate the first $O(\log(1/\epsilon))$ terms of this series.

On the other hand, convergence is not guaranteed when $|1 - \ell|$ is larger than 1. Even if $1 - \ell$ is between $(-1, 1)$, convergence may be very slow. For example, if ℓ is $\frac{1}{\kappa}$ for some large κ, the definition of natural logarithm gives $(1-\ell)^\kappa \approx \frac{1}{e}$. That is, $O(\kappa)$ steps are needed to decrease the error of approximation by a constant factor.

Our two algorithms can be viewed as different approaches to accelerate this slow convergence. We first describe the approach taken for the sequential algorithm presented in Chapter 2, which obtains an ϵ-approximation in $O(m \log n \log(1/\epsilon))$ time.

1.5.1 Preconditioning

Preconditioning can be viewed as a systematic way to leverage the fact that some linear systems are easier to solve than others. In our restricted setting of dividing by scalars, this may not be the case for computers. However, it is certainly true for the reader: because our number system is base 10, it is much easier to divide a number by 0.1 (aka. multiply by 10). Consider the following situation where the recurrence given above converges slowly: ℓ is between 0.01 and 0.1. Since $1 - \ell$ can be as large as 0.99, about 100 iterations are needed to reduce the error by a constant factor. However, suppose instead of computing $\ell x = b$, we try to solve $(10\ell)\,x = 10b$. The guarantees on ℓ gives that 10ℓ is between 0.1 and 1, which in turn upper bounds $1 - \ell$ by 0.9. This means 10 iterations now suffice to decrease the error by a constant factor, a tenfold speedup. This speedup comes at a cost: we need to first multiply b by 10, and instead of computing $1 - \ell$ we need to compute $1 - 10\ell$. This results in an extra division by 0.1 at each step, but in our model of base ten arithmetic with scalars, it is much less expensive than most other operations.

The role that the value of 0.1 played is known as a preconditioner, and iterative methods involving such preconditioners are known as preconditioned iterative method. The recurrence that we showed earlier, unpreconditioned iterative methods, can be viewed as using the identity, 1 as a preconditioner. From this perspective, the reason for the tenfold speedup is apparent: for the range of ℓ that we are interested in, $[0.01, 0.1]$, 0.1 is an estimate that's better by a factor of ten when compared to 1.

In general, because **A** is a matrix, the preconditioner which we denote as **B** also needs to be a matrix of the same dimension. As in the case with scalars, each step of a preconditioned iterative method needs to solve a linear system involving **B**. Also, the number of iterations depends on the quality of approximation of **B** to **A**. This leads to a pair of opposing goals for the construction of a preconditioner:

- **B** needs to be similar to **A**.

- It is easier to solve linear systems of the form $\mathbf{Bx} = \mathbf{b}'$.

To illustrate the trade-off between these two goals, it is helpful to consider the two extreme points that completely meets one of these two requirements. If $\mathbf{B} = \mathbf{A}$, then the two matrices are identical and only one iteration is required. However, solving a system in **B** is identical to solving a system in **A**. On the other hand, using the identity matrix **I** as the preconditioner makes it very easy to solve, but the number of iterations may be large. Finding the right middle point between these two conditions is a topic that's central to numerical analysis (see e.g. [Axe94]).

Many existing methods for constructing preconditioners such as Jacobi iteration or Gauss-Seidel iteration builds the preconditioner directly from the matrix **A**. In the early 90s, Vaidya proposed a paradigm shift [Vai91] for generating preconditioners:

> Since $\mathbf{A} = \mathbf{L}_G$ corresponds to some graph G, its preconditioner should be the graph Laplacian of another graph H.

In other words, one way to generate a preconditioner for **A** is to generate a graph H based on G, and let $\mathbf{B} = \mathbf{L}_H$. Similarity between $\mathbf{L}_G$ and $\mathbf{L}_H$ can then be bounded using spectral graph theory. This in turn allows one to draw connections to combinatorial graph theory and use mappings between the edges of G and H to derive bounds.

These ideas were improved by Boman and Hendrickson [BH01], and brought to full fruition in the first nearly-linear time solver by Spielman and Teng [ST06]. Our sequential solver algorithm in Chapter 2 uses this framework with a few improvements. The main idea of this framework is to find a graph H that's similar to G, but is highly tree-like. This allows H to be reduced to a much smaller graph using a limited version of Gaussian elimination. More specifically, G and H are chosen so that:

- The preconditioned iterative method requires t iterations.

- The system that H reduces to is smaller by a factor of $2t$.

This leads to a runtime recurrence of the following form:

$$T(m) = t\left(m + T\left(\frac{m}{2t}\right)\right) \tag{1.5}$$

Where the term of m inside the bracket comes from the cost of performing matrix-vector multiplications by $\mathbf{A}$ in each step, as well as the cost of reducing from H to the smaller system.

This runtime recurrence gives a total running time of $O(mt)$. The call structure corresponding to this type of recursive calls is known in the numerical analysis as a W-cycle, as illustrated in Figure 1.3

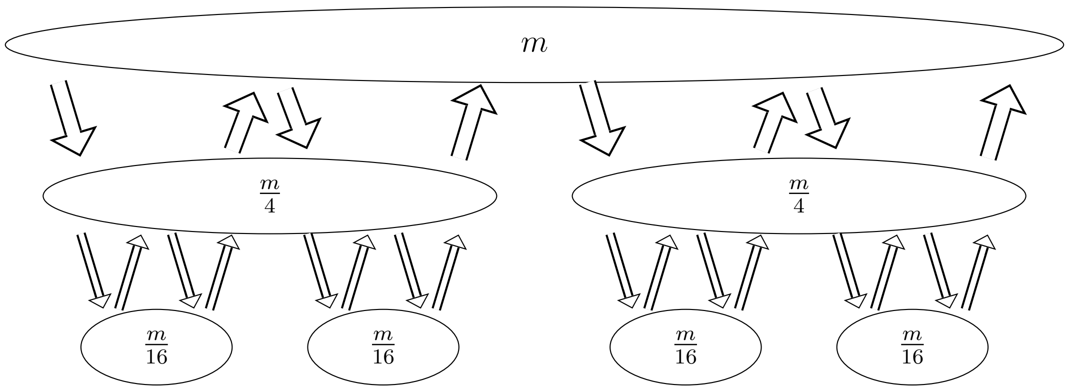


Figure 1.3: Call structure of a 3-level W-cycle algorithm corresponding to the runtime recurrence given in Equation 1.5 with $t = 2$. Each level makes 2 calls in succession to a problem that's 4 times smaller.

Analyzing the graph approximations along with this call structure leads to the main result in Chapter 2: a $O(m \log n \log(1/\epsilon))$ time algorithm for computing answers within an approximation of $1 \pm \epsilon$.

1.5.2 Factorization

Our parallel algorithm is based on a faster way of evaluating the series given in Equation 1.1. As we are solely concerned with this series, it is easier to denote $1 - \ell$ as α, by which the first t terms of this series simplifies to the following sum:

$$1 + \alpha + \alpha^2 + \ldots \alpha^{t-1}$$

Consider evaluating one term α^i. It can be done using i multiplications involving α. However, if i is a power of 2, e.g. $i = 2^k$ for some k, we can user much fewer multiplications since α^{2^k} is the square of $\alpha^{2^{k-1}}$. We can compute $\alpha^2, \alpha^4, \alpha^8, \ldots$ by repeatedly squaring the previous result, which gives us α^{2^k} in k multiplications. It is a special case of the exponentiation by squaring (see e.g. Chapter 31 of [CSRL01]), which allows us to evaluate any α^i in $O(\log i)$ multiplications.

This shortened evaluation suggest that it might be possible to evaluate the entire sum of the first t terms using a much smaller number of operations. This is indeed the case, and one way to derive such a short form is to observe the result of multiplying $1 + \alpha + \ldots \alpha^{t-1}$ by $1 + \alpha^t$:

$$\begin{aligned}\left(1 + \alpha^t\right)\left(1 + \alpha + \ldots \alpha^{t-1}\right) &= 1 + \alpha + \ldots + \alpha^{t-1} \\ &\quad + \alpha^t + \alpha^{t+1} \ldots \alpha^{2t-1} \\ &= 1 + \alpha + \ldots + \alpha^{2t-1}\end{aligned}$$

In a single multiplication, we are able to double the number of terms of the power series that we evaluate. An easy induction then gives that when $t = 2^k$, the sum of the first t terms has a succinct representation with k terms:

$$1 + \alpha + \ldots \alpha^{2^k - 1} = (1 + \alpha)\left(1 + \alpha^2\right)\left(1 + \alpha^4\right)\ldots\left(1 + \alpha^{2^{k-1}}\right)$$

This gives an approach for evaluating the first t terms of the series from Equation 1.1 in $O(\log t)$ matrix-vector multiplications. However, here our analogy between matrices and scalars breaks down: the square of a sparse matrix may be very dense. Furthermore, directly computing the square of a matrix requires matrix multiplication, for which the current best algorithm by Vassilevska-Williams [Wil12] takes about $O(n^{2.3727})$ time.

Our algorithms in Chapter 3 give a way to replace these dense matrices with sparse ones. We first show that solving any graph Laplacian can be reduced to solving a linear system $\mathbf{I} - \boldsymbol{\mathcal{A}}$ such that:

1. The power series given in Equation 1.1 converges to a constant error after a moderate (e.g. **poly** (n)) number of terms.

2. Each term in the factorized form of the power series involving $\boldsymbol{\mathcal{A}}$ has a sparse equivalent with small error.

3. The error caused by replacing these terms with their approximations accumulate in a controllable way.

The third requirement due to the fact that each step of the repeated squaring requires approximation. Our approximation of $\boldsymbol{\mathcal{A}}^4$ is based on approximating the square of our approximation to $\boldsymbol{\mathcal{A}}^2$. It is further complicated by the fact that matrices, unlike scalars, are non-commutative. Our algorithm handles these issues by a modified factorization that's able to handle the errors incurred by successive matrix approximations. This leads to a slightly more complicated algebraic form that doubles the number of matrix-vector multiplications. It multiplies by the matrix $\mathbf{I} + \boldsymbol{\mathcal{A}}$ both before handing a vector off to routines involving the approximation to $\boldsymbol{\mathcal{A}}^2$, and after receiving back an answer from it. An example of a call structure involving 3 levels is shown in Figure 1.4 . When viewed as a recursive algorithm, this is known as a V-cycle algorithm in the numerical analysis literature.

One further complication arises due to the difference between scalars and matrices. The non-commutativity of approximation matrices with original ones means that additional steps are needed to ensure symmetry of operators. When moving from terms involving $\mathbf{A}$ to $\mathbf{A}^2$, it performs matrix-vector multiplications involving $\mathbf{I} + \mathbf{A}$ both before and after evaluating the problem involving $\mathbf{A}^2$ (or more accurately its approximation).

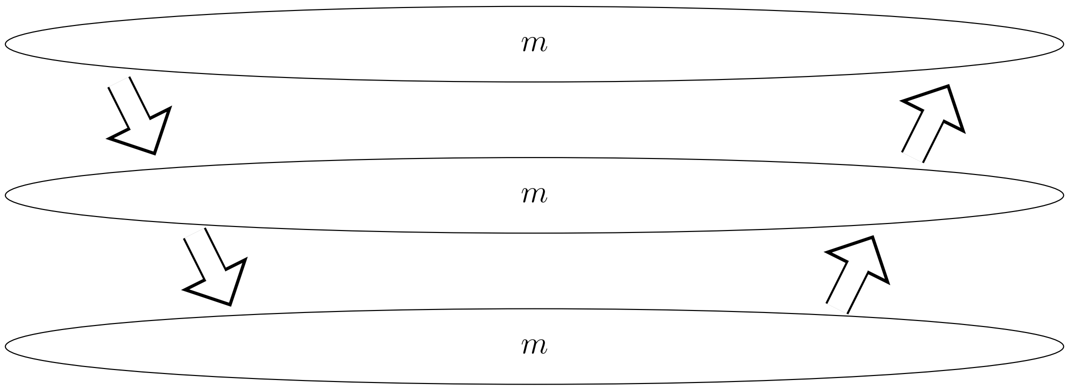


Figure 1.4: Call structure of a 3-level V-cycle algorithm. Each level makes 1 calls to a problem of comparable size. A small number of levels leads to a fast algorithm.

In terms of total number of operations performed, this algorithm is more expensive than the W-cycle algorithm shown in Chapter 2. However, a k-level V-cycle algorithm only has a $O(k)$ sequential dependency between the calls. As each call only performs matrix-vector multiplications, this algorithm is highly parallel. The V-cycle call structure is also present in the tool of choice used in numerical analysis for solving SDD linear systems: multigrid algorithms (see e.g. [BHM00]). This connection suggests that further improvements in speed, or theoretical justifications for multigrid algorithms for SDD linear systems may be possible.

1.6 Matrices and Similarity

The overview in Section 1.5 used the one-dimensional special case of scalars as illustrative examples. Measuring similarity between n-dimensional matrices and vectors is more intricate. We need to bound the convergence of power series involving matrices similar to the one involving scalars in Equation 1.1. Also, as we're working with vectors with multiple entries, converging towards a solution vector also needs to be formalized. In this section we give some background on ways of extending from scalars to matrices/vectors. All the results stated in this section are known tools for analyzing spectral approximations, and some of the proofs are deferred to Appendix A. We will distinguish between different types notationally as well: scalars will be denoted using regular lower case characters, matrices by capitalized bold symbols, and vectors by bold lower case characters.

1.6.1 Spectral Decomposition

Spectral graph theory is partly named after the study of the spectral decomposition of the graph Laplacian $\mathbf{L}_G$. Such decompositions exist for any symmetric matrix $\mathbf{A}$. It expresses $\mathbf{A}$ as a sum of orthogonal matrices corresponding to its eigenspaces. These spaces are defined by eigenvalue/vectors pairs, $(\lambda_1, \mathbf{u}_1) \dots (\lambda_n, \mathbf{u}_n)$ such that $\mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i$. Furthermore, we may assume that the vectors $\mathbf{u}_1, \mathbf{u}_2 \dots \mathbf{u}_n$ are orthonormal, that is:

$$\mathbf{u}_i^T \mathbf{u}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The decomposition of $\mathbf{A}$ can then be written as:

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

The orthogonality of the eigenspaces means that many functions involving $\mathbf{A}$ such as multiplication and inversion acts independently on the eigenspaces. When $\mathbf{A}$ is full rank, its inverse can be written as:

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

and when it's not, this decomposition allows us to define the pseudoinverse. By rearranging the eigenvalues, we may assume that the only zero eigenvalues are $\lambda_1 \dots \lambda_{n-r}$ where r is the rank of the matrix (in case of the graph Laplacian of a connected graph, $r = n - 1$). The pseudoinverse is then given by:

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

This decomposition can also be extended to polynomials. Applying a polynomial $p(x) = \sum a_0 + a_1 x^1 + a_2 x^2 + \dots + a_d x^d$ to $\mathbf{A}$ leads to the matrix $\sum a_0 + a_1 \mathbf{A} + a_2 \mathbf{A}^2 + \dots + a_d \mathbf{A}^d$. When viewed using spectral decompositions, this matrix can also be viewed as applying the polynomial to each of the eigenvalues.

Lemma 1.6.1 *If $p(x)$ is a polynomial, then:*

$$p(\boldsymbol{A}) = \sum_{i=1}^{n} p(\lambda_i) \boldsymbol{u}_i \boldsymbol{u}_i^T$$

Therefore, if $p(\lambda_i)$ approaches $\frac{1}{\lambda_i}$ for all non-zero eigenvalues λ_i, then $p(\mathbf{A})$ approaches $\mathbf{A}^{\dagger}$ as well. This is the main motivation for the analogy to scalars given in Section 1.5. It means that polynomials that work well for scalars, such as Equation 1.1, can be used on matrices whose eigenvalues satisfy the same condition.

1.6.2 Similarities Between Matrices and Algorithms

The way we treated $p(\mathbf{A})$ as a matrix is a snapshot of a powerful way for analyzing numerical algorithms. It was introduced by Spielman and Teng [ST06] to analyze the recursive preconditioning framework. In general, many algorithms acting on $\mathbf{b}$ can be viewed as applying a series of linear transformations to $\mathbf{b}$. These transformations can in turn be viewed as matrix-vector multiplications, with each operation corresponding to a matrix. For example, adding the second entry of a length 2 vector $\mathbf{b}$ to the first corresponds to evaluating:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{b}$$

and swapping the two entries of a length 2 vector $\mathbf{b}$ corresponds to evaluating:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{b}$$

A sequence of these operations can in turn be written as multiplying $\mathbf{b}$ by a series of matrices. For example, swapping the two entries of $\mathbf{b}$, then adding the second to the first can be written as:

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{b}$$

In the absence of round-off errors, matrix products are associative. As a result, we may conceptually multiply these matrices together to a single matrix, which in turn represents our entire algorithm. For example, the two above operations is equivalent to evaluating:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \mathbf{b}$$

Most of our algorithms acting on an input $\mathbf{b}$ can be viewed as symmetric linear operators on $\mathbf{b}$. Such operators can in turn be represented by (possibly dense) matrices, which we'll denote using $\mathbf{Z}$ with various subscripts. Even though such an algorithm may not explicitly compute $\mathbf{Z}$, running it with $\mathbf{b}$ as input has the same effect as computing $\mathbf{Z}\mathbf{b}$. This allows us to directly bound the spectral approximation ratios between $\mathbf{Z}$ and $\mathbf{A}^{\dagger}$.

Measuring the similarity between two linear operators represented as matrices is more complicated. The situation above with $p(\mathbf{A})$ and $\mathbf{A}^{\dagger}$ is made simpler by the fact that their spectral decompositions have the same eigenvectors. This additional structure is absent in most of our matrix-approximation bounds. However, if we are comparing $\mathbf{A}$ against the zero matrix, any spectral decomposition of $\mathbf{A}$ can also be used to give the zero matrix by setting all eigenvalues to 0. This gives a generalization of non-negativity to matrices known as positive semi-definiteness.

Definition 1.6.2 (Positive Semi-definiteness) *A matrix $\mathbf{A}$ is positive semi-definite if all its eigenvalues are non-negative.*

This definition extends to pairs of matrices $\mathbf{A}, \mathbf{B}$ in the same way as scalars. The matrix generalization of $\leq$, $\preceq$, can be defined as follows:

Definition 1.6.3 (Partial Ordering Among Matrices) *A partial ordering $\preceq$ can be defined on matrices by letting $\boldsymbol{A} \preceq \boldsymbol{B}$ if $\boldsymbol{B} - \boldsymbol{A}$ is positive semi-definite.*

We can now formally state the guarantees of the power series given in Equation 1.1 when all eigenvalues of $\mathbf{A}$ are in a good range. The following proof is a simplified version of Richardson iteration, more details on it can be found in Chapter 5.1. of [Axe94].

Lemma 1.6.4 *If $\boldsymbol{L}$ is an $n \times n$ matrix with m non-zero entries where all eigenvalues are between $\frac{1}{2}$ and $\frac{3}{2}$. Then for any $\epsilon > 0$, there is an algorithm $\textsc{Solve}_{\boldsymbol{L}}\left(\boldsymbol{b}\right)$ that under exact arithmetic is a symmetric linear operator in $\boldsymbol{b}$ and runs in $O\left(m \log\left(1/\epsilon\right)\right)$ time. The matrix realizing this operator, $\boldsymbol{Z}$, satisfies:*

$$(1 - \epsilon) \boldsymbol{L}^{\dagger} \preceq \boldsymbol{Z} \preceq (1 + \epsilon) \boldsymbol{L}^{\dagger}$$

Proof Our algorithm is to evaluate the first $N = O\left(\log\left(1/\epsilon\right)\right)$ terms of the power series from Equation 1.1. If we denote this polynomial as p_N, then we have $\mathbf{Z} = p_N\left(\mathbf{L}\right)$. Its running time follows from Horner's rule (see e.g. Chapter 2 of [CSRL01]) and the cost of making one matrix-vector multiplication in $\mathbf{I} - \mathbf{L}$.

We now show that $\mathbf{Z}$ spectrally approximates $\mathbf{L}^{\dagger}$. Let a spectral decomposition of $\mathbf{L}$ be:

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

Lemma 1.6.1 gives that applying p_N to each eigenvalue also gives a spectral decomposition to $\mathbf{Z}$:

$$\mathbf{Z} = \sum_{i=1}^{n} p_n\left(\lambda_i\right) \mathbf{u}_i \mathbf{u}_i^T$$

Furthermore, the calculations that we performed in Section 1.5, specifically Equation 1.4 gives that $p_n\left(\lambda_i\right)$ is a $1 \pm \epsilon$ approximation to $\frac{1}{\lambda_i}$:

$$(1 - \epsilon) \frac{1}{\lambda_i} \leq p_n\left(\lambda_i\right) \leq (1 + \epsilon) \frac{1}{\lambda_i}$$

This in turn implies the upper bound on $\mathbf{Z}$:

$$\begin{aligned}(1 + \epsilon) \mathbf{L}^{\dagger} - \mathbf{Z} &= \sum_{i=1}^{n} (1 + \epsilon) \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T - \sum_{i=1}^{n} p_N\left(\lambda_i\right) \mathbf{u}_i \mathbf{u}_i^T \\ &= \sum_{i=1}^{n} \left((1 + \epsilon) \frac{1}{\lambda_i} - p_N\left(\lambda_i\right) \right) \mathbf{u}_i \mathbf{u}_i^T\end{aligned}$$

This is a spectral decomposition of $(1+\epsilon)\mathbf{L}^{\dagger} - \mathbf{Z}$ with all eigenvalues non-negative, therefore $\mathbf{Z} \preceq (1+\epsilon)\mathbf{L}^{\dagger}$. The LHS also follows similarity. ■

Showing bounds of this form is our primary method of proving convergence of algorithms in this thesis. Most of our analyses are simpler than the above proof as they invoke known results on iterative methods as black-boxes. Proofs of these results on iterative methods are given in Appendix D for completeness. The manipulations involving $\preceq$ that we make mostly rely on facts that extends readily from scalars. We will make repeated use of the following two facts in our algebraic manipulations.

Fact 1.6.5 *If $\boldsymbol{A}$, $\boldsymbol{B}$ are $n \times n$ positive semi-definite matrices such that $\boldsymbol{A} \preceq \boldsymbol{B}$, then:*

1. *For any scalar $\alpha \geq 0$, $\alpha\boldsymbol{A} \preceq \alpha\boldsymbol{B}$.*
2. *If $\boldsymbol{A}'$ and $\boldsymbol{B}'$ are also $n \times n$ positive semi-definite matrices such that $\boldsymbol{A}' \preceq \boldsymbol{B}'$, then $\boldsymbol{A} + \boldsymbol{A}' \preceq \boldsymbol{B} + \boldsymbol{B}'$.*

The composition of $\preceq$ under multiplication is more delicate. Unlike scalars, $\mathbf{A} \preceq \mathbf{B}$ and $\mathbf{V}$ being positive semi-definite do not imply $\mathbf{AV} \preceq \mathbf{BV}$. Instead, we will only use compositions involving multiplications by matrices on both sides as follows:

Lemma 1.6.6 (Composition of Spectral Ordering) *For any matrices $\boldsymbol{V}, \boldsymbol{A}, \boldsymbol{B}$, if $\boldsymbol{A} \preceq \boldsymbol{B}$, then $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} \preceq \boldsymbol{V}^T\boldsymbol{B}\boldsymbol{V}$*

1.6.3 Error in Vectors

Viewing algorithms as linear operators and analyzing the corresponding matrices relies crucially on the associativity of the operations performed. This property does not hold under round-off errors: multiplying x by y and then dividing by y may result in an answer that's different than x. As a result, guarantees for numerical algorithms are often more commonly stated in terms of the distance between the output and the exact answer. Our main results stated in Section 1.3 are also in this form. Below we will formalize these measures of distance, and state the their close connections to the operator approximations mentioned above.

One of the simplest ways of measuring the distance between two vectors $\mathbf{x}$ and $\bar{\mathbf{x}}$ is by their Euclidean distance. Since distances are measured based on the difference between $\mathbf{x}$ and $\bar{\mathbf{x}}$, $\mathbf{x} - \bar{\mathbf{x}}$, we may assume $\bar{\mathbf{x}} = \mathbf{0}$ without loss of generality. The Euclidean norm of $\mathbf{x}$ is $\sqrt{\mathbf{x}^T\mathbf{x}}$, and can also be written as $\sqrt{\mathbf{x}^T\mathbf{I}\mathbf{x}}$ where $\mathbf{I}$ is the identity matrix. Replacing this matrix by $\mathbf{A}$ leads to matrix norms, which equals to $\mathbf{x}^T\mathbf{A}\mathbf{x}$.

Matrix norms are useful partly because they measure distances in ways aligned to the eigenspaces of $\mathbf{A}$. Replacing $\mathbf{A}$ by its spectral decomposition $\mathbf{A}$ gives:

$$\begin{aligned}\mathbf{x}^T\mathbf{A}\mathbf{x} &= \sum_i \lambda_i \mathbf{x}^T\mathbf{u}_i^T\mathbf{u}_i\mathbf{x} \\ &= \sum_i \lambda_i \left(\mathbf{x}^T\mathbf{u}\right)^2\end{aligned}$$

When $\lambda_i \geq 0$, this is analogous to the Euclidean norm with different weights on entries. Therefore, positive semi-definite matrices such as SDD matrices leads to matrix norms. The **A**-norm distance between two vectors **x** and **y** can then be written as:

$$\|\mathbf{x} - \bar{\mathbf{x}}\|_{\mathbf{A}} = \sqrt{(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{A} (\mathbf{x} - \bar{\mathbf{x}})}$$

When **A** is positive semi-definite, the guarantees for algorithms for solving linear systems in **A** are often stated in terms of the **A**-norm distance between **x** and $\bar{\mathbf{x}}$. We can show that the operator based guarantees described above in Section 1.6.2 is a stronger condition than small distances in the **A**-norm.

Lemma 1.6.7 *If $\boldsymbol{A}$ is a symmetric positive semi-definite matrix and $\boldsymbol{B}$ is a matrix such that:*

$$(1-\epsilon)\boldsymbol{A}^{\dagger} \preceq \boldsymbol{B} \preceq (1+\epsilon)\boldsymbol{A}^{\dagger}$$

then for any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$, $\boldsymbol{x} = \boldsymbol{B}\boldsymbol{b}$ satisfies:

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \epsilon \|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

We can also show that any solver algorithm with error < 1 can be improved into a $1 \pm \epsilon$ one by iterating on it a small number of times. A variant of Richardson iteration similar to the one used in the proof of Lemma 1.6.4 leads to the following black-box reduction to constant factor error:

Lemma 1.6.8 *If $\boldsymbol{A}$ is a positive semi-definite matrix and* $\text{SOLVE}_{\boldsymbol{A}}$ *is a routine such that for any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$,* $\text{SOLVE}_{\boldsymbol{A}}(\boldsymbol{b})$ *returns a vector $\boldsymbol{x}$ such that:*

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \frac{1}{2} \|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

Then for any ϵ, there is an algorithm $\text{SOLVE}'_{\boldsymbol{A}}$ *such that any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$,* $\text{SOLVE}'_{\boldsymbol{A}}(\boldsymbol{b})$ *returns a vector $\boldsymbol{x}$ such that:*

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \epsilon \|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

by making $O(\log(1/\epsilon))$ calls to $\text{SOLVE}'_{\boldsymbol{A}}$ *and matrix-vector multiplications to $\boldsymbol{A}$. Furthermore,* $\text{SOLVE}'_{\boldsymbol{A}}$ *is stable under round-off errors.*

This means that obtaining an algorithm whose corresponding linear operator is a constant factor approximation to the inverse suffices for a highly efficient solver. This is our main approach for analyzing the convergence of solver algorithms in Chapters 2 and 3.

1.6.4 Graph Similarity

The running times and guarantees of our algorithms rely on bounds on spectral similarities between matrices. There are several ways of generating spectrally close graphs, and a survey of them can be found in an article by Batson et al. [BSST13]. The conceptually simplest algorithm for spectral sparsification is by Spielman and Srivastava [SS08]. This algorithm follows the sampling scheme shown in Algorithm 1, which was first introduced in the construction of cut sparsifiers by Benczur and Karger [BK96]

Algorithm 1 Graph Sampling Scheme

```
SAMPLE
Input: Graph $G = (V, E, \mathbf{w})$, non-negative weight $\tilde{\boldsymbol{\tau}}_e$ for each edge, error parameter $\epsilon$.
Output: Graph $H' = (V, E', \mathbf{w}')$.
1: for $O(\sum \tilde{\boldsymbol{\tau}} \log n\epsilon^{-2})$ iterations do
2:     Pick an edge with probability proportional to its weight
3:     Add a rescaled copy of the edge to the sampled graph
4: end for
```

This algorithm can be viewed as picking a number of edges from a distribution. These samples are rescaled so that in expectation we obtain the same graph. Spielman and Srivastava [SS08] showed the that for spectral approximations, the right probability to sample edges is weight times effective resistance. Recall that the effective resistance of an edge $e = uv$ is the voltage needed to pass 1 unit of current from u to v. Formally the product of weight and effective resistance, τ_e can be defined as:

$$\boldsymbol{\tau}_e = \mathbf{w}_{uv} \boldsymbol{\chi}_{uv}^T \mathbf{L}_G^{\dagger} \boldsymbol{\chi}_{uv} \tag{1.6}$$

Where $\mathbf{L}_G^{\dagger}$ is the pseudo-inverse of $\mathbf{L}_G$ and $\boldsymbol{\chi}_{uv}$ is the vector with -1 at vertex u and 1 at vertex v. It was shown in [SS08] that if $\tilde{\boldsymbol{\tau}}$ are set close to $\boldsymbol{\tau}$, the resulting graph H is a sparsifier for G with constant probability. This proof relied on matrix-Chernoff bounds shown independently by Rudelson and Vershynin [RV07], Ahlswede and Winter [AW02], and improved by Tropp [Tro12]. Such bounds also play crucial roles in randomized numerical linear algebra algorithms, where the values τ_e described above are known as statistical leverage scores [Mah11].

Koutis et al. [KMP10] observed that if $\tilde{\boldsymbol{\tau}}$ upper bounds $\boldsymbol{\tau}$ entry-wise, H will still be a sparsifier. Subsequent improvements and simplifications of the proof have also been made by Vershynin [Ver09] and Harvey [Har11]. To our knowledge, the strongest guarantees on this sampling process in the graph setting is due to Kelner and Levin [KL11] and can be stated as follows:

Theorem 1.6.9 *For any constant c, there exist a setting of constants in the sampling scheme shown in Algorithm 1 such that for any weighted graph $G = (V, E, \boldsymbol{w})$, upper bounds $\tilde{\boldsymbol{\tau}} \geq \boldsymbol{\tau}$, and error parameter ϵ, the graph H given by the sampling process has $O\left(\sum_e \tilde{\boldsymbol{\tau}}_e \log n\epsilon^{-2}\right)$ edges and satisfies:*

$$(1-\epsilon)\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq (1+\epsilon)\boldsymbol{L}_G$$

with probability at least $1 - n^{-c}$

A proof of this theorem, as well as extensions that simplify the presentation in Chapter 2 are given in Appendix B. The (low) failure rate of this algorithm and its variants makes most of our solver algorithms probabilistic. In general, we will refer to a success probability of $1 - n^{-c}$ for arbitrary c as "high probability". As most of our algorithms rely on such sampling routines, we will often state with high probability instead of bounding probabilities explicitly.

A key consequences of this sparsification algorithm can be obtained by combining it with Foster's theorem [Fos53].

Lemma 1.6.10 *For any graph, the sum of weight times effective resistance over all edges is exactly* $n - 1$, $\sum_e \boldsymbol{\tau}_e = n - 1$.

Combining Lemma 1.6.10 with Theorem 1.6.9 gives that for any graph G and any $\epsilon > 0$, there exists a graph H with $O\left(n \log n \epsilon^{-2}\right)$ edges such that $(1 - \epsilon)\mathbf{L}_G \preceq \mathbf{L}_H \preceq (1 + \epsilon)\mathbf{L}_G$. In other words, any dense graph has a sparse spectral equivalent. Obtained spectrally similar graphs using upper bounds of these probabilities is a key part of our algorithms in Chapters 2 and 3. The need to compute such upper bounds is also the main motivation for the embedding algorithms in Chapter 4.

Chapter 2

Nearly $O(m \log n)$ Time Solver

In this chapter we present sequential algorithms for solving SDD linear systems based on combinatorial preconditioning. Our algorithms can be viewed as both a speedup and simplification of the algorithm by Spielman and Teng [ST06]. One of the key insights in this framework is that finding similar graphs with fewer edges alone is sufficient for a fast algorithm. Specifically, if the number of edges is n plus a lower order term, then the graph is 'tree like'. Repeatedly performing simple reduction steps along the tree reduces the graph to one whose size is proportional to the number of off-tree edges. This need for edge reduction will also be the main driving goal behind our presentation.

We will formalize the use of these smaller graphs, known as ultra-sparsifiers in Section 2.1, then describe our algorithms in the rest of this chapter. To simplify presentation, we first present the guarantees of our algorithm in the model where all arithmetic operations are exact. The numerical stability of the algorithm is then analyzed in the fixed point arithmetic model in Section 2.6. The main components of these algorithms appeared in two papers joint with Ioannis Koutis and Gary Miller [KMP10, KMP11], while the improvements in Section 2.5 is joint with Gary Miller and unpublished. Our result in the strongest form is as follows:

Theorem 2.0.1 *Let $\boldsymbol{L}$ be an $n \times n$ graph Laplacian with m non-zero entries. We can preprocess $\boldsymbol{L}$ in $O\left(m \log n \log \log n\right)$ time so that, with high probability, for any vector $\boldsymbol{b} = \boldsymbol{L}\bar{\boldsymbol{x}}$, we can find in $O\left(m \log n \log(1/\epsilon)\right)$ time a vector $\boldsymbol{x}$ satisfying $\left\|\boldsymbol{x} - \bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}} < \epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}}$. Furthermore, if fixed point arithmetic is used with the input given in unit lengthed words, $O(1)$ sized words suffices for the the output as well as all intermediate calculations.*

When combined with the nearly-optimal construction of ultra-sparsifiers by Kolla et al. [KMST10], our approach in Section 2.5 also implies solver algorithms that runs in $O\left(m\sqrt{\log n}\log\left(1/\epsilon\right)\right)$ time for any vector $\mathbf{b}$. We omit this extension as it requires quadratic preprocessing time using existing techniques.

2.1 Reductions in Graph Size

Our starting point is an iterative method with improved parameters. Given graphs G and H and a parameter κ such that $\mathbf{L}_G \preceq \mathbf{L}_H \preceq \kappa\mathbf{L}_G$, a recurrence similar to Equation 1.1 allows us to solve a linear system involving $\mathbf{L}_G$ to constant accuracy by solving $O(\kappa)$ ones involving $\mathbf{L}_H$. This iteration count can be improved using Chebyshev iteration to $O\left(\sqrt{\kappa}\right)$. Like Richardson iteration, it is well-known in the numerical analysis literature [Saa03, Axe94]. The guarantees of Chebyshev iteration under exact arithmetic can be stated as:

Lemma 2.1.1 (Preconditioned Chebyshev Iteration) *There exists an algorithm* PRECONCHEBY *such that for any symmetric positive semi-definite matrices* $\boldsymbol{A}$*,* $\boldsymbol{B}$*, and* κ *where*

$$\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa \boldsymbol{A}$$

any error tolerance $0 < \epsilon \leq 1/2$*,* PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *in the exact arithmetic model is a symmetric linear operator on* $\boldsymbol{b}$ *such that:*

1. *If* $\boldsymbol{Z}$ *is the matrix realizing this operator, then* $(1-\epsilon)\boldsymbol{A}^{\dagger} \preceq \boldsymbol{Z} \preceq (1+\epsilon)\boldsymbol{A}^{\dagger}$
2. *For any vector* $\boldsymbol{b}$*,* PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *takes* $N = O\left(\sqrt{\kappa} \log\left(1/\epsilon\right)\right)$ *iterations, each consisting of a matrix-vector multiplication by* $\boldsymbol{A}$*, a solve involving* $\boldsymbol{B}$*, and a constant number of vector operations.*

A proof of this Lemma is given in Appendix D.2 for completeness. Preconditioned Chebyshev iteration will serve as the main driver of the algorithms in this chapter. It allows us to transform the graph into a similar one which is sparse enough to be reduced by combinatorial transformations.

These transformations are equivalent to performing Gaussian elimination on vertices of degrees 1 and 2. In these limited settings however, this process has natural combinatorial interpretations. We begin by describing a notion of reducibility that's a combinatorial interpretations of partial Cholesky factorization. This interpretation is necessary for some of the more intricate analyses involving multiple elimination steps.

Definition 2.1.2 *A graph* G *with* m_G *edges is reducible to a graph* H *with* m_H *edges if given any solver algorithm* SOLVE$_H$ *corresponding to a symmetric linear operator on* $\boldsymbol{b}$ *such that:*

- *For any vector* $\boldsymbol{b}$*,* SOLVE$_H\left(\boldsymbol{b}\right)$ *runs in time* $\mathcal{T}$
- *The matrix realizing this linear operator,* $\boldsymbol{Z}_H$ *satisfies:* $\boldsymbol{L}_H^{\dagger} \preceq \boldsymbol{Z}_H \preceq \kappa \boldsymbol{L}_H^{\dagger}$

We can create in $O(1)$ *time a solver algorithm for* $\mathbf{L}_G$*,* SOLVE$_G$ *that is a symmetric linear operator on* $\boldsymbol{b}$ *such that:*

- *For any vector* $\boldsymbol{b}$*,* SOLVE$_G\left(\boldsymbol{b}\right)$ *runs in time* $\mathcal{T} + O\left(m_G - m_H\right)$
- *The matrix realizing this linear operator,* $\boldsymbol{Z}_G$ *satisfies:* $\boldsymbol{L}_G^{\dagger} \preceq \boldsymbol{Z}_G \preceq \kappa \boldsymbol{L}_G^{\dagger}$

We will reduce graphs using the two rules below. They follow from standard analyses of partial Cholesky factorization, and are proven in Appendix C for completeness.

Lemma 2.1.3 *Let G be a graph where vertex u has only one neighbor v. H is reducible to a graph H that's the same as G with vertex u and edge uv removed.*

Lemma 2.1.4 *Let G be a graph where vertex u has only two neighbors v_1, v_2 with edge weights $\boldsymbol{w}_{uv_1}$ and $\boldsymbol{w}_{uv_2}$ respectively. G is reducible a graph H that's the same as G except with vertex u and edges uv_1, uv_2 removed and weight*

$$\frac{\boldsymbol{w}_{uv_1}\boldsymbol{w}_{uv_2}}{\boldsymbol{w}_{uv_1}+\boldsymbol{w}_{uv_2}} = \frac{1}{\frac{1}{\boldsymbol{w}_{uv_1}}+\frac{1}{\boldsymbol{w}_{uv_2}}}$$

added to the edge between v_1 and v_2.

These two reduction steps can be applied exhaustively at a cost proportional to the size reduction. However, to simplify presentation, we will perform these reductions only along a (spanning) tree. It leads to a guarantee similar to the more aggressive version: applying these tree reductions greedily on graph with n vertices and m edges results in one with $O(m-n)$ vertices and edges. We will term this routine GREEDYELIMINATION.

Lemma 2.1.5 (Greedy Elimination) *Given a graph G along with a spanning tree T_G, there is a routine* GREEDYELIMINATION *that performs the two elimination steps in Lemmas 2.1.3 and 2.1.4 on vertices that are incident to no off-tree edges and have degree at most 2. If G has n vertices and m edges,* GREEDYELIMINATION(G, T_G) *returns in $O(m)$ time a graph H containing at most $7(m-n)$ vertices and edges along such that G is reducible to H, along with a spanning tree T_H of H formed by*

Proof Note that each of the two operations affect a constant number of vertices, and removes the corresponding low degree vertex. Therefore, one way to implement this algorithm in $O(m)$ time is to track the degrees of tree nodes, and maintain an event queue of low degree vertices.

We now bound the size of H. Let the number of off-tree edges be m', and V' be the set of vertices incident to some off tree edge, then $|V'| \leq 2m'$. The fact that none of the operations can be applied means that all vertices in $V(H) \setminus V'$ have degree at least 3. So the number of edges in H is at least:

$$\frac{3}{2}\left(n_h - |V'|\right) + m'$$

On the other hand, both the two reduction operations in Lemmas 2.1.3 and 2.1.4 preserves the fact that T is a tree. So there can be at most $n_H - 1$ tree edges in H, which means the total number of edges is at most:

$$n_H - 1 + m' \leq n_H + m'$$

Combining these two bounds gives:

$$\begin{aligned}\frac{3}{2}\left(n_H - |V'|\right) + m' &\leq n_H + m' \\ 3n_H - 3|V'| &\leq 2n_H \\ n_H &\leq 3|V'| \leq 6m'\end{aligned}$$

Since the tree in H has $n_H - 1$ edges, the number of edges can in turn be bounded by $n_H + m' \leq 7m'$. ■

Therefore significant savings in running times can be obtained if G is tree-like. This motivated the definition of ultra-sparsifiers, which are central to the solver framework by Spielman and Teng [ST06]. The following is equivalent to Definition 1.1 of [ST06], but is graph theoretic and makes the tree-like structure more explicit.

Definition 2.1.6 (Ultra-Sparsifiers) *Given a graph G with n vertices and m edges, a (κ, h)-ultra-sparsifier H is a graph on the same set of vertices such that:*

- *H consists of a spanning tree T along with $\frac{hm}{\kappa}$ off-tree edges.*
- *$\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq \kappa \boldsymbol{L}_G$.*

Spielman and Teng showed that fast algorithms for constructing (κ, h)-ultra-sparsifiers leads to solvers for SDD linear systems that run in $O\left(mh \log\left(1/\epsilon\right)\right)$ time. At a high level this is because Lemma 2.1.5 allows us to reduce solving a linear system on the ultra-sparsifier to one on a graph of size $O(\frac{hm}{\kappa})$ edges, while Lemma 2.6.3 puts the iteration count at κ. So the running time for solving a system with m edges, $\mathcal{T}(m)$ follows a recurrence of the form:

$$\mathcal{T}(m) \leq O\left(\sqrt{\kappa}\right)\left(m + \mathcal{T}\left(\frac{hm}{\kappa}\right)\right)$$

Which is optimized by setting κ to about h^2, and solves to $\mathcal{T}(m) = O\left(hm\right)$.

We will first show a construction of $(\kappa, O(\log^2 n))$-ultra-sparsifiers in Section 2.2, which by themselves imply a solver that runs in about $m^{5/4}$ time. In Section 2.3, we describe the recursive preconditioner framework from [ST06] and show that it leads to a $O(m \log^2 n \log\log n \log(1/\epsilon))$ time solver. Then we optimize this framework together with our ultra-sparsification algorithm to show an even faster algorithm in Section 2.5. This gives an algorithm for solving graph Laplacians in $O(m \log n \log(1/\epsilon))$ time.

2.2 Ultra-Sparsification Algorithm

As with the case of generating spectral sparsifiers, sampling by (upper bounds of) effective resistance is also a natural approach for generating ultra-sparsifiers. However, to date obtaining high quality estimates of effective resistances in nearly-linear time without using solvers remains an open problem. Our workaround to computing exact effective resistances relies on Rayleigh's monotonicity law.

Lemma 2.2.1 (Rayleigh's Monotonicity Law) *The effective resistance between two vertices does not decrease when we remove edges from a graph.*

This fact has a simple algebraic proof by the inequality $(\mathbf{L}_G + \mathbf{L}_{G'})^{\dagger} \preceq \mathbf{L}_G^{\dagger}$ as long as the null space on both sides are the same. It's also easy to check it in the extreme case. When edges are removed so that two vertices are disconnected, the effective resistance between them becomes infinite, which is at least any previous values.

Our use of Rayleigh's monotonicity law will avoid extreme case, but just barely. We remove as many edges as possible while keeping the graph connected. In other words, we will use resistive values w.r.t. a single tree as upper bounds for the true effective resistance τ_e. Since there is only one path connecting the end points of e in the tree, the resistance between its two end points is the sum of the resistive values along this path.

An example of this calculation is shown in Figure2.1, where the tree that we calculate the resistive value by is shown in bold.

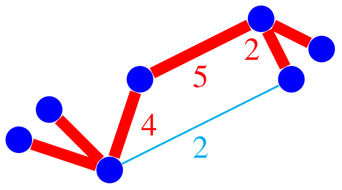


Figure 2.1: The effective resistance $R_T(e)$ of the blue off-tree edge in the red tree is $1/4 + 1/5 + 1/2 = 0.95$. Its stretch $\mathbf{str}_T(e) = \mathbf{w}_e R_T(e)$ is $(1/4 + 1/5 + 1/2)/(1/2) = 1.9$

An edge's weight times its effective resistance measured according to some tree can also be defined as its stretch [1]. This is a central notion in metric embedding literature [AKPW95] and has been extensively studied [Bar96, Bar98, FRT04, EEST08, ABN08, AN12]. As a result, we will use $\mathbf{str}_T(e)$ as an alternate notation for this product of weight and tree effective resistance. Formally, if the path between the end points of e is $\mathcal{P}_T(e)$, the stretch of edge e is:

$$\begin{aligned}\mathbf{str}_T(e) =& \mathbf{w}_e \sum_{e' \in \mathcal{P}_T(e)} r_{e'} \\ =& \mathbf{w}_e \sum_{e' \in \mathcal{P}_T(e)} \frac{1}{\mathbf{w}_{e'}}\end{aligned} \tag{2.1}$$

We can also extend this notation to sets of edges, and use $\mathbf{str}_T(E_G)$ to denote this total stretch of all edges in G. Since stretches are upper bounds for sampling probabilities, the guarantees of the sampling procedure from Theorem 1.6.9 can be stated as:

[1]Formally, it equals to stretch in the graph where the length of an edge is the reciprocal of its weight. We will formalize this in Chapter 4

Lemma 2.2.2 *There is an algorithm that takes a graph $G = (V, E_G, \boldsymbol{w})$ with n vertices and a spanning tree and returns a graph H with $n - 1 + O\left(\boldsymbol{str}_T(E_G \setminus E_T)\log n\right)$ edges such that with high probability:*

$$\frac{1}{2}\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq \frac{3}{2}\boldsymbol{L}_G$$

Note that the number of off-tree edges retained equals to $O(\log n)$ times their total stretch, $\mathbf{str}_T(E_G \setminus E_T)$. As the stretch of an edge corresponds to the probability of it being sampled, we need a tree where the total stretch to be as low as possible. Due to similar applications of a tree with small total stretch in [ST06], algorithms generating them have received significant interest [EEST08, ABN08, KMP11, AN12]. The current state of the art result is by Abraham and Neiman [AN12]:

Theorem 2.2.3 *There is an algorithm* LOWSTRETCHTREE *such that given a graph $G = (V, E, w')$,* LOWSTRETCHTREEG *outputs a spanning tree T of G satisfying $\sum_{e \in E} = O(m \log n \log\log n)$ in $O(m \log n \log\log n)$ time.*

Combining Lemma 2.2.2 and Theorem 2.2.3 as stated gives H with $O(m \log^2 n \log\log n)$ edges, which is more edges than the number of edges in G. Of course, these 'extra' edges exist in the form of multi-edges, and the edge count in the resulting graph can still be bounded by m. One way to see why direct sampling by stretch is not fruitful is to consider the case where all edges in the graph have unit weight (and therefore unit resistance). Here the stretch of an edge cannot be less than 1, which means the total estimates that we get is at least m. When combined with the guarantees of the Sampling algorithm from Theorem 1.6.9, this leads to more edges (samples to be precise) than we started with.

We remedy this issue by bringing in the parameter that we have yet to use: κ. Note that so far, the sparsifiers that we generated are off only by a constant factor from G, while the definition of ultra-sparsifiers allow for a difference of up to κ. We will incorporate this parameter in the simplest way possible: scale up the tree by a factor of κ and applying this sampling algorithm. This leads to the algorithm shown in Algorithm 2. The guarantees of this algorithm is as follows:

Algorithm 2 Ultra-sparsification Algorithm

```
ULTRASPARSIFY
Input: Graph $G$, spanning tree $T$, approximation factor $\kappa$
Output: Graph $H$
1: Let $G'$ be $G$ with $T$ scaled up by a factor of $\kappa$, and $T'$ the scaled up version of $T$
2: Calculate $\tilde{\boldsymbol{\tau}}_e = \min\{1, \mathbf{str}_{T'}(e)\}$ for all $e \notin T'$
3: return $T' + \text{SAMPLE}(E_{G'} \setminus E_{T'}, \tilde{\boldsymbol{\tau}})$
```

Theorem 2.2.4 (Ultra-Sparsifier Construction) *There exists a setting of constants in* ULTRASPARSIFY *such that given a graph G and a spanning tree T,* ULTRASPARSIFY*(G, T) returns in $O(m \log n + \frac{\textbf{\textit{str}}_T(G) \log^2 n}{\kappa})$ time a graph H that is with high probability a $\left(\kappa, O\left(\frac{\textbf{\textit{str}}_T(G)}{m} \log n\right)\right)$ -ultra-sparsifier for G.*

Proof Let G' be the graph G with T scaled up by a factor of κ. We first bound the condition number. Since the weight of an edge is increased by at most a factor of κ, we have $G \preceq G' \preceq \kappa G$.

Furthermore, because of the scaling, the effective resistance along the tree of each non-tree edge decreased by a factor of κ. Thus for any off tree edge e, we have:

$$\textbf{str}_{T'}(e) = \frac{1}{\kappa} \textbf{str}_T(e)$$

Summing over all edges gives:

$$\textbf{str}_{T'}(E_{G'} \setminus E_{T'}) \leq \frac{1}{\kappa} \textbf{str}_T(E_G \setminus E_T)$$

Using Lemma 2.2.2, we can obtain a graph H with n vertices and $n-1+\frac{1}{\kappa}O\left(\textbf{str}_T(E_G \setminus E_T) \log n\right)$ edges such that with high probability:

$$\frac{1}{2}\mathbf{L}_{G'} \preceq \mathbf{L}_H \preceq \frac{3}{2}\mathbf{L}_{G'}$$

Chaining this with the relation between G and G' gives:

$$\mathbf{L}_G \preceq \mathbf{L}_{G'} \preceq 2\mathbf{L}_H \preceq 3\mathbf{L}_{G'} \preceq 3\kappa \mathbf{L}_G$$

For the running time, we need to first computed the effective resistance of each non-tree edge by the tree. This can be done using the offline LCA algorithm [Tar79], which takes $O\left(m\alpha\left(m\right)\right)$ in the pointer machine model. This is less than the first term in our bound. To perform each step of the sampling process, we map each edge to an interval of $[0, 1]$, and find the corresponding sample in $O(\log n)$ time via. binary search. This gives the second term in our bound. ■

Using this ultra-sparsification algorithm with $\kappa = m^{1/2} \log^2 n \log\log n$ leads to a solver algorithm that runs in about $m^{5/4}$ time.

Lemma 2.2.5 *In the exact floating arithmetic model, given a graph Laplacian for G with n vertices and m edges, we can preprocess G in $O\left(m \log n \log\log n\right)$ time to obtain a solver algorithm* SOLVE$_G$ *that's a symmetric linear operator on* ***b*** *such that with high probability:*

- *For any vector* ***b***, SOLVE$_G$ (***b***) *runs in $O\left(m^{5/4} \log n \sqrt{\log\log n}\right)$ time.*
- *If* ***Z*** *is a matrix realizing this operator, we have: $\frac{1}{2}\boldsymbol{L}_G^{\dagger} \preceq \boldsymbol{Z} \preceq \frac{3}{2}\boldsymbol{L}_G^{\dagger}$*

Proof Consider the following two-level scheme: generate a (κ, h)-ultra-sparsifier H using Lemma 2.2.4, reduce its size using the greedy elimination procedure given in Lemma 2.1.5 to obtain G_1, and compute an exact inverse of the smaller system.

Since H has n vertices and $n-1+\frac{hm}{k}$ edges, the greedy elimination procedure from Lemma 2.1.5 gives that G_1 has size $O(\frac{hm}{k})$ vertices and edges. Therefore, the inverse of G_1 has size $O\left(\left(\frac{hm}{k}\right)^2\right)$ and can be computed using matrix inversion in $O\left(\left(\frac{hm}{k}\right)^\omega\right)$ time [Wil12].

Also, the condition number of κ implies that $O(\sqrt{\kappa})$ iterations are needed to obtain a solver for $\mathbf{L}_G$. Each iteration consists of a matrix-vector multiplication involving $\mathbf{L}_G$, and a solve involving $\mathbf{L}_H$. This gives a total running time of:

$$O\left(\sqrt{\kappa}\left(m + \left(\frac{hm}{\kappa}\right)^2\right) + \left(\frac{hm}{\kappa}\right)^\omega\right) \tag{2.2}$$

The inner term is optimized when $\kappa = h\sqrt{m}$, giving a total running time of:

$$O\left(h^{\frac{1}{2}} m^{\frac{5}{4}} \log(1/\epsilon) + m^{\frac{\omega}{2}}\right) \tag{2.3}$$

The current best bound of $\omega \approx 2.37$ [Wil12] gives $\frac{\omega}{2} < \frac{5}{4}$, so the last term can be discarded. Substituting in $h = O(\log^2 n \log\log n)$ gives the stated running time. ■

2.3 Recursive Preconditioning and Nearly-Linear Time Solvers

As alluded at the end of Section 2.1, the full potential of ultra-sparsifiers are realized when they are applied in a recursive fashion. A main technical issue that needs to be resolved is that each level of the recursion produces an approximate solution. This means that errors need to be propagated up the levels of the recursion. As a result, it's useful to list the systems involved in all layers of the recursive calls in the form of $G_0 \ldots G_d$. To construct level $i+1$ of this chain, we first construct a (κ_i, h)-ultra-sparsifier for G_i, H_{i+1}. Then G_{i+1} is obtained by performing greedy elimination on H_{i+1}. This structure can be formalized via. the definition of a preconditioner chain.

Definition 2.3.1 (Preconditioning Chain) *A* ***preconditoining chain*** *for a graph G is a sequence of graphs $\{G = G_0, H_1, G_1, \ldots, G_d\}$ along with size bounds $\{m_0, m_1, \ldots, m_d\}$ and spectral bounds $\{\kappa_0, \kappa_1, \ldots, \kappa_d\}$ such that:*

1. $\boldsymbol{L}_{G_i} \preceq \boldsymbol{L}_{H_{i+1}} \preceq \kappa_i \boldsymbol{L}_{G_i}$ *for all* $0 \leq i \leq d-1$ *and* $\kappa_d = 1$.
2. $G_i = \textsc{GreedyElimination}(H_i)$ *for all* $i \geq 1$.
3. *The size of G_i is at most m_i and m_d is smaller than a fixed constant.*

The extra condition of $\kappa_d = 1$ is to ease the runtime analysis, since the number of matrix-vector multiplies involving G_i also depends on κ_i. The preconditioning chain leads to recursive algorithms for solving linear systems involving G_i. Preconditioned Chebyshev iteration allows us to solve this system by making $O(\sqrt{\kappa_i})$ calls to solves involving H_i. This in turn leads to solves involving G_{i+1}. This leads to the W-cycle algorithm described in Section 1.5.1. Its performance can be described using the parameters of the preconditioning chain as follows:

Lemma 2.3.2 *Given a preconditioning chain along with any error parameter ϵ, for each i there is an algorithm* $\textsc{Solve}_{G_i}$ *such that:*

- *For any vector* $\boldsymbol{b}$*,* $\textsc{Solve}_{G_i}(\boldsymbol{b})$ *runs in time:* $O\left(\sum_{i'=i}^{d} m_{i'} \prod_{j=i}^{i'} \sqrt{c_{rec}\kappa_j}\right)$*, where* c_{rec} *is an absolute constant.*
- $\textsc{Solve}_{G_i}$ *corresponds to a symmetric linear operator on* $\boldsymbol{b}$*, and if* $\mathbf{Z}_{G_i}$ *is the matrix realizing this operator, it satisfies:* $\frac{1}{2}\boldsymbol{L}^{\dagger}_{G_i} \preceq \mathbf{Z}_{G_i} \preceq \frac{3}{2}\boldsymbol{L}^{\dagger}_{G_i}$

Proof The proof is by induction backwards on the levels i. The case where $i = d$ follows by the assumption that m_d is smaller than a constant. Suppose the result is true for $i+1$, then we have an algorithm $\textsc{Solve}_{G_{i+1}}$ that's a symmetric linear operator and its corresponding matrix $\mathbf{Z}_{G_{i+1}}$ satisfies:

$$\frac{1}{2}\mathbf{L}^{\dagger}_{G_{i+1}} \preceq \mathbf{Z}_{G_{i+1}} \preceq \frac{3}{2}\mathbf{L}^{\dagger}_{G_{i+1}}$$

As G_{i+1} was obtained by greedy elimination from H_{i+1}, H_{i+1} is reducible to G_{i+1}. Therefore we can obtain an algorithm $\textsc{Solve}_{H_{i+1}}$ corresponding to a matrix $\mathbf{Z}_{H_{i+1}}$ such that:

$$\frac{1}{2}\mathbf{L}^{\dagger}_{H_{i+1}} \preceq \mathbf{Z}_{H_{i+1}} \preceq \frac{3}{2}\mathbf{L}^{\dagger}_{H_{i+1}}$$
$$\frac{2}{3}\mathbf{L}_{H_{i+1}} \preceq \mathbf{Z}^{\dagger}_{H_{i+1}} \preceq 2\mathbf{L}_{H_{i+1}}$$

The definition of reducibility in Defintion 2.1.2 gives that computing $\mathbf{Z}_{H_{i+1}}\mathbf{b}$ for some vector $\mathbf{b}$ takes time that's $O(m_i - m_{i+1}) = O(m_i)$ more than the time of running $\textsc{Solve}_{H_{i+1}}$. Invoking the inductive hypothesis gives that $\textsc{Solve}_{H_{i+1}}$ runs in time:

$$O(m_i) + \sum_{i'=i+1}^{d} m_{i'} \prod_{j=i+1}^{i'} \sqrt{c_{rec}\kappa_j} \leq O(m_i) + \sum_{i'=i+1}^{d} m_{i'} \prod_{j=i+1}^{i'} \sqrt{c_{rec}\kappa_j}$$

As $\mathbf{L}_{H_{i+1}}$ satisfies $\mathbf{L}_{G_i} \preceq \mathbf{L}_{H_{i+1}} \preceq \kappa_i \mathbf{L}_{G_i}$, $\mathbf{Z}^{\dagger}_{H_{i+1}}$ also satisfies:

$$\frac{2}{3}\mathbf{L}_{G_i} \preceq \mathbf{Z}^{\dagger}_{H_{i+1}} \preceq 2\kappa_i \mathbf{L}_{G_i}$$

Therefore preconditioned Chebyshev iterations as given by Lemma 2.1.1 with $\mathbf{A} = \mathbf{L}_{G_i}$ and $\mathbf{B} = \frac{3}{2}\mathbf{Z}^{\dagger}_{H_{i+1}}$ gives us a routine $\textsc{Solve}_{G_i}$ meeting the required conditions. Its iteration count is

$O(\sqrt{\frac{2}{3}\kappa_i}) = O(\sqrt{\kappa_i})$. Furthermore, each solve involving $\mathbf{B}$ is equivalent to computing $\frac{2}{3}\mathbf{Z}_{H_{i+1}}\mathbf{b}'$ for some vector $\mathbf{b}'$. This is in turn equivalent to calling $\textsc{Solve}_{\mathbf{L}_{H_{i+1}}}(\mathbf{b}')$. Substituting the running time this routine gives a total running time of:

$$O\left(\sqrt{\kappa_i}\right)\left(O(m_i) + \sum_{i'=i+1}^{d} m_{i'} \prod_{j=i+1}^{i'} \sqrt{c_{rec}\kappa_j}\right)$$
$$= m_i O(\sqrt{\kappa_i}) + O(\sqrt{\kappa_i}) \cdot \sum_{i'=i+1}^{d} m_{i'} \prod_{j=i+1}^{i'} \sqrt{c_{rec}\kappa_j}$$

By choosing c_{rec} to be larger than the absolute constants and rearranging gives the total runtime. ■

This recursive framework was envisioned with ultra-sparsifiers in mind. Therefore, our ultra-sparsification algorithm from Theorem 2.2.4 can readily substituted into it. By picking a uniform κ at each level and rerunning the ultra-sparsification algorithm, we obtain the following result first shown in [KMP10].

Lemma 2.3.3 *In the exact arithmetic model, given a graph G with n vertices, m edges, and corresponding graph Laplacian $\mathbf{L}_G$. For any vector $\mathbf{b} = \mathbf{L}_G\bar{\mathbf{x}}$ and and error $\epsilon > 0$, we can find in $O(m \log^2 n \log\log n \log(1/\epsilon))$ time a vector $\mathbf{x}$ such that with high probability:*

$$\|\mathbf{x} - \bar{\mathbf{x}}\|_{\mathbf{A}} \leq \epsilon \|\bar{\mathbf{x}}\|_{\mathbf{A}}$$

Algorithm 3 Simple Preconditioning Chain Construction

```
BuildChainSimple
Input: Graph $G$.
Output: Preconditioning chain.

 1: $G_0 := G$
 2: $d = 0$
 3: Pick appropriate constants $c$ and $c_{stop}$.
 4: while $m_d > c_{stop}$ do
 5:     $T_{G_d} \leftarrow$ LowStretchTree($G_d$)
 6:     $\kappa_d \leftarrow c \log^4 n_d \log\log^2 n_d$
 7:     $H_{d+1} \leftarrow$ UltraSparsify($G_d, T_{G_d}, \kappa_d$)
 8:     $G_{d+1} \leftarrow$ GreedyElimination($H_{d+1}$)
 9:     $d \leftarrow d + 1$
10: end while
11: return $\{G_0, H_1, G_1, \ldots H_d, G_d\}, \{\kappa_0 \ldots \kappa_{d-1}, 1\}$
```

Proof We will generate a preconditioning chain by calling the ultra-sparsifier on G_i and performing greedy elimination until G_i is of constant size. Pseudocode of this algorithm is shown in Algorithm 3. Since we only made $O(\log n)$ calls to ULTRASPARSIFY, applying union bound gives that these calls all succeed with high probability. If we let c_u be the constant from the ultra-sparsification algorithm from Theorem 2.2.4 and the low-stretch spanning tree from Theorem 2.2.3, the the number of off-tree edges in H_i can be bounded by:

$$\frac{c_u \log^2 m_i \log\log m_i}{c \log^4 m_i \log\log^2 m_i} m_i = \frac{c_u}{c \log^2 m_i \log\log m_i} m_i$$

Since G_{i+1} is obtained from H_{i+1} using greedy elimination, its edge count can be bounded using Lemma 2.1.5:

$$m_{i+1} \leq \frac{7c_u}{c \log^2 m_i \log\log m_i} m_i$$

Applying this inductively gives;

$$m_i \leq m_0 \prod_{j=0}^{i-1} \left(\frac{7c_u}{c_1 \log^2 n_j \log\log n_j} \right)$$

Recall that the total work for SOLVE(G_0) given by Lemma 2.3.2 is:

$$\sum_{i=0}^{d} m_i \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j}$$

Invoking the bound on m_i on each term allows us to bound them as:

$$\begin{aligned} m_i \prod_{j=i}^{i} \sqrt{c_{rec}\kappa_j} &\leq m_0 \prod_{j=0}^{i-1} \left(\frac{7c_u}{c \log^2 n_j \log\log n_j} \right) \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j} \\ &\leq m_0 \prod_{j=0}^{i-1} \left(\frac{7c_u}{c \log^2 n_j \log\log n_j} \right) \prod_{j=0}^{i} \sqrt{c_{rec} c \log^4 n_j \log\log^2 n_j} \\ &= m_0 \sqrt{c_{rec} c \log^4 n_i \log\log^2 n_i} \prod_{j=0}^{i-1} \left(\frac{7c_u \sqrt{c_{rec} c \log^4 n_j \log\log^2 n_j}}{c \log^2 n_j \log\log n_j} \right) \\ &= m_0 \sqrt{c_{rec} c \log^4 n_i \log\log^2 n_i} \prod_{j=0}^{i-1} \left(\frac{7c_u \sqrt{c_{rec}}}{\sqrt{c}} \right) \end{aligned}$$

So if we pick c such that $c \geq 100 c_u^2 c_{rec}$, the last term can be bounded by $\left(\frac{1}{2}\right)^i$ Also, since c and c_{rec} are both constants and $n_i \leq n$, the first term can be bounded by $O(\log^2 n \log\log n)$. So the overall work can be bounded by:

$$\sum_{i=0}^{d} O(m \log^2 n \log\log n) \left(\frac{1}{2}\right)^i$$

As the terms are geometrically decreasing, this gives a total of $O(m \log^2 n \log\log n)$. The runtime needed to get to an accuracy of ϵ can then be obtained by converting error to vector form as shown in Lemma 1.6.7, and reducing error by iterating using Lemma 1.6.8 ■

2.4 Reuse, Recycle, and Reduce the Tree

We now show that a more holistic examination of the preconditioning chain leads to an even faster algorithm. The key observation is that the previous low stretch spanning tree remains as an extremely good tree in the ultrasparsifier. Here it is helpful to view both ULTRASPARSIFY and GREEDYELIMINATION as returning a tree along with a graph. We first observe that if (H, T_H) is returned by ULTRASPARSIFY(G, T_G, κ), the total stretch of H w.r.t T_H is smaller by a factor of κ, aka. $\mathbf{str}_{T_G}(G)/\kappa$.

Lemma 2.4.1 *If* $(H, T_H) = \text{TREEULTRASPARSIFY}(G, T_G, \kappa)$*, then* $\mathbf{str}_{T_H}(H) = \mathbf{str}_{T_G}(G)/\kappa$

Proof Let G'_i be the graph formed by scaling up T_G by a factor of κ in G. Let the new tree in this graph be T'_G. By definition of stretch we have $\mathbf{str}_{T_{G'}}(G') = \mathbf{str}_{T_G}(G)/\kappa$.

Now consider the effect of the sampling procedure used to obtain H from G'. Clearly the tree is unchanged, $T_H = T_{G'}$. When an edge e is picked, its weight is rescaled to its weight divided by the probability of it being sampled, which is in turn proportional to its stretch times $\log n$. So the new weight is:

$$\begin{aligned}
\mathbf{w}'_e &= \frac{\mathbf{w}_e}{p_e} \\
&= \frac{\mathbf{w}_e}{\mathbf{str}_{T_H}(e) c_s \log n} \\
&= \frac{\mathbf{w}_e}{\mathbf{w}_e R_{T_H}(e) c_s \log n} \\
&= \frac{1}{R_{T_H}(e) c_s \log n}
\end{aligned}$$

So the stretch of the edge when sampled is:

$$\begin{aligned}
\mathbf{str}_{T_H}(e') &= \mathbf{w}'_e R_{T_H}(e) \\
&= \frac{1}{c_s \log n}
\end{aligned}$$

As the total number of edges sampled is $c_s\mathbf{str}_{T_H}(G')\log n = c_s \log n\frac{\mathbf{str}_{T_G}(G)}{\kappa}$, we have $\mathbf{str}_{T_H}(H) = \frac{\mathbf{str}_{T_G}(G)}{\kappa}$. ■

The other operation that we perform on the graphs is GREEDYELIMINATION. We next show that it also does not increase the total stretch of the off-tree edges.

Lemma 2.4.2 *Let* $(G, T_G) :=$ GREEDYELIMINATION(H, H_T)*, then:*

$$\boldsymbol{str}_{T_G}(G) = \boldsymbol{str}_{T_H}(H)$$

Proof It suffices to show that for any off tree edge e, $R_{T_H}(u, v) = R_{T_G}(u, v)$. Since the path between the endpoints of e, $\mathcal{P}_{T_G}(e)$, along with e forms a cycle, none of the vertices can be removed by the degree 1 vertex removal rule given by Lemma 2.1.3. Therefore, the only case to consider is the degree 2 vertex removal rule given by Lemma 2.1.4. Here two adjacent edges e_1 and e_2 on the path are replaced by an edge with weight:

$$\frac{1}{\frac{1}{\mathbf{w}_{e_1}} + \frac{1}{\mathbf{w}_{e_2}}}$$

But the resistance of the new edge is precisely the sum of resistances of the two edges removed. So the resistance of the path remains unchanged. ■

This leads to an algorithm for building an improved chain by reusing the spanning trees from one level to the next. Its pseudocode is given in Algorithm 4.

Algorithm 4 Preconditioning chain Construction

```
BUILDCHAIN
Input: Graph $G$, first stage reduction parameter $\kappa_0$, subsequent reduction parameter $c$.
Output: Preconditioning chain.

1: Choose constant $c_{stop}$
2: $G_0 \leftarrow G$
3: $T_{G_0} \leftarrow$ LOWSTRETCHTREE$(G_0)$
4: $d = 0$
5: while $m_d > c_{stop}$ do
6:     Set $\kappa_d$ to $c$ if $d > 0$
7:     $(H_{i+1}, T_{H_{d+1}}) \leftarrow$ ULTRASPARSIFY$(G_d, T_{G_d}, \kappa_d)$
8:     $(G_{d+1}, T_{G_{d+1}}) \leftarrow$ GREEDYELIMINATION$(H_{d+1}, T_{d+1})$
9:     $d \leftarrow d + 1$
10: end while
11: return $\{G_0, H_1, G_1, \ldots H_d, G_d\}, \{\kappa_0 \ldots \kappa_{d-1}, 1\}$
```

By an argument almost analogous to the proof of the $O\left(m \log^2 n \log\log n \log(1/\epsilon)\right)$ time algorithm from Lemma 2.3.3, we can show a running time of $O(m \log n\sqrt{\log\log n}\log(1/\epsilon)$. This was the main result shown in [KMP11].

Lemma 2.4.3 *If κ_0 is set to $\log^2 n \log\log n$, and κ_i to c for a suitable constant c for all $i \geq 1$, then for any vector* ***b****,* $\textsc{Solve}_{G_0}(\boldsymbol{b})$ *runs in $O(m \log n \sqrt{\log\log n})$ time.*

Proof We first show that the sizes of m_i are geometrically decreasing. Lemmas 2.4.1 and 2.4.2 gives:

$$\mathbf{str}_{T_{G_{i+1}}}(G_{i+1}) \leq \frac{1}{\kappa_i} \mathbf{str}_{T_{G_i}}(G_i)$$

Also, the size of the ultrasparsifier produced by UltraSparsify is directly dependent on stretch. If we let c_s denote the constant in Theorem 2.2.4, we can bound m_i when $i \geq 1$ by:

$$\begin{aligned} m_i &\leq 7c_s \log n \cdot \mathbf{str}_{T_{G_{i-1}}}(G_{i-1}) \\ &\leq 7c_s \log n \cdot \mathbf{str}_{T_{G_0}}(G_0) \prod_{j=0}^{i-1} \frac{1}{\kappa_j} \end{aligned}$$

Substituting this into the runtime bound given in Lemma 2.3.2 gives that the total runtime can be bounded by:

$$\sum_{i=0}^{d} m_i \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j} \leq O(\sqrt{\kappa_0} m) + \sum_{i=1}^{d} 7c_s \log n \cdot \mathbf{str}_{T_{G_0}}(G_0) \prod_{j=0}^{i-1} \frac{1}{\kappa_j} \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j}$$

Since for all $i \geq 1$, κ_i were set to c, this becomes:

$$\begin{aligned} &= O(\sqrt{\kappa_0} m) + \sum_{i=1}^{d} 7c_s \log n \cdot \mathbf{str}_{T_{G_0}}(G_0) \frac{1}{\kappa_0} \left(\frac{1}{c}\right)^{i-1} \sqrt{c_{rec}\kappa_0} \left(\sqrt{c_{rec}c}\right)^{i} \\ &= O(\sqrt{\kappa_0} m) + \frac{7c_s c_{rec}\sqrt{c} \log n \cdot \mathbf{str}_{T_{G_0}}(G_0)}{\sqrt{\kappa_0}} \sum_{i=1}^{d} \left(\sqrt{\frac{c_{rec}}{c}}\right)^{i-1} \end{aligned}$$

An appropriate choice of constant c makes $\frac{c_{rec}}{c}$ less than 1, which in turn makes geometric series involving $\sqrt{\frac{c_{rec}}{c}}$ sum to a constant. Also, our choice of $\kappa_0 = \log^2 n \log\log n$ means both $O(\sqrt{\kappa_0} m)$ and $\frac{7c_s \log n \cdot \mathbf{str}_{T_{G_0}}(G_0)}{\sqrt{\kappa_0}}$ evaluate to $O\left(m \log n \sqrt{\log\log n}\right)$, completing the proof. ■

2.5 Tighter Bound for High Stretch Edges

The analysis of the algorithm in the previous section is not tight w.r.t. high stretch edges. This is because edges with stretch more than $\frac{1}{\log n}$ are treated as many samples when we construct our ultrasparsifiers, even though they can be moved without change from G to H. When such bounds are taken into account, we obtain the following slightly improved bound.

Lemma 2.5.1 *For any constant $\alpha > \frac{1}{2}$, there is a choice of c such that a preconditioning chain returned by* BUILDCHAIN *leads to routine* $\textsc{Solve}_{G_0}$ *with running time:*

$$O(\log n) \cdot \left(m + \sum_e \left(\frac{\textbf{\textit{str}}_T(e)}{\log n}\right)^{\alpha}\right)$$

Proof Similar to the proof of Lemma 2.4.3, the running time of $\textsc{Solve}(G_0)$ can be bounded by:

$$\sum_{i=0}^{d} m_i \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j} = \sqrt{c_{rec}\kappa_0} \cdot \left(m + \sum_{i=0}^{d} m_i \left(c_{rec}c\right)^{\frac{i}{2}}\right)$$

The difference in our analysis is that m_i is decomposed into terms per edge depending on its stretch:

$$\begin{aligned} m_i &= \sum_e \min\left\{1, \log n \textbf{str}_T(e) \prod_{j=0}^{i-1} \frac{1}{\kappa_j}\right\} \\ &= \sum_e \min\left\{1, \frac{\textbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1}\right\} \end{aligned}$$

Therefore the second term in the total cost can be rewritten as a summation per edge:

$$\sum_{i=0}^{d} m_i \prod_{j=0}^{i} \sqrt{c_{rec}\kappa_j} = \sum_e \sum_{i=1}^{d} \left(c_{rec}c\right)^{\frac{i}{2}} \min\left\{1, \frac{\textbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1}\right\}$$

Therefore it suffices to bound this sum for each edge individually. For the rest of this proof, we will consider a single edge e. Let k be the value such that $\frac{\textbf{str}_T(e)}{\log n}$ is between c^{k-1} and c^k. When $i \leq k+1$, we can upper bound the term contained in $\min\{\cdot\}$ by 1, and obtain:

$$\begin{aligned} \sum_{i=1}^{k+1} \left(c_{rec}c\right)^{\frac{i}{2}} \min\left\{1, \frac{\textbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1}\right\} &\leq \sum_{i=1}^{k+1} \left(c_{rec}c\right)^{\frac{i}{2}} \\ &= \frac{\left(c_{rec}c\right)^{\frac{k+2}{2}} - \left(c_{rec}c\right)^{\frac{1}{2}}}{\left(c_{rec}c\right)^{\frac{1}{2}} - 1} \\ &\leq \left(c_{rec}c\right)^{\frac{k+2}{2}} \qquad \text{Assuming } c_{rec}c \geq 4 \end{aligned}$$

When $i > k+1$, we have:

$$\begin{aligned}\frac{\mathbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1} &\leq c^k \left(\frac{1}{c}\right)^{i-1} \\ &= \left(\frac{1}{c}\right)^{i-k-1}\end{aligned}$$

And the summation from $i = k+1$ to d gives:

$$\begin{aligned}\sum_{i=k+1}^{d} (c_{rec}c)^{\frac{i}{2}} \min\left\{1, \frac{\mathbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1}\right\} &\leq \sum_{i=k+1}^{d} (c_{rec}c)^{\frac{i}{2}} \left(\frac{1}{c}\right)^{i-k-1} \\ &= (c_{rec}c)^{\frac{k+1}{2}} \sum_{i=0}^{d-k-1} \left(\frac{c_{rec}}{c}\right)^{i/2}\end{aligned}$$

Once again, by choosing $c > 4c_{rec}$, we have $\frac{c_{rec}}{c} \leq \frac{1}{4}$, so the geometric series can be bounded by $O(1)$. Adding both of these two terms together gives:

$$\sum_{i=1}^{d} (c_{rec}c)^{\frac{i}{2}} \min\left\{1, \frac{\mathbf{str}_T(e)}{\log n}\left(\frac{1}{c}\right)^{i-1}\right\} \leq O\left((c_{rec}c)^{\frac{k+2}{2}}\right)$$

For an appropriately large choice of c (e.g. $c \geq c_{rec}^{\frac{2}{\alpha-\frac{1}{2}}}$), we have:

$$(c_{rec}c)^{\frac{1}{2}} \leq c^{\alpha}$$

So we can in turn bound the work associated with e by:

$$\begin{aligned}O\left((c_{rec}c)^{\frac{k+2}{2}}\right) &\leq O\left(c^{(k+2)\alpha}\right) \\ &\leq O\left(\left(\frac{\mathbf{str}_T(e)}{\log n}\right)^{\alpha}\right)\end{aligned}$$

Where the last line follows from the assumption that $c^{k-1} \leq \frac{\mathbf{str}_T(e)}{\log n}$ and c being a constant. Summing over all edges then gives the total bound. ■

This lemma shows that stretch under a different norm is more closely related to runtime of recursive methods using Chebyshev iteration. In Section 4.4, specifically Claim 4.4.1, we sketch $O(m \log n \log\log n)$ time algorithms for the construction of trees where the second parameter is $O(m)$:

Claim 4.4.1 *Let α be any constant such that $0 \leq \alpha < 1$. Given weighted graph $G = (V, E, \boldsymbol{w})$,* *we can find in $O(m \log n \log\log n)$ time a spanning tree T such that:*

$$\sum_e \left(\frac{\textbf{\textit{str}}_T(e)}{\log n} \right)^\alpha \leq O(m)$$

This leads to preconditioning chains on which recursive preconditioning runs in $O(m \log n \log(1/\epsilon))$ time, and gives the result stated in Theorem 2.0.1 in the exact arithmetic model.

2.6 Stability Under Fixed Point Arithmetic

We now show that our algorithm is stable under fixed-point arithmetic. Various models of round-off errors are possible, and a discussion of such errors can be found in most books on numerical analysis (e.g. Chapter 2 of [Hig02]). The model that we work under is fixed-point, which is more restricted than the more realistic floating-point model. A number is then stored by keeping a bounded number of digits both before and after the decimal point. This amount of digits, which we assume is more than the number of digits that values of the input are given in, is also referred to as a word in the unit-cost RAM model. Round-off errors in this model occur when the lower digits are truncated after a computation. This can be viewed as perturbing the results of computations additively by ϵ_m, which is known as the machine epsilon. To simplify our analysis, we will assume such perturbations are worst-case.

Storing a fixed number of digits after the decimal point can also be viewed as keeping all numbers involved as integers while keeping a common denominator. The connection between fixed-point and integer arithmetic means that if the length of a number is W, arithmetic in numbers of length $O(W)$ can be performed in $O(1)$ time. Therefore, convergence using words of length $O(1)$ in some sense makes an algorithm stable under fixed-point arithmetic since its running time only increases by a constant factor from the exact setting.

One of the major simplifications enabled by working with fixed point arithmetic is that we can assume that our vectors are always orthogonal to the all ones vector. This is because as a post-processing step, we can set the last entry to negation of the sum of all other entries. Since the exact version of all of our intermediate vectors are also orthogonal to the all ones vector, the extra distortion caused by this can be absorbed by decreasing the machine epsilon by a factor of $\textbf{poly}(n)$.

At the bottom level, one of the main sources of round-off errors are the linear-algebraic operations: matrix-vector multiplication, scaling a vector by a constant, and adding vectors. Here it is convenient to represent this error as a vector, which we will denote using **err**. We will incorporate round-off errors by showing that $\epsilon_m < 1/\textbf{poly}(U)$ suffices for convergence. Since the sizes of all matrices and vectors are bounded by by $\textbf{poly}(n) \leq \textbf{poly}(U)$, we can instead let ϵ_m be the bound on the magnitude of **err**, aka. $\|\textbf{err}\|_2 \leq \epsilon_m$. Furthermore, we will treat all vector scaling operations as being performed with the exact value of the corresponding coefficient. This is done by computing this coefficient to an accuracy inversely proportional to the maximum entry of the vector times $\epsilon_m/\textbf{poly}(U)$. This represents a constant factor overhead in the amount of digits being tracked.

We will show that when edge weights are polynomially bounded, using $O(1)$-sized words for all intermediate steps suffice for approximate solutions. This is done in two main steps: showing that sufficiently small values of machine epsilon leads to a small cumulative error; and bounding magnitude of all intermediate values using this assumption of small cumulative error. These two steps correspond to bounding the number of digits needed after the decimal point, and in front of it respectively.

The main difficulty of the first step is the propagation of errors in our recursive algorithm. The main idea of our proof is to explicitly track all the error terms, and show that they increase by a factor of $\mathbf{poly}(\kappa_i)$ when we move from level $i+1$ to level i. This suffices because the key fact that the product of all κ_i in the preconditioning chain is $\mathbf{poly}(n)$. The second step of upper bounding the magnitude of vectors is then done by proceeding down the preconditioning chain similarly. Since our goal is to bound increases in error by $\mathbf{poly}(\kappa_i)$, we do not optimize for the exponent on κ_i in our proofs. In fact, we often leave larger exponents in order to simplify presentation.

In the exact arithmetic setting, our analysis treats solver algorithms as symmetric linear operators and analyzed their corresponding matrices. One side-effect of this is that for a fixed preconditioning chain (as defined in Definition 2.3.1), recursive preconditioning is viewed as a deterministic process. [2] The introduction of adversarial round-off errors means that running recursive preconditioning twice using the same chain on the same input may not even give the same result. As a result, our proofs are based on bounding the deviation from this exact operator. We will make use of the following definition captures both of these error parameters, as well as the running time.

Definition 2.6.1 *An algorithm* $\textsc{Solve}_{\boldsymbol{A}}$ *is an* $(\epsilon, \epsilon_1, \mathcal{T})$*-solver for a positive semi-definite matrix* $\mathbf{A}$ *if there exists a symmetric matrix* $\mathbf{Z}$ *such that:*

1. $(1-\epsilon)\boldsymbol{A}^{\dagger} \preceq \boldsymbol{Z} \preceq (1+\epsilon)\boldsymbol{A}^{\dagger}$

2. *For any vector* $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$*,* $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ *produces in time* $\mathcal{T}$ *a vector* $\boldsymbol{x}$ *such that:*

$$\left\|\boldsymbol{x} - \boldsymbol{Z}\boldsymbol{b}\right\|_{\boldsymbol{A}} \leq \epsilon_1$$

Similar to the setting of exact arithmetic, we will refer to any matrix $\mathbf{Z}$ meeting these conditions as a matrix approximating $\textsc{Solve}_{\mathbf{A}}$. When $\mathbf{b}$ is provided in a normalized form and the additive error ϵ_1 is significantly less than the minimum non-zero eigenvalue of $\mathbf{A}^{\dagger}$ (aka. the inverse of the maximum eigenvalue of $\mathbf{A}$), this also gives convergence in $\mathbf{A}$ norm.

Lemma 2.6.2 *If the maximum eigenvalue of* $\mathbf{A}$*, is at most* $\epsilon\epsilon_1^{-1}$ *and* $\textsc{Solve}_{\boldsymbol{A}}$ *is an* (ϵ, ϵ_1)*-solver for* $\mathbf{A}$*. Then for a vector* $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$ *such that* $\left\|\boldsymbol{b}\right\|_2 \geq 1$*,* $\boldsymbol{x} = \textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ *satisfies:*

$$\left\|\boldsymbol{x} - \bar{\boldsymbol{x}}\right\|_{\boldsymbol{A}} \leq 2\epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{A}}$$

[2]Note on the other hand that the preconditioning chains is generated probabilistically, making the overall algorithm probabilistic.

Proof

Let $\mathbf{Z}$ be a matrix approximating $\textsc{Solve}_{\mathbf{A}}$. The triangle inequality allows us to decompose the error into two terms:

$$\|\mathbf{x} - \bar{\mathbf{x}}\|_{\mathbf{A}} \leq \|\mathbf{x} - \mathbf{Z}\mathbf{b}\|_{\mathbf{A}} + \|\bar{\mathbf{x}} - \mathbf{Z}\mathbf{b}\|_{\mathbf{A}}$$

The first term can be bounded by ϵ_1 by the given condition. Lemma 1.6.7 allows us to bound the second term by $\epsilon \|\bar{\mathbf{x}}\|_{\mathbf{A}}$.

We now need to lower bound $\|\bar{\mathbf{x}}\|_{\mathbf{A}}$. Since $\mathbf{b}$ is in the column space of $\mathbf{A}$, $\|\bar{\mathbf{x}}\|_{\mathbf{A}} = \|\mathbf{b}\|_{\mathbf{A}^\dagger}$. The bound on the maximum eigenvalue of $\mathbf{A}$ gives that the minimum non-zero eigenvalue of $\mathbf{A}^\dagger$ is at least $\epsilon^{-1}\epsilon_1$. Combining these gives $\|\mathbf{b}\|_{\mathbf{A}^\dagger} \geq \epsilon^{-1}\epsilon_1 \|\mathbf{b}\|_2 = \epsilon^{-1}\epsilon_1$.

So both terms can be bounded in terms of relative error, giving the overall bound. ■

We will use this definition to show that round-off errors accumulate in a controllable way in recursive solvers that use the preconditioner chains given in Definition 2.3.1. There are three sources of these errors: iterative methods, greedy elimination, and matrix-vector operations. For iterative methods, a system in $\mathbf{L}_{G_i}$ is solved by solving $O(\sqrt{\kappa_i})$ systems in $\mathbf{L}_{H_{i+1}}$. In Appendix D, we show the following bound on Chebyshev iteration with round-off errors:

Lemma 2.6.3 (Preconditioned Chebyshev Iteration) *Given a positive semi-definite matrix $\boldsymbol{A}$ with m non-zero entries and all non-zero eigenvalues between $\frac{1}{\lambda}$ and λ, a positive semi-definite matrix $\boldsymbol{B}$ such that $\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa\boldsymbol{A}$, and a $(0.2, \epsilon_1, \mathcal{T})$-solver for $\boldsymbol{B}$, $\textsc{Solve}_{\boldsymbol{B}}$. If $\epsilon_m < \frac{\epsilon_1}{20\kappa\lambda}$, preconditioned Chebyshev iteration gives a routine $\textsc{Solve}_{\boldsymbol{A}}$ that is a $(0.1, O(\kappa\epsilon_1), O(\sqrt{\kappa}(m + \mathcal{T})))$-solver for $\boldsymbol{A}$. Furthermore, all intermediate values in a call $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ have magnitude at most $O(\sqrt{\lambda}\left(\|\boldsymbol{b}\|_{\boldsymbol{A}^\dagger} + \kappa\epsilon_1\right))$ and the calls to $\textsc{Solve}_{\boldsymbol{B}}$ involve vectors $\boldsymbol{b}'$ such that $\|\boldsymbol{b}'\|_{\boldsymbol{B}^\dagger} \leq \|\boldsymbol{b}\|_{\boldsymbol{A}^\dagger} + O\left(\kappa\epsilon_1\right)$.*

To the systems involving $\mathbf{L}_{H_{i+1}}$ are in turn solved by performing greedy elimination on it and recursing on the result. In Appendix C, we prove the following guarantees about the greedy elimination process. Note that the weights can fluctuate by a factor of n in a single iteration. For example a unit weight path of length n with endpoints incident to off-tree edges becomes a single edge of weight $\frac{1}{n}$. Therefore to avoid factors of $n^d = n^{O(\log n)}$, we will track the total resistance of tree edges instead.

Lemma 2.6.4 (Greedy Elimination) *Let G be a graph on n vertices with a spanning tree T_G such that the minimum resistance of a tree edge is at least $r_{\min} \leq 1$ and the total resistance is at most $r_s \geq 1$. If $\epsilon_m \leq \frac{r_{\min}}{3n^3 r_s}$, then running* $\textsc{GreedyElimination}$ *on G and T_G in $O(m)$ time to produces a graph H with a spanning tree T_H such that:*

1. *The weights of all off tree edges are unchanged.*

2. *The resistance of an edge in T_H is at least $r_{\min}$ and the total is at most $2r_s$.*

3. *If a tree edge in T_H has weight $\tilde{\boldsymbol{w}}_e$ and its weight if exact arithmetic is used would be $\bar{\boldsymbol{w}}_e$, then $\frac{1}{2}\bar{\boldsymbol{w}}_e \leq \tilde{\boldsymbol{w}}_e \leq \bar{\boldsymbol{w}}_e$.*

4. *Given a routine* $\text{SOLVE}_{\boldsymbol{L}_H}$ *that is a $(\epsilon, \epsilon_1, \mathcal{T})$-solver for $\boldsymbol{L}_H$, we can obtain a routine* $\text{SOLVE}_{\boldsymbol{L}_G}$ *that is a $(2\epsilon, 2\epsilon_1, \mathcal{T} + O(m))$-solver for $\boldsymbol{L}_G$, Furthermore, for any vector $\boldsymbol{b}$,* $\text{SOLVE}_{\boldsymbol{L}_G}$ *makes one call to* $\text{SOLVE}_{\boldsymbol{L}_H}$ *with a vector $\boldsymbol{b}'$ such that $\|\boldsymbol{b}'\|_{\boldsymbol{L}_H^\dagger} \leq \|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger} + 1$ and all intermediate values are bounded by $O(n^3(r_s + r_{\min}^{-1})(\|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger} + 1))$.*

Part 3 means that the preservation of stretch given in Lemma 2.4.2 still holds, albeit with an extra constant. It can be checked that this leads to a constant factor increase in the runtime bounds shown in Lemmas 2.4.3, and 2.5.1, assuming that the preconditioner chain is invoked in the same way. To this end, we use the two Lemmas above to show a version of Lemma 2.3.2 that incorporates round-off errors. To start an inductive argument for bounding round-off errors, we first need to show that edge weights in all layers of the solver chain are polynomially bounded. These weights are modified in three ways:

1. Scaling of the tree in ULTRASPARSIFY
2. Rescaling of sampled off-tree edges sampled in ULTRASPARSIFY.
3. Modification of edges weights by GREEDYELIMINATION.

Lemma 2.6.5 *If all edge weights in $G = G_0$ are in the range $[1, U]$ and $\epsilon_m \leq \frac{1}{3}n^{-3}U^{-1}\prod_{i=0}^{d}(2\kappa_i)^{-1}$, the requirements of Lemma C.2.5 are met in all calls to* GREEDYELIMINATION *during the construction of the solver chain, and the weights of all tree edges in all levels of the preconditioning chain can be bounded by $\left[2^{-d}n^{-2}, U\prod_{i=1}^{d-1}\kappa_i\right]$.*

Proof We show by induction down the preconditioner chain that the minimum resistance in G_i is at least $\prod_{j<i}\kappa_j^{-i}U^{-1}$, the total resistance is at most $r_s \leq 2^{i-1}n^2$, and the requirements of Lemma C.2.5 are met when creating G_i for $i \geq 1$. For the base case of $i = 0$ we have $r_{\min} \geq U^{-1}$ and $r_s \leq n^2$.

For the inductive step, note that the sampling procedure does not change the weight of tree edges. Therefore the only changes in tree weights in H_{i+1} are from scaling up the tree by a factor of at most κ_i. This means that the minimum resistance in H_{i+1} is bounded by $\prod_{j<i+1}\kappa_j^{-i}U^{-1}$, while the bound for total resistance still holds. Since $\kappa_i \geq 1$, these two conditions along with the bound on ϵ_m implies the requirement of $\epsilon_m \leq \frac{r_{\min}}{3n^3 r_s}$ by Lemma C.2.5. Its guarantees then gives that the minimum resistance in G_{i+1} is at least $\prod_{i=1}^{d}\kappa_i^{-i}U^{-1}$ while the total is at most $2^i m$. Therefore the inductive hypothesis holds for $i+1$ as well.

The lower bound for resistance gives the upper bound on edge weight. Also, the upper bound on total resistance is also an upper bound on the resistance of a single edge. Taking its inverse gives the lower bound on edge weights. ■

It remains to bound the weights of off-tree edges. The simplification to ULTRASPARSIFY of keeping instead of sampling high stretch edges ensure that the weights of samples do not decrease. The total edge weight also follows readily from the spectral guarantees obtained.

Lemma 2.6.6 *If all edge weights in $G = G_0$ are in the range $[1, U]$ and $\epsilon_m \leq \frac{1}{3}n^{-3}U^{-1}\prod_{i=0}^{d}(2\kappa_i)^{-1}$, then with high probability all edge weights in G_i are in the range $\left[2^{-d}n^{-2},\ n^2U\prod_{i=0}^{d}(2\kappa_i)\right]$.*

Proof The lower bound on edge weights follows from the lower bound of tree edge weights from Lemma 2.6.5 and the fact that weights of sampled off-tree weights do not decrease.

The upper bound for weights of tree edges is also given by Lemma 2.6.5 For the off-tree edges, their total weight which is initially bounded by mU. If H_{i+1} is an ultra-sparsifier for G_i, applying the test vector χ_u to $\mathbf{L}_G$ and $\mathbf{L}_H$ gives that the weighted degree of each vertex increases by a factor of at most κ_i. Therefore the total weight of off-tree edges increases by a factor of at most κ_i from G_i to H_{i+1} with high probability. These edge weights are then left unchanged by the greedy elimination procedure. Hence the total weight of off-tree edges in any G_i can be bounded by $n^2U\prod_{j<i}\kappa_j$ as well. ■

The recursive nature of the solver makes it possible for errors accumulated to affect the magnitude of the vectors passed to subsequent calls. Therefore, we need to start with bounding the error, or deviations in lower order bits.

Lemma 2.6.7 *If all edge weights in G are in the range $[1, U]$ and $\epsilon_m \leq n^{-3}U^{-1}\prod_{i=0}^{d}(2\kappa_i)^{-1}$, there is a constant c_{rec} such that the routine SOLVE_{G_i} as given in Lemma 2.3.2 is a $\left(0.1,\ n^2U\epsilon_m\prod_{j=0}^{i-1}\kappa_j 20\prod_{j=i}^{d}c_{rec}\kappa_j^2,\ \sum_{j=i}^{d}m_j\prod_{k=i}^{j}c_{rec}\sqrt{\kappa_k}\right)$-solver for $\mathbf{L}_{G_i}$.*

Proof The proof is by induction. For $i = d$, we can apply known results about the numerical stability of Cholesky factorizations (e.g. Theorem 10.6 in [Hig02]). Since G_d has fixed size and all edge weights in G_d fit into $O(1)$ sized words, we can compute the answer up to round-off error. Also, since the maximum edge weight of an edge in G_d is $U\prod_{j=0}^{d}\kappa_j$, the $\mathbf{L}_{G_d}$-norm of this vector is at most $20\epsilon_m\prod_{j=0}^{d}\kappa_j$

For the inductive step, assume that the inductive hypothesis holds for $i+1$. Then there is a $(0.1,\ \epsilon_1,\ \mathcal{T})$ solver for $\mathbf{L}_{G_{i+1}}$ with:

$$\epsilon_1 = U\epsilon_m\prod_{j=0}^{i}\kappa_j\prod_{j=i+1}^{d}c_{rec}\kappa_j^2$$

$$\mathcal{T} = \sum_{j=i}^{d}m_j\prod_{k=i}^{j}$$

Since G_{i+1} is obtained from H_{i+1} by greedy elimination, Lemma C.2.5 gives a routine $\text{SOLVE}_{\mathbf{L}_{H_{i+1}}}$ that is a $(0.2, 2\epsilon_1, O(m_i) + \mathcal{T})$-solver for $\mathbf{L}_{H_{i+1}}$.

It remains to bound the effect of using preconditioned Chebyshev iteration obtain a solver for $\mathbf{L}_{G_i}$ using $\textsc{Solve}_{\mathbf{L}_{H_{i+1}}}$. The bounds on edge weights in G_i given in Lemma 2.6.6 means that the non-zero eigenvalues of $\mathbf{L}_{G_{i+1}}$ are in the range $\left[2^{-d}n^{-3},\ n^2U\prod_{i=0}^{d}(2\kappa_i)\right]$ Therefore $\lambda = n^2U\prod_{i=0}^{d}(2\kappa_i)$ suffices as the eigenvalue bound needed for Lemma 2.6.3. It can also be checked that our bound on ϵ_m satisfies $\epsilon_m \leq \frac{\epsilon_1}{20\kappa\lambda}$. Since $\mathbf{L}_{G_i} \preceq \mathbf{L}_{H_{i+1}} \preceq \kappa\mathbf{L}_{G_i}$, preconditioned Chebyshev iteration as stated in Lemma 2.6.3 gives $\textsc{Solve}_{\mathbf{L}_{G_i}}$ that is $(0.1, O(\kappa_i\epsilon_1), O(\sqrt{\kappa_i}(m_i+\mathcal{T})))$-solver for $\mathbf{L}_{G_i}$. By choosing c_{rec} to be larger than any of the constants in the $O(\cdot)$, these parameters can be simplified to:

$$\begin{aligned}
c_{rec}\kappa_i\epsilon_1 &= c_{rec}\kappa_i U\epsilon_m \prod_{j=0}^{i}\kappa_j \prod_{j=i+1}^{d} c_{rec}\kappa_j^2 \\
&= c_{rec}\kappa_i U\epsilon_m \prod_{j=0}^{i-1}\kappa_j \prod_{j=i}^{d} c_{rec}\kappa_j^2 \\
c_{rec}\sqrt{\kappa_i}\,(m_i+\mathcal{T}) &= c_{rec}\sqrt{\kappa_i})\left(m_i + \sum_{j=i+1}^{d} m_j \prod_{k=i}^{j} c_{rec}\sqrt{\kappa_k}\right) \\
&= \sum_{j=i}^{d} m_j \prod_{k=i}^{j} c_{rec}\sqrt{\kappa_k}
\end{aligned}$$

Hence the inductive hypothesis holds for i as well. ■

It remains to show that none of the values encountered in the intermediate steps are too large. These values can be upper bounded using the assumption that overall error is not too large.

Lemma 2.6.8 *If all edge weights in $G = G_0$ are in the range $[1, U]$ and $\epsilon_m \leq n^{-3}U^{-1}\prod_{i=1}^{d} c_{rec}^{-1}\kappa_i^{-2}$, then on input of any vector $\boldsymbol{b}$, all intermediate values in a call to $\textsc{Solve}_{\boldsymbol{L}_{G_0}}$ can be bounded by $O\left(n^2U\left(\|\boldsymbol{b}\|_{\boldsymbol{L}_G^{\dagger}}+n\right)\prod_{i=0}^{d}c_{rec}\kappa_i\right)$*

Proof The choice of ϵ_m satisfies the requirements of of Lemmas 2.6.3 and 2.1.5. It also implies that $O(\kappa_i\epsilon_1) \leq 1$ in all the invocations of preconditioned Chebyshev iteration. Therefore the vector $\mathbf{b}$ passed to $\textsc{Solve}_{\mathbf{L}_{G_i}}$ and $\textsc{Solve}_{\mathbf{L}_{H_i}}$ all satisfy $\|\mathbf{b}\|_{\mathbf{L}_{G_i}^{\dagger}}, \|\mathbf{b}\|_{\mathbf{L}_{H_i}^{\dagger}} \leq \|\mathbf{b}\|_{\mathbf{L}_G^{\dagger}} + d \leq \|\mathbf{b}\|_{\mathbf{L}_G^{\dagger}} + n$

The bounds of intermediate values then follows from the bounds on edge weights given in Lemma 2.6.6 and Lemma 2.6.3 with $\lambda = n^2U\prod_{i=0}^{d}(2\kappa_i)$ and Lemma 2.1.5 with $r_s \leq 2^d n^2$ and $r_{\min}^{-1} \leq n^2U\prod_{i=0}^{d}(2\kappa_i)$. Also, the bounds on intermediate values of the last level follows from known results on Cholesky factorization [Hig02] and the bounds on the input vectors. ■

As $\kappa_1 = O(\log^2 n)$ and κ_i are constants for all $i \geq 2$, their product is $\mathbf{poly}\,(n)$. Substituting these values shows that the algorithm is stable when ϵ_m is bounded by $1/\mathbf{poly}\,(U)$. This allows us to bound the numerical precision of the algorithm from Theorem 2.0.1.

Proof of Theorem 2.0.1: Let the word size be $\log U$. That is, $n, m, \epsilon^{-1} \le U$, and all entries in **b** and weights in **L** are entries between 1 and U.

Note that since $\kappa_0 \le n$, $d = O(\log n)$ and κ_i are constant for all $i > 1$, $\prod_{i=0}^{d} \kappa_i$ is $\textbf{poly}(U)$. Therefore Lemma 2.6.7 gives that setting $\epsilon_m = \textbf{poly}(U^{-1})$ leads to a $O\left(0.2, 0.2U^{-2}, \sum_i^d m_i \prod_{j=i}^{d} c_{rec}\sqrt{\kappa_j}\right)$-solver to **L**.

This choice of ϵ_m also meets the requirement of Lemma 2.6.8. Hence, the maximum magnitude of a vector encountered is:

$$n^2 U \left(\|\mathbf{b}\|_{\mathbf{L}_{G_i}^{\dagger}} + n \right) \prod_{j=0}^{d} c_{rec}\kappa_j = \textbf{poly}(U) \|\mathbf{b}\|_{\mathbf{L}_G}^{\dagger}$$

Since all entries of **b** are at most U, $\|\mathbf{b}\|_2 \le \sqrt{n}U$. As the graph is connected and all weights are at least 1, the minimum eigenvalue of $\mathbf{L}_{G_i}$ is at least $\frac{1}{n^2}$. So $\|\mathbf{b}\|_{\mathbf{L}_{G_i}}^{\dagger} \le n^{3/2}U = \textbf{poly}(U)$. Hence, it suffices to keeping $O(1)$ words both before and after the decimal point.

Therefore, $O(1)$ sized words suffice and the total running time of the solver follows from Lemma 2.5.1 and Claim 4.4.1. Furthermore, the maximum eigenvalue of $\mathbf{L}_{G_0}$ can also be bounded by $nU = U^2$. Lemma 2.6.2 gives that the output **x** also satisfies: $\|\mathbf{x} - \bar{\mathbf{x}}\|_{\mathbf{L}_{G_0}} \le 0.4 \|\bar{\mathbf{x}}\|_{\mathbf{L}_{G_0}}$. This meets the requirements of the Richardson iteration based error reduction routine given in Lemma 1.6.8. Hence, iterating $O(\log(1/\epsilon))$ times gives the lower error guarantees. ■

Chapter 3

Polylog Depth, Nearly-Linear Work Solvers

The algorithm shown in Chapter 2 is a routine for solving SDD linear systems whose operation count, or running time, is close to the size of the input. Fundamental linear algebraic primitives such as matrix-vector multiplication and Gaussian elimination are inherently parallel. This means that the speed of many linear algebraic algorithms can also be increased by dividing the work among multiple processors or machines. Therefore, a natural question to ask is whether such parallel speedups are also possible for nearly-linear time SDD linear system solvers.

Since the number of processors is independent of most other parameters, parallelism can be quantified in a variety of ways. One way to measure the efficiency of a parallel algorithm is by its depth and work. The number of operations performed by the algorithm, which is equivalent to the sequential running time of the algorithm, is the **work** performed. On a single processor, this is equivalent to the sequential runtime of the algorithm. The other parameter, depth, measures the longest chain of sequential dependencies in the algorithm. This is known as the parallel time of the algorithm, and is equivalent to the running time of the algorithm when there is an unbounded number of processors. In this terminology, matrix-vector multiplication involving a matrix with m non-zero entries has $O(\log n)$ depth and $O(m)$ work; and Gaussian elimination can be parallelized to $O(\log^2 n)$ depth and $O(n^3)$ work.

Works in parallel algorithms has led to many routines that run in polylog depth, which is also the case for the two routines described above. This in part led to the definition of the complexity class $\mathcal{NC}$, which contains algorithms that run in $\textbf{poly}(\log n)$ depth and $\textbf{poly}(n)$ work. However, the parallel speedup is also limited by number of processors, which rarely approaches the size of the problem. As a result, it is often more important for parallel algorithms to be work-efficient, or nearly so. That is, the work performed by the algorithm is similar to sequential algorithms, or is within a small factor from it. A more in depth discussion of the various models of parallelism can be found in surveys such as [Ble96].

A close examination of the recursive preconditioning framework from [ST06] shows that it can be parallelized. Each recursive call performs a constant number of matrix-vector multiplications, and an appropriate setting of constants can reduce the number of calls to about $\sqrt{m}$. To our knowledge, this was first formally observed in the solver for planar graphs by Koutis and Miller [KM07].

Their algorithm runs in $O(n \log(1/\epsilon))$ work and $O(n^{1/6+\alpha} \log(1/\epsilon))$ depth [1]. Subsequently, a similar observation was made for general graphs by Blelloch et al. [BGK+13], leading to a bound of $O(m \log^{O(1)} n \log(1/\epsilon))$ work and $O(m^{1/3+\alpha} \log(1/\epsilon))$ depth. Both of these results focused on efficiently constructing the recursive solver chain in parallel, and parallelized the solver itself in a direct manner.

One additional modification was needed in these two algorithms to obtain a depth less than the more direct $\sqrt{m}$ bound mentioned above. Instead of terminating the recursion at constant sized problems, they invert the system directly whenever the cost is low enough. The higher costs of these steps are then offset by the slower growth of the recursive tree. The nearly $m^{1/3}$ depth of the Blelloch et al. algorithm [BGK+13] comes from applying Gaussian elimination. Specifically, an $n \times n$ linear system can be preprocessed in $O(\textbf{poly}(\log n))$ depth and $O(n^3)$ work to create an inverse that can be evaluated in $O(\textbf{poly}(\log n))$ depth and $O(n^2)$ work. The speedup to almost $n^{1/6}$depth by Koutis and Miller [KM07] relies on the existence of sparse representations of inverses when the graph is planar. The nested dissection algorithm by Lipton et al. [LRT79] computes in $O(\textbf{poly}(\log n))$ depth and $O(n^{3/2})$ work an inverse that can be evaluated in $O(\textbf{poly}(\log n))$ depth and $O(n \log n)$ work.

In terms of algorithmic cost, these sparse representations of inverses are essentially polylog depth, nearly-linear work solver algorithms. Empirically, the gains obtained by using them are often more pronounced. Only a small number of matrix-vector multiplications are needed to apply such an inverse, while more intricate algorithms such as recursive preconditioning have larger constant factors. As a result, sparse representations of inverses, or even dense inverses, are widely used as base cases of recursions in solver algorithms. Therefore, a sparse representation of the inverse would also be of value even if the preprocessing cost is higher.

One of the difficulties in obtaining a polylog depth, nearly-linear work algorithm is the need for an approach different than recursive preconditioning with ultra-sparsifiers. This approach obtains a nearly-linear running time by generating a chain of increasingly sparser graphs. The optimality of Chebyshev iteration means that direct parallelizations are likely to result in $\textbf{poly}(n)$ depth. Recent works by Kelner et al. [KOSZ13] and Lee and Sidford [LS13] replaced the recursive calls with data structures, leading to two-layer algorithms. However, their algorithms still use subgraphs as preconditioners, and as stated have even higher sequential dependencies of $\Omega(m)$.

The call structure of recursive preconditioning is known in numerical analysis as the W-cycle. The widely used multigrid methods on the other hand use a V-cycle call structure. The main distinction between these two call structures is that each recursive level in a W-cycle algorithms makes several calls to the next, while in the V-cycle each level makes one call to the next. If the number of levels is small, and each call only makes a small number of matrix-vector multiplications, the parallelism of a V-cycle algorithm is evident. Our main result in this chapter is a V-cycle, polylog depth, and nearly-linear work parallel solver for SDD :

Theorem 3.0.1 *Given an $n \times n$ graph Laplacian $\mathbf{L}$ with m non-zero entries such that the ratio between maximum and minimum edge weights is U. In the concurrent read, concurrent write (CRCW)*

[1]On planar graphs, m is bounded by $O(n)$

parallel RAM (PRAM) model with exact arithmetic, we can preprocess $\boldsymbol{L}$ *in* $O(\log^3(nU)\log^{7/2} n + \log^3(nU)\log\log(nU)\log^2 n)$ *depth and* $O(m(\log^4(nU)\log^{9/2} n + \log^5(nU)\log^3 n))$ *work such that with high probability for any* $\boldsymbol{b} = \boldsymbol{L}\bar{\boldsymbol{x}}$ *and* ϵ*, we can return in* $O(\log(nU)\log n \log(1/\epsilon))$ *depth and* $O(m\log^3(nU)\log n\log(1/\epsilon))$ *work a vector* $\boldsymbol{x}$ *such that* $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{L}} \leq \epsilon \|\bar{\boldsymbol{x}}\|_{\boldsymbol{L}}$.

3.1 Overview of Our Approach

Our algorithm aims to evaluate a factorization of a simplified version of the Richardson iteration. Algebraically it can also be viewed as combining fast exponentiation with sparsification, and this view is discussed in Subsection 1.5.2. On the other hand, a combinatorial interpretation is helpful to understand the interaction of our algorithm with the various approximations that we introduce in intermediate steps.

For simplicity, consider SDD matrices of the form $\mathbf{I} - \boldsymbol{\mathcal{A}}$ where $\boldsymbol{\mathcal{A}}$ has the following special structure:

1. Symmetric $\forall i, j : \boldsymbol{\mathcal{A}}_{ij} = \boldsymbol{\mathcal{A}}_{ji}$
2. All entries are positive, $\forall i, j : \boldsymbol{\mathcal{A}}_{ij} \geq 0$.
3. Each row and column sum to less than 1: $\forall i : \sum_j \boldsymbol{\mathcal{A}}_{ij} < 1$.

It's easy to check that $\mathbf{I} - \boldsymbol{\mathcal{A}}$ belongs to the class of SDD matrices. More specifically it is a graph Laplacian plus a diagonal matrix so that the diagonal entries are 1. The general version of our algorithm will work with SDDM matrices, which are graph Laplacians plus non-zero diagonal matrices. It can be shown that graph Laplacians, and in turn SDD matrices, can be transformed to SDDM matrices with similar properties. Furthermore, in Section 3.3 we will show that such matrices can be normalized to ones with algebraic properties similar to $\mathbf{I} - \boldsymbol{\mathcal{A}}$.

This special form allows us to interpret $\boldsymbol{\mathcal{A}}$ as the transition matrix of a random walk. If we are at vertex i, we will move to vertex j with probability $\boldsymbol{\mathcal{A}}_{ij}$. Therefore, if the current probabilities of being at vertices are given by the vector $\mathbf{b}$, the probabilities after one step is:

$$\boldsymbol{\mathcal{A}}\mathbf{b}$$

Since the total probability of moving to other vertices is less than 1, this walk also terminates with non-zero probability at each step. Therefore, the long-term behavior of such a walk from any starting distribution is zero. We can obtain a non-trivial long term behavior by re-injecting probabilities back into the vertices at each step. Specifically, if at each step we inject probabilities given by the vector $\mathbf{b}$ into all vertices, the stationary distribution is:

$$\mathbf{b} + \boldsymbol{\mathcal{A}}\mathbf{b} + \boldsymbol{\mathcal{A}}^2\mathbf{b}\ldots$$

Since $\boldsymbol{\mathcal{A}}^i\mathbf{b} \rightarrow \mathbf{0}$ when $i \rightarrow \infty$, this stationary distribution is well defined. Therefore, we may also let it be **x** and solve for it, giving:

$$\begin{aligned} \mathbf{x} &= \boldsymbol{\mathcal{A}}\mathbf{x} + \mathbf{b} \\ (\mathbf{I} - \boldsymbol{\mathcal{A}})\,\mathbf{x} &= \mathbf{b} \\ \mathbf{x}\,(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}\,\mathbf{b} \end{aligned}$$

Therefore, computing the stationary distribution of this random walk is equivalent to solving the SDD linear system $\mathbf{I} - \boldsymbol{\mathcal{A}}$. In other words, one way to solve the system $\mathbf{I} - \boldsymbol{\mathcal{A}}$ is to simulate the random walk given by $\boldsymbol{\mathcal{A}}$ a large number of steps. This can be viewed as a combinatorial view of the power series given in Equation 1.1, and has the same convergence properties. If the condition number, or ratio between maximum and minimum eigenvalues, of $\mathbf{I} - \boldsymbol{\mathcal{A}}$ is κ, then evaluating $O(\kappa)$ steps of the random walk suffices to give a good approximation to the answer.

One way to reduce the number of steps that needs to be evaluated is the observation that injecting **b** at all vertices every step is equivalent to injecting $\mathbf{b}_1 = \boldsymbol{\mathcal{A}}\mathbf{b} + \mathbf{b}$ every other step. In between these steps, two steps of the random walk can be taken. This is equivalent to a single step of the random walk computed by $\boldsymbol{\mathcal{A}}^2$. Therefore, the stationary distribution can also be computed by a random walk with transition given by $\boldsymbol{\mathcal{A}}_1 = \boldsymbol{\mathcal{A}}^2$ and injects $\mathbf{b}_1$ every step.

The cost of computing $\mathbf{b}_1$ is linear in the number of entries in $\boldsymbol{\mathcal{A}}$. Therefore, this allows us to reduce solving $(\mathbf{I} - \boldsymbol{\mathcal{A}})\mathbf{x} = \mathbf{b}$ to solving a system in $\mathbf{I} - \boldsymbol{\mathcal{A}}^2 = \mathbf{I} - \boldsymbol{\mathcal{A}}_1$. Because $\boldsymbol{\mathcal{A}}_1$ corresponds to the two-step transition probabilities, it also has the same properties that $\boldsymbol{\mathcal{A}}$ has: symmetric, non-negative, and all rows/columns sum to less than 1. Furthermore, we can show that the condition number of $\boldsymbol{\mathcal{A}}_1$ is half that of $\boldsymbol{\mathcal{A}}$. This means that repeating this for $O(\log \kappa)$ iterations leads to a well-conditioned matrix which can easily be solved in parallel. This is the main structure of our iterative method.

The next obstacle in this approach is that $\boldsymbol{\mathcal{A}}_1 = \boldsymbol{\mathcal{A}}^2$ could be a dense matrix Even for the star with vertices $2 \dots n$ connected to vertex 1, $\boldsymbol{\mathcal{A}}^2$ is equivalent to the complete graph. As a result, we will need to work sparse equivalents of $\boldsymbol{\mathcal{A}}_1$. Here the observation that $\boldsymbol{\mathcal{A}}^2$ also corresponds to the probability of random walks is crucial. It implies that $\mathbf{I} - \boldsymbol{\mathcal{A}}_1$ is symmetric diagonally dominant, and therefore has sparse equivalents. Solving such a sparse equivalent, e.g. $\mathbf{I} - \tilde{\boldsymbol{\mathcal{A}}}_1$ would in turn lead us to similar solutions.

Although this line of reasoning is intuitive, it has one major technical issue: the approximation $\mathbf{I} - \tilde{\boldsymbol{\mathcal{A}}}_1$ may no longer commute with $\mathbf{I} + \boldsymbol{\mathcal{A}}$. Since the input to the recursive solve is $\mathbf{b}_1 = (\mathbf{I} - \boldsymbol{\mathcal{A}})\,\mathbf{b}$, we cannot bound the error of directly replacing $(\mathbf{I} - \boldsymbol{\mathcal{A}}_1)$ with $\left(\mathbf{I} - \tilde{\boldsymbol{\mathcal{A}}}_1\right)^{-1}$. Instead, the technical components of our algorithm relies on the following different expression of $(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}$:

Fact 3.1.1

$$(\boldsymbol{I} - \boldsymbol{\mathcal{A}})^{-1} = \frac{1}{2}\left(\boldsymbol{I} + (\boldsymbol{I} + \boldsymbol{\mathcal{A}})\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1}(\boldsymbol{I} + \boldsymbol{\mathcal{A}})\right)$$

This expression can be viewed as a symmetrized version of the two-step random walk shown above. We will formalize its interaction with a chain of SDD matrices, each a sparsifier of the next, in Section 3.2. We will also show how to reduce from general SDD matrices to ones with properties similar to these random walk transition matrices. This establishes the existence of a polylog depth, nearly-linear work solver chain, or in other words as sparse inverse representation. However, as $\boldsymbol{\mathcal{A}}^2$ may be a dense $n \times n$ matrix, the construction of such solver chains may still take quadratic work.

The rest of this chapter then focuses on obtaining spectral sparsifiers efficiently in parallel. In Section 3.3, we find a crude sparsifier of $\mathbf{I} - \boldsymbol{\mathcal{A}}$ whose size is only a factor of $\log n$ larger than $\boldsymbol{\mathcal{A}}$. This allows us to invoke parallelizations of existing sparsification algorithms. and obtain a polylog depth, nearly-linear work algorithm.

We then give an alternate algorithm with better guarantees for constructing these sparsifiers in Section 3.4. It is based on reducing to smaller graphs using methods similar to the construction of graph sparsifiers by Koutis et al. [KLP12]. This approach in turn relies on low stretch subgraphs and low diameter decompositions, which we present in Chapter 4.

3.2 Parallel Solver

We now describe our solver algorithm in full detail. Instead of requiring $\boldsymbol{\mathcal{A}}$ to have all rows/columns sum to less than 1, we instead work under the condition that all its eigenvalues are strictly between -1 and 1. We first formalize the definition of our sequence of approximations to $\mathbf{I} - \boldsymbol{\mathcal{A}}^2$ as a parallel solver chain:

Definition 3.2.1 (Parallel Solver Chain) *A parallel solver chain is a sequence of matrices $\boldsymbol{\mathcal{A}}_0, \boldsymbol{\mathcal{A}}_1 \ldots \boldsymbol{\mathcal{A}}_d$ and a set of error bounds $\epsilon_0 \ldots \epsilon_{d-1} < \frac{1}{10}$ such that:*

1. *Each $\boldsymbol{\mathcal{A}}_i$ is symmetric, and have all entries non-negative.*
2. *For each $i < d$ we have $(1 - \epsilon_i)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_i^2\right) \preceq (\boldsymbol{I} - \boldsymbol{\mathcal{A}}_{i+1}) \preceq (1 + \epsilon_i)(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_i)$.*
3. *All eigenvalues of $\boldsymbol{\mathcal{A}}_d$ are in the range $\left[-\frac{2}{3}, \frac{2}{3}\right]$.*

We first show that such a parallel solver chain leads to a parallel solver algorithm with depth $O(d \cdot \mathbf{poly}(\log n))$ and work proportional to the total number of non-zero entries. Our algorithm using this chain makes critical use of the representation of $(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}$ mentioned in the overview:

Fact 3.1.1

$$(\boldsymbol{I} - \boldsymbol{\mathcal{A}})^{-1} = \frac{1}{2}\left(\boldsymbol{I} + (\boldsymbol{I} + \boldsymbol{\mathcal{A}})\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1}(\boldsymbol{I} + \boldsymbol{\mathcal{A}})\right)$$

Proof Since $\boldsymbol{\mathcal{A}}$ commutes with $\mathbf{I}$, we have:

$$\left(\mathbf{I} - \boldsymbol{\mathcal{A}}^2\right) = (\mathbf{I} - \boldsymbol{\mathcal{A}})(\mathbf{I} + \boldsymbol{\mathcal{A}})$$

inverting both sides using the fact that $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ allows us to rewrite the inverse term as:

$$\left(\mathbf{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1} = \left(\mathbf{I} + \boldsymbol{\mathcal{A}}\right)^{-1} \left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1}$$

Substituting this in the right hand side of the required identity gives:

$$\begin{aligned}\frac{1}{2}\left(\mathbf{I} + (1 + \boldsymbol{\mathcal{A}})\left(\mathbf{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1}(\mathbf{I} + \boldsymbol{\mathcal{A}})\right) &= \frac{1}{2}\left(\mathbf{I} + (\mathbf{I} + \boldsymbol{\mathcal{A}})(\mathbf{I} + \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} + \boldsymbol{\mathcal{A}})\right)\\ &= \frac{1}{2}\left(\mathbf{I} + (\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} + \boldsymbol{\mathcal{A}})\right)\end{aligned}$$

Replacing $\mathbf{I}$ with $\mathbf{I}$ with $(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} - \boldsymbol{\mathcal{A}})$ and collecting factors accordingly then gives:

$$\begin{aligned}\frac{1}{2}\left(\mathbf{I} + (\mathbf{I} + \boldsymbol{\mathcal{A}})\left(\mathbf{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1}(\mathbf{I} + \boldsymbol{\mathcal{A}})\right) &= \frac{1}{2}\left((\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} - \boldsymbol{\mathcal{A}}) + (\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}(\mathbf{I} + \boldsymbol{\mathcal{A}})\right)\\ &= (\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1}\end{aligned}$$

■

Alternatively, this can be proven by diagonalizing $\boldsymbol{\mathcal{A}}$, and prove the equation for diagonal matrices, where it in turn suffices to show it for scalars. It is algebraically identical to the proof above, but gives a more intuitive reason why identities involving scalars 1 and α also hold for $\mathbf{I}$ and $\boldsymbol{\mathcal{A}}$. The guarantees of our algorithm can be described by the following lemma:

Lemma 3.2.2 *Given a parallel solver chain with d levels where $\boldsymbol{\mathcal{A}}_i$ has $\boldsymbol{nnz}(\boldsymbol{\mathcal{A}}_i) \geq n$ non-zero entries. There is a routine $\text{SOLVE}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0}$ that's a symmetric linear operator in $\boldsymbol{b}$ such that:*

- *For any vector $\boldsymbol{b}$, $\text{SOLVE}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0}(\boldsymbol{b})$ takes $O(d \log n)$ depth and $O\left(\sum_{i=0}^{d} \boldsymbol{nnz}(\boldsymbol{\mathcal{A}}_i)\right)$ work.*
- *If $\mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0}$ is the symmetric matrix such that $\mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0}(\boldsymbol{b}) = \text{SOLVE}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0}(\boldsymbol{b})$, then: $\prod_{i=0}^{d-1}(1 + \epsilon_i)^{-1}(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_0)^{-1} \preceq \mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_0} \preceq 5\prod_{i=0}^{d-1}(1 - \epsilon_i)^{-1}(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_0)^{-1}$*

Proof The proof is by induction. For the base case of $d = 0$, all eigenvalues of $\boldsymbol{\mathcal{A}}_0$ are in the range $\left[-\frac{2}{3}, \frac{2}{3}\right]$. This in turn gives that all eigenvalues of $\mathbf{I} - \boldsymbol{\mathcal{A}}_0$ are in the range $\left[\frac{1}{3}, \frac{5}{3}\right]$ and we have:

$$\begin{aligned}\frac{1}{5}(\mathbf{I} - \boldsymbol{\mathcal{A}}_0) \preceq \frac{1}{3}\mathbf{I} \preceq (\mathbf{I} - \boldsymbol{\mathcal{A}}_0)\\ (\mathbf{I} - \boldsymbol{\mathcal{A}}_0)^{-1} \preceq 3\mathbf{I} \preceq 5(\mathbf{I} - \boldsymbol{\mathcal{A}}_0)^{-1}\end{aligned}$$

Which means that the choice of $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_0} = 3\mathbf{I}$ suffices.

For the inductive case, suppose the result is true for all parallel solver chains of length $d - 1$. Note that $\boldsymbol{\mathcal{A}}_1 \ldots \boldsymbol{\mathcal{A}}_d$ forms such a chain for $\boldsymbol{\mathcal{A}}_1$. Therefore by the inductive hypothesis we have an operator $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}$ such that:

$$\prod_{i=1}^{d-1}(1+\epsilon_i)^{-1}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_1\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1} \preceq 5\prod_{i=1}^{d-1}(1-\epsilon_i)^{-1}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_1\right)^{-1}$$

In order to construct a solver for $\mathbf{I}-\boldsymbol{\mathcal{A}}_0$, we invoke the factorization given in Fact 3.1.1, and replace $\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)^{-1}$ with $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}$. This leads to an algorithm $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_0}$ that evaluates the following linear operator:

$$\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_0} = \frac{1}{2}\left(\mathbf{I}+\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\right)$$

$\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_0}\mathbf{b}$ then requires two matrix-vector multiplications involving $\mathbf{I}+\boldsymbol{\mathcal{A}}_1$, vector operations, and one call to $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}(\mathbf{b})$. By the assumption that $\mathbf{nnz}\left(\boldsymbol{\mathcal{A}}_0\right) \geq n$, these operations can be done in $O\left(\log n\right)$ depth and $O\left(\mathbf{nnz}\left(\boldsymbol{\mathcal{A}}_0\right)\right)$ work. Incorporating the cost of running $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}$ given by the inductive hypothesis leads to the required depth/work bound.

It remains to bound the error. Here for simplicity we show the lower bound, and the upper bound follows similarly. The guarantees of $\boldsymbol{\mathcal{A}}_1$ gives

$$\begin{aligned}\mathbf{I}-\boldsymbol{\mathcal{A}}_1 &\preceq (1+\epsilon_0)\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)\\ (1+\epsilon_0)^{-1}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)^{-1} &\preceq \left(\mathbf{I}-\boldsymbol{\mathcal{A}}_1\right)^{-1}\end{aligned}$$

Combining this with the inductive hypothesis gives:

$$(1+\epsilon_0)^{-d}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}$$

The $\mathbf{I}+\boldsymbol{\mathcal{A}}_1$ terms on both sides of $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}$ allows us to compose this bound by invoking Lemma 1.6.6:

$$\prod_{i=0}^{d-1}(1+\epsilon_i)^{-1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)^{-1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right) \preceq \left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)$$

Since each $(1+\epsilon)^{-1} \leq 1$, $(1+\epsilon)^{-1}\mathbf{I} \preceq \mathbf{I}$ as well. Adding these to both sides and dividing by two then gives:

$$\prod_{i=0}^{d-1}(1+\epsilon_i)^{-1}\frac{1}{2}\left(\mathbf{I}+\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0^2\right)^{-1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\right) \preceq \frac{1}{2}\left(\mathbf{I}+\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1}\left(\mathbf{I}+\boldsymbol{\mathcal{A}}_0\right)\right)$$

By Fact 3.1.1, the LHS above is equal to $\prod_{i=0}^{d-1}(1+\epsilon_i)^{-1}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_0\right)$ while the RHS is $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_0}$. Applying a similar argument for the upper bound then gives that the inductive hypothesis holds for d as well. ■

This shows that parallel solver chains readily lead to parallel algorithms with depth similar to the depth of the solver chain. We can also show that when ϵ is sufficiently small, the spectrum of $\boldsymbol{\mathcal{A}}_i$ improves rapidly. This in turn implies that after a small number of levels, $\boldsymbol{\mathcal{A}}_i$ meets the conditions required for the last level. We first need to show that the range of eigenvalues do not change much under spectral approximations.

Lemma 3.2.3 *If $\boldsymbol{A}$ is a positive semi-definite matrix with eigenvalues in the range $[\lambda_{\min}, \lambda_{\max}]$ and $\boldsymbol{B}$ is a matrix such that $(1-\epsilon)\boldsymbol{A} \preceq \boldsymbol{B} \preceq (1+\epsilon)\boldsymbol{A}$. Then all eigenvalues of $\boldsymbol{B}$ are in the range $[(1-\epsilon)\lambda_{\min}, (1+\epsilon)\lambda_{\max}]$*

Proof The upper bound on eigenvalue implies that $\mathbf{A} \preceq \lambda_{\max}\mathbf{I}$. Therefore $\mathbf{B} \preceq (1+\epsilon)\mathbf{A} \preceq (1+\epsilon)\lambda_{\max}\mathbf{I}$, and the maximum eigenvalue of $\mathbf{B}$ can be bounded by $(1+\epsilon)\lambda_{\max}$. The bound on the minimum eigenvalue of $\mathbf{B}$ follows similarly. ■

We can now analyze the effect of descending one step down the parallel solver chain, and show that the eigenvalues of the next level improves by a constant factor.

Lemma 3.2.4 *If for some $\epsilon_i < 1/10$ we have:*

$$(1-\epsilon_i)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_i^2\right) \preceq (\boldsymbol{I} - \boldsymbol{\mathcal{A}}_{i+1}) \preceq (1+\epsilon_i)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}_i\right)$$

and $\boldsymbol{\mathcal{A}}_i$ has all eigenvalues in the range $[-1+\lambda, 1-\lambda]$: Then all eigenvalues of $\boldsymbol{\mathcal{A}}_{i+1}$ are in the range:

- $\left[-\frac{1}{10}, 1-\frac{4}{3}\lambda\right]$ *if* $\lambda \leq \frac{1}{3}$.
- $\left[-\frac{1}{10}, \frac{2}{3}\right]$ *otherwise.*

Proof Since all the eigenvalues of $\boldsymbol{\mathcal{A}}_i$ are in the range $[-1+\lambda, 1-\lambda]$. the eigenvalues of $\boldsymbol{\mathcal{A}}^2$ are in the range $[0, (1-\lambda)^2]$.

When $\lambda < \frac{1}{3}$, we can use the fact that $(1-t)^2 \leq 1 - \frac{5}{3}t$ for all $t \leq \frac{1}{3}$ to reduce the upper bound to:

$$(1-\lambda)^2 \leq 1 - \frac{5}{3}\lambda$$

Therefore all eigenvalues of $\mathbf{I} - \boldsymbol{\mathcal{A}}_i^2$ are in the range $\left[\frac{5}{3}\lambda, 1\right]$. Combining this with the approximation factor between $\mathbf{I} - \boldsymbol{\mathcal{A}}_i^2$ and $\mathbf{I} - \boldsymbol{\mathcal{A}}_{i+1}$ and Lemma 3.2.3 gives that all eigenvalues of $\mathbf{I} - \boldsymbol{\mathcal{A}}_{i+1}$ are in the range $\left[(1-\epsilon)\frac{5}{3}\lambda, 1+\epsilon\right]$. By the assumption of $\epsilon < 1/10$, this can in turn be bounded by $\left[\frac{3}{2}\lambda, \frac{11}{10}\right]$. Subtracting off the $\mathbf{I}$ then gives the bound on the eigenvalues of $\boldsymbol{\mathcal{A}}_{i+1}$.

When $\lambda \geq \frac{1}{3}$, $1-\lambda \leq \frac{2}{3}$. So the eigenvalues of $\boldsymbol{\mathcal{A}}_i^2$ are in the range $\left[0, \frac{4}{9}\right]$, and the ones for $\mathbf{I} - \boldsymbol{\mathcal{A}}_i^2$ are in the range $\left[\frac{5}{9}, 1\right]$. Incorporating approximation bounds gives that the eigenvalues of $\mathbf{I} - \boldsymbol{\mathcal{A}}_{i+1}$ are in the range $\left[(1-\epsilon)\frac{5}{9}, \frac{11}{10}\right]$. Since $\epsilon < \frac{1}{10}$, $(1-\epsilon)\frac{5}{9} \geq \frac{1}{2} > \frac{1}{3}$. So all eigenvalues of $\boldsymbol{\mathcal{A}}_{i+1}$ are between $\left[-\frac{1}{10}, \frac{2}{3}\right]$.

■

Applying this inductively gives that the depth of the chain can be bounded logarithmically in the condition number of $\boldsymbol{\mathcal{A}}$.

Corollary 3.2.5 *Given any matrix $\boldsymbol{\mathcal{A}}$ with eigenvalues in the range $\left[-1+\frac{1}{\kappa}, 1-\frac{1}{\kappa}\right]$. A parallel solver chain for $\mathbf{I} - \boldsymbol{\mathcal{A}}$ can be terminated at depth $d = O\left(\log \kappa\right)$.*

Proof Applying Lemma 3.2.4 inductively gives that when $i \geq 1$, the eigenvalues of $\boldsymbol{\mathcal{A}}_i$ are in the range $\left[-\frac{1}{10}, \max\left\{\frac{2}{3}, 1-\left(\frac{4}{3}\right)^i \frac{1}{\kappa}\right\}\right]$.

For any κ, we have that once $i > \log_{\frac{4}{3}} \kappa$ the above bound becomes $\left[-\frac{1}{10}, \frac{2}{3}\right]$. This satisfies the requirement of the last level of the parallel solver chain. So $d = O\left(\log \kappa\right)$ suffices. ■

This gives a good bound on the depth of running parallel solver chains. It also means that as long as the size of each $\boldsymbol{\mathcal{A}}_i$ is reasonably small, the total amount of work is small. We next show how to construct parallel solver chains for matrices that that are closely related to graph Laplacians.

3.3 Nearly-Linear Sized Parallel Solver Chains

In the overview in Section 3.1, the matrix $\boldsymbol{\mathcal{A}}$ corresponds to the transition probability matrix of the random walk. For a weighted graph, such a matrix can be defined by taking edges with probabilities proportional to their weights. If the adjacency matrix is $\mathbf{A}$, then the weighted degree of j is $\mathbf{d}_j = \sum_i \mathbf{A}_{ij}$ and the probability of moving from j to i is $\frac{\mathbf{A}_{ij}}{\mathbf{D}_{jj}}$. Therefore, the more general version of the random walk transition matrix is $\mathbf{A}\mathbf{D}^{-1}$ where $\mathbf{D}$ is the diagonal matrix with weighted degrees.

It is possible to analyze our algorithm using this transition matrix, but as it is not symmetric, the connection to matrix norms is not as direct. We will instead work with a more friendly form of this matrix known as the normalized Laplacian. It is obtained by a applying a similarity transform involving $\mathbf{D}^{-1/2}$, giving $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. In our analysis it is easier to keep $\mathbf{I} - \boldsymbol{\mathcal{A}}$ full rank. As a result, we also need to use a slight generalizations of graph Laplacians, SDDM and SDDM$_0$ matrices.

Definition 3.3.1 *A symmetric matrix $\mathbf{M}$ is SDDM$_0$ if:*

- *All off-diagonal entries non-positive, $\mathbf{M}_{ij} \leq 0$ for all $i \neq j$.*
- *All row/column sums non-negative, aka. $\mathbf{M}_{ii} \geq \sum_{j \neq i} -\mathbf{M}_{ij}$.*

SDDM$_0$ matrices are the subclass of SDD matrices where all off-diagonal entries are non-positive. We can further restrict ourselves to SDDM matrices, which have the additional restriction of being positive definite. Our analysis readily extends to SDDM$_0$ matrices when the null space is known. However, this restriction to full rank matrices significantly simplifies our analysis. We will reduce SDD matrices to SDDM matrices by first reducing to graph Laplacians, then increasing the diagonal slightly.

The decomposition of a graph Laplacian into diagonal minus adjacency matrix also extends to SDDM matrices. As the 2-step random walk can have self loops, we will allow $\mathbf{A}$ to have diagonal entries as well. This leads to a variety of possible ways of splitting $\mathbf{M}$ into $\mathbf{D} - \mathbf{A}$. In the numerical analysis literature, these splittings are closely related to choosing $\mathbf{D}$ as a preconditioner and are well studied (see Chapter 6 of [Axe94]). We will instead use them to normalize $\mathbf{A}$ and create $\boldsymbol{\mathcal{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. We also need to generate a sparse equivalent of the two step transition matrix:

$$\begin{aligned}\mathbf{I} - \boldsymbol{\mathcal{A}}^2 &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\\ &= \mathbf{D}^{-1/2}\left(\mathbf{D} - \mathbf{A}\mathbf{D}^{-1}\mathbf{A}\right)\mathbf{D}^{-1/2}\end{aligned}$$

The key fact about this two-step transition matrix is:

Fact 3.3.2 *If $\boldsymbol{M}$ is a SDDM matrix with splitting $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$, then $\hat{\boldsymbol{M}} = \boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}$ is also a SDDM matrix.*

Proof It is clear that $\hat{\mathbf{M}}$ is symmetric, and all entries in $\mathbf{A}\mathbf{D}^{-1}\mathbf{A}$ are non-negative. Therefore therefore it suffices to check that $\hat{\mathbf{M}}$ is strictly diagonally dominant. Let $\hat{\mathbf{A}}$ denote $\mathbf{A}\mathbf{D}^{-1}\mathbf{A}$. Since both $\mathbf{A}$ and $\mathbf{D}$ are non-negative, the entries of $\hat{\mathbf{A}}$ are also non-negative. Consider the sum of the off-diagonal entries in row i.

$$\begin{aligned}\sum_j \hat{\mathbf{A}}_{ij} &= \sum_j \left(\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\right)_{ij}\\ &= \sum_j \sum_k \mathbf{A}_{ik}\mathbf{D}_{kk}^{-1}\mathbf{A}_{kj}\\ &= \sum_k \mathbf{A}_{ik}\mathbf{D}_{kk}^{-1}\left(\sum_j \mathbf{A}_{kj}\right)\\ &\leq \sum_k \mathbf{A}_{ik}\\ &\leq \mathbf{D}_{ii}\end{aligned}$$

For the row/column where the diagonal entry is strictly dominant, the last inequality holds strictly as well. The existence of at least one such row/column in $\mathbf{M}$ means that $\hat{\mathbf{M}}$ is also positive definite. ■

Due to the close connection between $\hat{\mathbf{M}}$ and graph Laplacians, it has a sparse equivalent. Furthermore, we can show that a (scaled) version of this sparse equivalent also has a splitting that involves $\mathbf{D}$.

Lemma 3.3.3 *If $\hat{\boldsymbol{M}}$ is a SDDM matrix with a splitting into $\boldsymbol{D} - \hat{\boldsymbol{A}}$ and $\tilde{\boldsymbol{M}}$ is another SDDM matrix such that $(1-\epsilon)\hat{\boldsymbol{M}} \preceq \tilde{\boldsymbol{M}} \preceq \hat{\boldsymbol{M}}$. Then $\tilde{\boldsymbol{M}}$ has a splitting into $\boldsymbol{D} - \tilde{\boldsymbol{A}}$ such that all the entries of $\tilde{\boldsymbol{A}}$ are non-negative. Furthermore, if $\boldsymbol{I} - \hat{\boldsymbol{\mathcal{A}}} = \boldsymbol{D}^{-1/2}\hat{\boldsymbol{M}}\boldsymbol{D}^{-1/2}$ and $\boldsymbol{I} - \tilde{\boldsymbol{\mathcal{A}}} = \boldsymbol{D}^{-1/2}\tilde{\boldsymbol{M}}\boldsymbol{D}^{-1/2}$ are normalizations corresponding to these splittings, then $(1-\epsilon)(\boldsymbol{I} - \hat{\boldsymbol{\mathcal{A}}}) \preceq (\boldsymbol{I} - \tilde{\boldsymbol{\mathcal{A}}}) \preceq \boldsymbol{I} - \hat{\boldsymbol{\mathcal{A}}}$.*

Proof

Let $\tilde{\mathbf{D}}$ be the diagonal of $\tilde{\mathbf{M}}$. Since $\tilde{\mathbf{M}} \preceq \hat{\mathbf{M}}$, we have $\chi_i^T \tilde{\mathbf{M}} \chi_i \leq \chi_i^T \hat{\mathbf{M}} \chi_i$ for all indicator vectors χ_i. Also, as $\hat{\mathbf{A}} \geq 0$, $\hat{\mathbf{M}}_{ii} \leq \mathbf{D}_{ii}$ and $\tilde{\mathbf{D}} = \tilde{\mathbf{M}}_{ii} \leq \mathbf{D}_{ii}$. Therefore, $\mathbf{D} - \tilde{\mathbf{D}}$ is a non-negative diagonal matrix, so adding it to the negated off-diagonal entries of $\tilde{\mathbf{M}}$, $\tilde{\mathbf{D}} - \tilde{\mathbf{M}}$, gives the $\tilde{\mathbf{A}}$ that meets our requirements. Applying composition of spectral bounds from Lemma 1.6.6 with $\mathbf{V} = \mathbf{D}^{1/2}$ then gives:

$$(1-\epsilon)\, \mathbf{D}^{-1/2} \hat{\mathbf{M}} \mathbf{D}^{-1/2} \preceq \mathbf{D}^{-1/2} \tilde{\mathbf{M}} \mathbf{D}^{-1/2} \preceq \mathbf{D}^{-1/2} \hat{\mathbf{M}} \mathbf{D}^{-1/2}$$
$$(1-\epsilon) \left(\mathbf{I} - \hat{\boldsymbol{\mathcal{A}}}\right) \preceq \mathbf{I} - \tilde{\boldsymbol{\mathcal{A}}} \preceq \mathbf{I} - \hat{\boldsymbol{\mathcal{A}}}$$

■

Therefore, we can focus on constructing a sparse equivalent of $\hat{\mathbf{M}}$. To do so, we need to give parallel algorithms for spectral sparsification. However, some of the existing methods, such as sparsification by effective resistance require access to a solver. Using such methods would require recursing on a smaller graph (while incurring errors), leading to a more intricate algorithm that we will show in Section 3.4. Using combinatorial methods leads to higher polylog factors, but a conceptually simpler overall algorithm. It can be checked that the spectral sparsification algorithm by Spielman and Teng [ST08], as well as subsequent improvements by Andersen et al. [ACL06] and Andersen et al. [AP09] can run in polylog depth and nearly-linear work. The approach of combining spanners of random subgraphs by Kapralov et al. [KP12] also appears to be parallelizable. As our final bounds depend on a different approach with a smaller depth/work bound, we will state these results without explicitly stating the exponent on $\log n$ and use $\log^{O(1)} n$ instead.

Lemma 3.3.4 *Given an $n \times n$ SDDM matrix $\boldsymbol{M}$ with m non-zero entries and error parameter ϵ, we can find in $O\left(\log^{O(1)} n\right)$ depth and $O\left(m \log^{O(1)} n\right)$ work a SDDM matrix $\tilde{\boldsymbol{M}}$ with $O\left(n \log^{O(1)} n \epsilon^{-2}\right)$ edges such that with high probability we have:*

$$(1-\epsilon)\boldsymbol{M} \preceq \tilde{\boldsymbol{M}} \preceq (1+\epsilon)\boldsymbol{M}$$

However, we cannot directly apply this Lemma to the graph Laplacian obtained from $\hat{\mathbf{M}}$ since it may have quadratic size. For example, if $\mathbf{A}$ corresponds to a star graph with all vertices connected to vertex 1, $\mathbf{A}\mathbf{D}^{-1}\mathbf{A}$ will have all pairs of vertices connected since they can reach each other via. vertex 1. As a result, we need to first generate a sparse matrix that's similar to $\hat{\mathbf{M}}$. We do so by noting that $\hat{\mathbf{M}}$ is the sum of a collection of complete graphs weighted by vertices.

$$\mathbf{A}\mathbf{D}^{-1}\mathbf{A} = \sum_i \mathbf{D}_{ii}^{-1} \mathbf{A}_{i*}^T \mathbf{A}_{i*} \tag{3.1}$$

This along with Fact 3.3.2 allows us to write $\hat{\mathbf{M}}$ as $\sum_i \mathbf{L}_i + \mathbf{D}'$ where $\mathbf{L}_i$ corresponds to the graph with edge weights given by $\mathbf{D}_{ii}^{-1} \mathbf{A}_{i*}^T \mathbf{A}_{i*}$ and $\mathbf{D}'$ is a positive diagonal matrix. We can also calculate

$\mathbf{D}'$ explicitly in ways similar to the proof of Fact 3.3.2.

Although each $\mathbf{L}_i$ can be dense, the number of vertices involved in it is still bounded by the degree of vertex i. Therefore it suffices to efficiently generate spectral sparsifiers for $\mathbf{L}_i$. We will do so using the sparsification routine given in Theorem 1.6.9. However, we also cannot compute all edges and need to sample the edges implicitly from the distribution. Therefore, we need closed forms for the weights and effective resistances of edges in $\mathbf{L}_i$.

Lemma 3.3.5 *For two neighbors of i, j and k, the weight of the edge between j and k ($j \neq k$) in $\boldsymbol{L}_i$ is:*

$$\frac{\boldsymbol{A}_{ij}\boldsymbol{A}_{ik}}{\boldsymbol{D}_{ii}}$$

and the effective resistance between j and k in $\boldsymbol{L}_i$ is:

$$\frac{\boldsymbol{D}_{ii}}{\boldsymbol{d}_i}\left(\frac{1}{\boldsymbol{A}_{ij}}+\frac{1}{\boldsymbol{A}_{ik}}\right)$$

Where $\boldsymbol{d}_i$ is the total weight of edges incident to i, $\sum_{j\neq i}\boldsymbol{A}_{ij}$.

Proof The edge weights follows from the corresponding term in $\mathbf{A}_{i*}^T\mathbf{A}_{i*}$. It can be checked that when $j \neq k$, the vector:

$$\mathbf{x}_l = \begin{cases} 0 & \text{if } l \neq j,k \\ \frac{\mathbf{D}_{ii}}{\mathbf{A}_{ij}\mathbf{d}_i} & \text{if } l = j \\ -\frac{\mathbf{D}_{ii}}{\mathbf{A}_{ik}\mathbf{d}_i} & \text{if } l = k \end{cases}$$

Is a solution to $\mathbf{L}_i\mathbf{x} = \mathbf{e}_{jk}$. As $\mathbf{e}_{jk}$ is orthogonal to the null space of $\mathbf{L}_i$, it suffices to compute the result using this vector:

$$\begin{aligned}\mathbf{e}_{jk}^T\mathbf{L}_i^+\mathbf{e}_{jk} &= \mathbf{e}_{jk}^T\mathbf{x} \\ &= \frac{\mathbf{D}_{ii}}{\mathbf{d}_i}\left(\frac{1}{\mathbf{A}_{ij}}+\frac{1}{\mathbf{A}_{ik}}\right)\end{aligned}$$

■

Therefore we can sample this star graph implicitly using sparsification by effective resistance as given in Theorem 1.6.9. This leads a sparsifier for $\mathbf{L}_i$ whose size is $O(\log n\epsilon^{-2})$ times the number of neighbors of vertex i. Combining these forms the overall sparsifier.

Lemma 3.3.6 *Given a SDDM matrix $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$ corresponding to a graph with n vertices and m edges, along with any parameter ϵ. Let $\hat{\boldsymbol{M}} = \boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}$. We can construct in $O(\log n)$ depth $O\left(m\log n^2\epsilon^{-2}\right)$ work a SDDM matrix $\tilde{\boldsymbol{M}}$ corresponding to a graph on n vertices and*

$O(m \log n\epsilon^{-2})$ edges such that:

$$(1-\epsilon)\tilde{\boldsymbol{M}} \preceq \hat{\boldsymbol{M}} \preceq (1+\epsilon)\tilde{\boldsymbol{M}}$$

Proof We will apply the sparsification procedure given in Theorem 1.6.9 to each $\mathbf{L}_i$ separately. Let the degrees of the vertices be $n_1 \dots n_n$ respectively. Lemma 3.3.5 gives that the sampling probability of an edge jk in $\mathbf{L}_i$ is:

$$\left(\frac{\mathbf{A}_{ij}\mathbf{A}_{ik}}{\mathbf{D}_{ii}}\right) \cdot \frac{\mathbf{D}_{ii}}{\mathbf{d}_i}\left(\frac{1}{\mathbf{A}_{ij}} + \frac{1}{\mathbf{A}_{ik}}\right) = \frac{\mathbf{A}_{ij} + \mathbf{A}_{ik}}{\mathbf{d}_i}$$

To sample this distribution, we can compute the total probability incident to each vertex j since the only term to remove is $k = j$. Sampling this distribution gives j, and another sampling step then gives us k. Therefore, we can samples the edges of $\mathbf{L}_i$ at the cost of $O(\log n)$ per edge, and as the samples are taken independently they can be done in parallel. The guarantees of Theorem 1.6.9 then gives that the resulting graph Laplacian $\tilde{\mathbf{L}}_i$ has $O(n_i \log n\epsilon^{-2})$ edges such that:

$$(1-\epsilon)\tilde{\mathbf{L}}_i \preceq \mathbf{L}_i \preceq (1+\epsilon)\tilde{\mathbf{L}}_i$$

To put these together, note that $\hat{\mathbf{M}}$ is the sum of $\mathbf{L}_i$ plus a positive diagonal matrix $\mathbf{D}'$. The fact that $\sum_i n_i = 2m$ gives the bound on the total number of edges. Since we can also compute the degrees of each vertex j in $\mathbf{L}_i$ explicitly, $\mathbf{D}'$ can be obtained in in $O(\log n)$ depth and $O(m+n)$ work. Returning $\sum_i \tilde{\mathbf{L}}_i + \mathbf{D}'$ then gives the sparsifier. ■

Putting together this crude sparsification step and the sparsifiers given in Lemma 3.3.4 gives an algorithm for constructing the next level of the solver chain from the current one.

Lemma 3.3.7 *Given an $n \times n$ SDDM matrix $\boldsymbol{M}$ with m non-zero entries, splitting $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$ along with $\boldsymbol{\mathcal{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, and any error parameter ϵ. We can find in $O\left(\log^{O(1)} n\right)$ depth and $O\left(m \log^{O(1)} n\epsilon^{-2}\right)$ work a SDDM matrix $\tilde{\boldsymbol{M}}$ with $O\left(n \log^{O(1)} n\epsilon^{-2}\right)$ edges. Furthermore, $\tilde{\boldsymbol{M}}$ has a splitting $\tilde{\boldsymbol{M}} = \boldsymbol{D} - \tilde{\boldsymbol{A}}$ such that with high probability $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\tilde{\boldsymbol{A}}\boldsymbol{D}^{-1/2}$ satisfies:*

$$(1-\epsilon)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right) \preceq \boldsymbol{I} - \tilde{\boldsymbol{\mathcal{A}}} \preceq (1+\epsilon)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right)$$

Proof Once again let $\hat{\mathbf{A}}$ denote $\mathbf{A}\mathbf{D}^{-1}\mathbf{A}$ and $\hat{\mathbf{M}} = \mathbf{D} - \hat{\mathbf{A}}$. Note that $\mathbf{I} - \boldsymbol{\mathcal{A}} = \mathbf{D}^{-1/2}\left(\mathbf{D} - \hat{\mathbf{A}}\right)\mathbf{D}^{-1/2}$. The crude sparsifier given in lemma 3.3.6 allows us to find in $O\left(\log^{O(1)} n\right)$ depth and $O\left(m \log^{O(1)} n\right)$ work $\tilde{\mathbf{M}}'$ with $O\left(n \log^{O(1)} n\epsilon^{-2}\right)$ non-zero entries such that:

$$(1-\epsilon)\hat{\mathbf{M}} \preceq \tilde{\mathbf{M}}' \preceq (1+\epsilon)\hat{\mathbf{M}}$$

The size of $\tilde{\mathbf{M}}'$ means that we can run the spectral sparsifier given in Lemma 3.3.4 on it in $O\left(\log^{O(1)} n\right)$ depth and $O\left(m \log^{O(1)} n\epsilon^{-2}\right)$ work. Let the resulting matrix be $\tilde{\mathbf{M}}''$. It has $O\left(n \log^{O(1)} n\epsilon^{-2}\right)$ non-zero entries and satisfies:

$$(1-\epsilon)\,\tilde{\mathbf{M}}' \preceq \tilde{\mathbf{M}}'' \preceq (1+\epsilon)\,\tilde{\mathbf{M}}'$$

Combining these identities, and letting $\tilde{\mathbf{M}} = (1+\epsilon)^{-2}$ gives:

$$(1-\epsilon)^2\,\hat{\mathbf{M}} \preceq \tilde{\mathbf{M}}'' \preceq (1+\epsilon)^2\,\hat{\mathbf{M}}$$
$$(1+\epsilon)^{-2}(1-\epsilon)^2\,\hat{\mathbf{M}} \preceq \tilde{\mathbf{M}} \preceq \hat{\mathbf{M}}$$

The coefficient can be simplified since $(1+\epsilon)^{-1} \geq 1-\epsilon$, and $(1-\epsilon)^4 \geq 1-4\epsilon$. Replacing ϵ by $\epsilon/4$ and applying Lemma 3.3.3 then gives the required condition. ■

This allows us to construct the parallel solver chains as given in Definition 3.2.1. It in turn gives a version of our main result, Theorem 3.0.1, with a higher exponent of $\log n$ in both depth and work.

Lemma 3.3.8 *Given a connected graph Laplacian* $\mathbf{L}$ *with* m *non-zero entries such that the ratio between maximum and minimum edge weights is* U*. We can preprocess* $\mathbf{L}$ *in* $O(\log(nU)\log^{O(1)} n)$ *depth and* $O(m\log^3(nU)\log^{O(1)} n)$ *work such that with high probability for any* $\epsilon > 0$ *we have an algorithm* $\text{SOLVE}_{\boldsymbol{L}}$ *that's a symmetric linear operator in* $\boldsymbol{b}$ *such that:*

- *If* $\boldsymbol{Z_L}$ *is the symmetric matrix such that* $\boldsymbol{Z_L b} = \text{SOLVE}_{\boldsymbol{L}}(\boldsymbol{b})$*, then:* $(1-\epsilon)\boldsymbol{L}^{\dagger} \preceq \boldsymbol{Z_L} \preceq (1+\epsilon)\boldsymbol{L}^{\dagger}$
- *Given any vector* $\boldsymbol{b}$*,* $\text{SOLVE}_{\boldsymbol{L}}(\boldsymbol{b})$ *runs in* $O(\log(nU)\log^{O(1)} n\log(1/\epsilon))$ *depth and* $O(m\log^3(nU)\log^{O(1)} n\log(1/\epsilon))$ *work.*

We first need to show that a SDDM matrix can be constructed from $\mathbf{L}$ by increasing the diagonal slightly. To bound the eigenvalues of the resulting matrix, we need to invoke the Courant-Fischer theorem:

Lemma 3.3.9 (Courant-Fischer Theorem) *Let* $\boldsymbol{A}$ *be an* $n \times n$ *symmetric matrix with eigenvalues* $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$*. Then the minimum and maximum eigenvalues are given by:*

$$\lambda_1 = \min_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$
$$\lambda_n = \max_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

Lemma 3.3.10 *Given any graph Laplacian* $\mathbf{L}$ *with such that the ratio between maximum and minimum edge weights is* U*, we can create a SDDM matrix* $\boldsymbol{M}$ *such that:*

1. *If* Π *is the projection onto the range space of* $\mathbf{L}$*, aka. the space orthogonal to the all 1s vector, then* $\mathbf{L} \preceq \Pi \boldsymbol{M} \Pi \preceq 2\boldsymbol{L}$

2. *If* $\boldsymbol{M}$ *is directly split into* $\boldsymbol{D} - \boldsymbol{A}$ *by taking its diagonal to be* $\boldsymbol{D}$*, then all eigenvalues of* $\boldsymbol{\mathcal{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ *are in the range* $\left[-1 + \frac{1}{2n^3U}, 1 - \frac{1}{2n^3U}\right]$.

Proof Without loss of generality we may assume that the edge weights in $\mathbf{L}$ to the range $[1, U]$. Since $\mathbf{L}$ is connected, the minimum non-zero eigenvalue of $\mathbf{L}$ is at least $\frac{1}{n^2}$. Therefore, $\frac{1}{n^2}\Pi \preceq \mathbf{L}$ and $\mathbf{M} = \mathbf{L} + \frac{1}{n^2}\mathbf{I}$ is a good approximation to $\mathbf{L}$ after projecting onto its rank space.

We now upper bound the eigenvalues of $\boldsymbol{\mathcal{A}}$ using the Courant-Fischer Theorem given in Lemma 3.3.9. For any vector $\mathbf{x}$, by letting $\mathbf{y} = \mathbf{D}^{-1/2}\mathbf{x}$ we have:

$$\frac{\mathbf{x}^T\boldsymbol{\mathcal{A}}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \frac{\mathbf{y}^T\mathbf{A}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}$$

Let $\mathbf{D}'$ be the diagonal of the input Laplacian $\mathbf{L}$, aka. $\mathbf{D} = \mathbf{D}' + \frac{1}{n^2}\mathbf{I}$. Since $\mathbf{L} = \mathbf{D}' - \mathbf{A}$ is positive semi-definite, we have $\mathbf{y}^T\mathbf{A}\mathbf{y} \leq \mathbf{y}^T\mathbf{D}'\mathbf{y}$. Also, since the degree of each vertex is at most nU, $\mathbf{y}^T\mathbf{D}'\mathbf{y} \leq nU\mathbf{y}^T\mathbf{y}$. Combining these gives:

$$\begin{aligned}\frac{\mathbf{y}^T\mathbf{A}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} &\leq \frac{\mathbf{y}^T\mathbf{D}'\mathbf{y}}{\mathbf{y}^T\mathbf{D}'\mathbf{y} + \frac{1}{n^2}\mathbf{y}^T\mathbf{y}} \\ &\leq \frac{1}{1 + \frac{1}{n^3U}} \\ &\leq 1 - \frac{1}{2n^3U}\end{aligned}$$

The lower bound on the eigenvalues of $\boldsymbol{\mathcal{A}}$ follows similarly from the positive-definiteness of the positive Laplacian $\mathbf{D}' + \mathbf{A}$. ■

We also need preconditioned iterative methods to reduce the constant error to $\pm\epsilon$. Either preconditioned Richardson or Chebyshev iteration can suffice here.

Lemma 2.1.1 (Preconditioned Chebyshev Iteration) *There exists an algorithm* PRECONCHEBY *such that for any symmetric positive semi-definite matrices* $\boldsymbol{A}$*,* $\boldsymbol{B}$*, and* κ *where*

$$\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa\boldsymbol{A}$$

any error tolerance $0 < \epsilon \leq 1/2$*,* PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *in the exact arithmetic model is a symmetric linear operator on* $\boldsymbol{b}$ *such that:*

1. *If* $\boldsymbol{Z}$ *is the matrix realizing this operator, then* $(1-\epsilon)\boldsymbol{A}^\dagger \preceq \boldsymbol{Z} \preceq (1+\epsilon)\boldsymbol{A}^\dagger$

2. *For any vector* $\boldsymbol{b}$*,* PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *takes* $N = O\left(\sqrt{\kappa}\log\left(1/\epsilon\right)\right)$ *iterations, each consisting of a matrix-vector multiplication by* $\boldsymbol{A}$*, a solve involving* $\boldsymbol{B}$*, and a constant number of vector operations.*

Proof of Lemma 3.3.8:

Lemma 3.3.10 gives a SDDM matrix $\mathbf{M}$ such that $\mathbf{L} \preceq \mathbf{\Pi M \Pi} \preceq 2\mathbf{L}$. Combining the eigenvalue bound on $\boldsymbol{\mathcal{A}}$ with Corollary 3.2.5 gives that a parallel solver chain for $\mathbf{I} - \boldsymbol{\mathcal{A}}$ has depth bounded by $d = O\left(\log\left(n^3 U\right)\right) = O\left(\log\left(nU\right)\right)$. We will construct such a chain with $\epsilon_i = \frac{1}{d}$ for some constant c by invoking Lemma 3.3.7 d times.

Applying induction on the sizes of the sparsifiers produced gives that with high probability each level after the first one has $O\left(n \log^{O(1)} \epsilon^{-2}\right) = O\left(n \log^2\left(nU\right) \log^{O(1)}\right)$ edges. This in turn means that the construction of each level can be done in $O\left(\log^{O(1)} n\right)$ depth and $O\left(m \log^2\left(nU\right) \log^{O(1)} n\right)$ work, giving a total of $O\left(\log\left(nU\right) \log^{O(1)}\right)$ depth and $O\left(m \log^3\left(nU\right) \log^{O(1)} n\right)$ work. Also, the size of the parallel solver chain can be bounded by $O\left(m \log^3\left(nU\right) \log^{O(1)}\right)$

This parallel solver chain can then be used by Lemma 3.2.2 to evaluate a linear operator $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}$ such that:

$$(1+\epsilon)^{-d}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}} \preceq 5(1-\epsilon)^{-d}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1}$$

and for any vector $\mathbf{b}$, $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}\mathbf{b}$ can be evaluated by running $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}(\mathbf{b})$ in $O\left(d\right)$ depth and work proportional to the size of the parallel solver chain. Since we set ϵ to $\frac{1}{d}$, $(1+\epsilon)^{-d}$ and $(1-\epsilon)^{-d}$ can be bounded by constants, giving:

$$\frac{1}{3}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}} \preceq 15\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1}$$

Since $\mathbf{M} = \mathbf{D}^{1/2}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)\mathbf{D}^{1/2}$, applying Lemma 1.6.6 gives:

$$\frac{1}{3}\mathbf{M}^{-1} \preceq \mathbf{D}^{1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}\mathbf{D}^{1/2} \preceq 15\mathbf{M}^{-1}$$

Finally, combining this with the guarantees on $\mathbf{L}$ and $\mathbf{M}$ gives:

$$\frac{1}{6}\mathbf{L}^{\dagger} \preceq \mathbf{\Pi}\mathbf{D}^{1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}\mathbf{D}^{1/2}\mathbf{\Pi} \preceq 15\mathbf{L}^{\dagger}$$

The error can then be reduced to $\pm\epsilon$ using preconditioned Chebyshev iteration as given in Lemma 2.1.1 with $\mathbf{A} = \mathbf{L}$ and $\mathbf{B} = 15(\mathbf{\Pi}\mathbf{D}^{1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}\mathbf{D}^{1/2}\mathbf{\Pi})^{\dagger}$. Matrix-vector multiplications involving $\mathbf{A}$, $\mathbf{\Pi}$ and $\mathbf{D}^{1/2}$ can be done in $O\left(\log n\right)$ depth and $O\left(m\right)$ work. Therefore, the total depth/work is dominated by the calls to $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}$, each requiring $O\left(d\right) = O\left(\log\left(nU\right)\right)$ depth and $O\left(m \log^3\left(nU\right) \log^{O(1)}\right)$ work. The $O(\log(1/\epsilon))$ iterations required by preconditioned Chebyshev iteration then gives the total depth/work. ■

3.4 Alternate Construction of Parallel Solver Chains

We now give a more efficient algorithm for constructing parallel solver chains. It performs the spectral sparsification steps in ways similar to an algorithm by Koutis et al. [KLP12]. Specifically, it uses a solver for a spectrally close matrix to compute the sampling probabilities needed to generate a sparsifier. This spectrally close matrix is in turn obtained using the ultra-sparsification algorithm from Section 2.2, and therefore can be reduced to one of smaller size.

Since our goal is a polylog depth algorithm, we cannot perform all sparsification steps by recursing on the ultra-sparsifiers. Instead, we only do so for one matrix in the parallel solver chain, and reduce the solvers involving other matrices to it. This is done using the fact that $\mathbf{I} + \boldsymbol{\mathcal{A}}_i$ are well-conditioned for all $i \geq 1$. It allows us to apply polynomial approximations to factorizations that are more direct than Fact 3.1.1.

The key step in constructing the parallel solver chain is sparsifing a SDDM matrix $\mathbf{M}$. The current state-of-the-art methods for constructing sparsifiers with the fewest edges is to sample by effective resistance. This is described in Theorem 1.6.9, and the main algorithmic difficulty to compute the sampling probabilities. This can be done by approximately solving a number of linear systems involving $\mathbf{M}$ [SS08, KL11, KLP12]. When the approximate solver routine is viewed as a black-box, these construction algorithms can be stated as:

Lemma 3.4.1 *Given an $n \times n$ SDDM matrix $\boldsymbol{M}$ with m non-zero entries along with an algorithm* $\textsc{Solve}_{\boldsymbol{M}}$ *corresponding to a linear operator given by the matrix $\boldsymbol{Z}_{\boldsymbol{M}}$ such that $\frac{1}{2}\boldsymbol{M}^{-1} \preceq \boldsymbol{Z}_{\boldsymbol{M}} \preceq 2\boldsymbol{M}^{-1}$, and any error $\epsilon > 0$. We can compute using $O(\log n)$ parallel calls to* $\textsc{Solve}_{\boldsymbol{M}}$ *plus an overhead of $O(\log n)$ depth and $O(m \log n)$ work a matrix $\tilde{\boldsymbol{M}}$ with $O\left(n \log n \epsilon^{-2}\right)$ non-zero entries such that with high probability $(1-\epsilon)\boldsymbol{M} \preceq \tilde{\boldsymbol{M}} \preceq \boldsymbol{M}$.*

It's worth mentioning that any constants in the spectral approximation between $\mathbf{M}^{-1}$ and $\mathbf{Z}_{\mathbf{M}}$ is sufficient, and the ones above are picked for simplicity.

Our goal is to use this sparsification routine to generate the parallel solver chain given in Definition 3.2.1. We first show that if we're given an approximate solver for $\mathbf{I} - \boldsymbol{\mathcal{A}}_1$, we can construct an approximate solver for $\mathbf{I} - \boldsymbol{\mathcal{A}}_i^2$. Our starting point is once again the identity $1 - \alpha^2 = (1+\alpha)(1-\alpha)$. Its inverse can be symmetrized by moving a factor of $(1+\alpha)^{-1/2}$ to the right:

$$\left(1-\alpha^2\right)^{-1} = (1+\alpha)^{-1/2}(1-\alpha)^{-1}(1+\alpha)^{-1/2}$$

We first show that there exists a low degree polynomial that gives a good approximation to $(1+\alpha)^{-1/2}$, the inverse square root of $1+\alpha$.

Lemma 3.4.2 *For any $\delta > 0$, there exists a polynomial* $\textsc{ApxPwr}_{-\frac{1}{2}}(\lambda, \delta)$ *of degree $O(1/\delta)$ such that for all $\alpha \in [-1/2, 1]$:*

$$(1-\delta)(1+\alpha) \leq \textsc{ApxPwr}_{-\frac{1}{2}}(\alpha, \delta) \leq (1+\delta)(1+\alpha)^{-1/2}$$

Proof Scaling $1+\alpha$ down by a factor of $\frac{2}{3}$ and multiplying the result by $\left(\frac{3}{2}\right)^{1/2}$ still preserves multiplicative approximations. Therefore it suffices to work under the different assumption of $1+\alpha \in \left[\frac{1}{3}, \frac{4}{3}\right]$, or $|\alpha| \leq \frac{2}{3}$. The Taylor expansion of $f(\alpha) = (1+\alpha)^{-1/2}$, or McLaurin series of $f(\alpha)$ is:

$$\begin{aligned}
f(\alpha) &= 1 + \left(-\frac{1}{2}\right)\alpha + \frac{1}{2!}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\alpha^2 + \frac{1}{3!}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\left(-\frac{5}{2}\right)\alpha^3 + \ldots \\
&= \sum_{j=0}^{\infty} \frac{(-1)^j (2j)!}{(j!)^2 4^j}\alpha^j \\
&= \sum_{j=0}^{\infty} \frac{(-1)^j}{4^j}\binom{2j}{j}\alpha^j
\end{aligned}$$

Since $\binom{2j}{j}$ is one of the terms in the expansion o $(1+1)^{2j} = 4^j$, the magnitude of each coefficient is at most 1. Therefore if we only take the first k terms, the additive error is at most:

$$\sum_{j=k}^{\infty} |x|^j = \frac{|x|^j}{1-|\alpha|} \tag{3.2}$$

Since $|x| \leq \frac{2}{3}$, taking the $O(\log(1/\delta))$ terms gives an additive error is at most 0.1δ. We will let this polynomial be $\textsc{ApxPwr}_{-\frac{1}{2}}$. Also, since $1+x \leq 2$, $(1+\alpha)^{-1/2} \geq 2^{-1/2} \geq 0.1$. So this error is a δ multiplicative error as well. ■

Note that in terms of the original α, this is a polynomial in $\frac{2}{3}(1+\alpha) - 1 = \frac{2}{3}\alpha - \frac{1}{3}$. To evaluate it, we can either recollect the coefficients, or simply incur a constant factor overhead. This bound can be transferred to the matrix setting in a straight-forward manner using diagonalization.

Lemma 3.4.3 *For any matrix $\boldsymbol{\mathcal{A}}$ with all eigenvalues in the range $\left[-\frac{1}{2}, 1\right]$ and any $\delta > 0$, there exists a polynomial* $\textsc{ApxPwr}_{-\frac{1}{2}}(\lambda, \delta)$ *of degree $O(1/\delta)$ such that:*

$$(1-\delta)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1} \preceq \textsc{ApxPwr}_{-\frac{1}{2}}(\boldsymbol{\mathcal{A}}, \delta)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}\right)^{-1}\textsc{ApxPwr}_{-\frac{1}{2}}(\boldsymbol{\mathcal{A}}, \delta) \preceq (1+\delta)\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1}$$

Proof Let a diagonalization of $\boldsymbol{\mathcal{A}}$ be $\mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$. Then $\left(\mathbf{I} - \boldsymbol{\mathcal{A}}^2\right)^{-1} = \mathbf{U}^T\left(\mathbf{I} - \mathbf{\Lambda}^2\right)^{-1}\mathbf{U}$ while the inner term becomes:

$$\begin{aligned}
&\textsc{ApxPwr}_{-\frac{1}{2}}(\boldsymbol{\mathcal{A}}, \delta)\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)^{-1}\textsc{ApxPwr}_{-\frac{1}{2}}(\boldsymbol{\mathcal{A}}, \delta) \\
&= \mathbf{U}^T\textsc{ApxPwr}_{-\frac{1}{2}}(\mathbf{\Lambda}, \delta)\mathbf{U}\mathbf{U}^T\left(\mathbf{I} - \mathbf{\Lambda}\right)^{-1}\mathbf{U}\mathbf{U}^T\textsc{ApxPwr}_{-\frac{1}{2}}(\mathbf{\Lambda}, \delta)\mathbf{U}
\end{aligned}$$

The definition of spectral decomposition gives $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, removing these terms then simplifies this to:

$$\mathbf{U}^T \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathbf{\Lambda}, \delta\right)\left(\mathbf{I} - \mathbf{\Lambda}\right)^{-1} \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathbf{\Lambda}, \delta\right)\mathbf{U}$$

The middle term is a diagonal matrix, which means that the entries are operated on separately by the polynomials. Furthermore, the eigenvalue bounds gives that all entries of $\mathbf{\Lambda}$ are in the range $\left[-\frac{1}{2}, 1\right]$. This allows us to apply Lemma 3.4.2 with $\delta/3$ to obtain $\textsc{ApxPwr}_{-\frac{1}{2}}$ such that.

$$(1-\delta)\left(\mathbf{I} - \mathbf{\Lambda}^2\right)^{-1} \preceq \mathbf{U}^T \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathbf{\Lambda}, \delta\right)\left(\mathbf{I} - \mathbf{\Lambda}\right)^{-1} \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathbf{\Lambda}, \delta\right)\mathbf{U} \preceq (1+\delta)\left(\mathbf{I} - \mathbf{\Lambda}^2\right)^{-1}$$

Applying the composition lemma from Lemma 1.6.6 then gives the result. ■

The spectrum requirement precludes us from applying it to $\mathcal{A}_0$, whose spectrum may vary greatly. On the other hand, all the layers after the first in a parallel solver chain have $\mathbf{I} + \mathcal{A}_i$ well conditioned. Applying this lemma recursively gives that a good approximate inverse to $\mathbf{I} - \mathcal{A}_1$ leads to a good approximate inverse to $\mathbf{I} - \mathcal{A}_i$ for any $i \geq 1$.

Lemma 3.4.4 *Let $\mathcal{A}_0 \ldots \mathcal{A}_i$ be the first $i+1$ levels of a parallel solver chain with $\epsilon_1 \ldots \epsilon_i \leq \frac{\epsilon}{4}$. Suppose $\textsc{Solve}_{\boldsymbol{I}-\mathcal{A}_1}$ is an algorithm corresponding to a symmetric linear operator given by the matrix $\boldsymbol{z}_{\boldsymbol{I}-\mathcal{A}_1}$ such that $\kappa_{\min}\left(\boldsymbol{I} - \mathcal{A}_1\right)^{-1} \preceq \boldsymbol{Z}_{\boldsymbol{I}-\mathcal{A}_1} \preceq \kappa_{\max}\left(\boldsymbol{I} - \mathcal{A}_1\right)^{-1}$. Then for each $i \geq 1$ we have an algorithm $\textsc{Solve}_{\boldsymbol{I}-\mathcal{A}_i^2}$ corresponding to a symmetric linear operator on $\boldsymbol{b}$ such that:*

1. *If $\boldsymbol{Z}_{\boldsymbol{I}-\mathcal{A}_i^2}$ is the matrix such that $\textsc{Solve}_{\boldsymbol{I}-\mathcal{A}_i^2}(\boldsymbol{b}) = \boldsymbol{Z}_{\boldsymbol{I}-\mathcal{A}_i^2}$, then $\kappa_{\min}\left(1-\epsilon\right)^i\left(\boldsymbol{I} - \mathcal{A}_i^2\right) \preceq \boldsymbol{Z}_{\boldsymbol{I}-\mathcal{A}_i^2} \preceq \kappa_{\max}\left(1+\epsilon\right)^i\left(\boldsymbol{I} - \mathcal{A}_i^2\right)$.*
2. *$\textsc{Solve}_{\boldsymbol{I}-\mathcal{A}_i^2}(\boldsymbol{b})$ makes one call to $\textsc{Solve}_{\boldsymbol{I}-\mathcal{A}_1}$ plus $O\left(\log\left(1/\epsilon\right)\right)$ matrix-vector multiplications involving $\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_i$ in sequential order.*

Proof The proof is by induction on i. Lemma 3.2.4 gives that when $i \geq 1$, the eigenvalues of $\mathcal{A}_i$ are in the range $[-\frac{1}{10}, 1]$. Therefore Lemma 3.4.3 can be applied to all these matrices. If we use an error $\delta = \epsilon/4$, we have:

$$\left(1 - \frac{\epsilon}{4}\right)\left(\mathbf{I} - \mathcal{A}_i^2\right)^{-1} \preceq \textsc{ApxPwr}_{-\frac{1}{2}}(\mathcal{A}_i, \frac{\epsilon}{4})\left(\mathbf{I} - \mathcal{A}_i\right)^{-1} \textsc{ApxPwr}_{-\frac{1}{2}}(\mathcal{A}_i, \frac{\epsilon}{4}) \preceq \left(1 + \frac{\epsilon}{4}\right)\left(\mathbf{I} - \mathcal{A}^2\right)^{-1}$$

The base case of $i = 1$ then follows from applying Lemma 1.6.6.

For the inductive case, we first show the lower bound as the upper bound follows similarly. The guarantees of the solver chain gives:

$$\left(1 - \epsilon_{i-1}\right)\left(\mathbf{I} - \mathcal{A}_i\right)^{-1} \preceq \left(\mathbf{I} - \mathcal{A}_{i-1}^2\right)^{-1}$$

Composing this bound with $\textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathcal{A}_i, \frac{\epsilon}{4}\right)$ and substituting in the above bounds gives:

$$\left(1 - \frac{\epsilon}{4}\right)\left(1 - \epsilon_{i-1}\right)\left(\mathbf{I} - \mathcal{A}_i^2\right)^{-1} \preceq \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathcal{A}_i, \frac{\epsilon}{4}\right)\left(\mathbf{I} - \mathcal{A}_{i-1}^2\right)^{-1} \textsc{ApxPwr}_{-\frac{1}{2}}\left(\mathcal{A}_i, \frac{\epsilon}{4}\right)$$

The fact that $\epsilon_{i-1} \leq \frac{\epsilon}{4}$ and $\left(1 - \frac{\epsilon}{4}\right)^2 \geq 1 - \epsilon$ simplifies the LHS to $(1-\epsilon)\left(\mathbf{I} - \boldsymbol{\mathcal{A}}_i^2\right)^{-1}$. The induction hypothesis also gives the following guarantee for $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}^2}$:

$$\kappa_{\min}(1-\epsilon)^{i-1}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}_{i-1}^2\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}^2}$$

Composing it with $\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)$ by Lemma 1.6.6 then gives:

$$\kappa_{\min}(1-\epsilon)^{i-1}\,\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)\left(\mathbf{I} - \boldsymbol{\mathcal{A}}_{i-1}^2\right)^{-1}\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)$$
$$\preceq \textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}^2}\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)$$

A spectral upper bound can also be obtained similarly. Therefore, it suffices for $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ to evaluate $\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}}^2\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)$. Lemma 3.4.2 gives that $\textsc{ApxPwr}_{-\frac{1}{2}}\left(\boldsymbol{\mathcal{A}}_i, \frac{\epsilon}{4}\right)$ can be evaluated using $O(\log(1/\epsilon))$ matrix-vector multiplications involving $\boldsymbol{\mathcal{A}}_i$, while $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}}^2$ can be evaluated by calling $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_{i-1}}$. Combining the work/depth bounds of these operations gives the inductive hypothesis for i as well. ■

This means that an approximate inverse to $\mathbf{I} - \boldsymbol{\mathcal{A}}_1$ suffices for constructing the entire parallel solver chain.

Lemma 3.4.5 *Let $\boldsymbol{I} - \boldsymbol{\mathcal{A}}$ be a normalized $n \times n$ SDDM matrix with m non-zero entries and all eigenvalues in the range $[\frac{1}{\kappa}, 2 - \frac{1}{\kappa}]$, and $\boldsymbol{\mathcal{A}}_1$ is the second level of a solver chain with $\boldsymbol{\mathcal{A}}_0 = \boldsymbol{\mathcal{A}}, \epsilon_0 = \frac{1}{10}$ and $\boldsymbol{I} - \boldsymbol{\mathcal{A}}_1$ has m_1 non-zero entries. If we're given a routine $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}$ corresponding to a matrix $\boldsymbol{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}$ such that $\boldsymbol{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}\boldsymbol{b} = \textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}(\boldsymbol{b})$ and $\frac{9}{10}(\boldsymbol{I}-\boldsymbol{\mathcal{A}})^{-1} \preceq \boldsymbol{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}} \preceq \frac{11}{10}(\boldsymbol{I}-\boldsymbol{\mathcal{A}})^{-1}$. Then we can construct the rest of the parallel solver chain so that for all $i \geq 1$, $\epsilon_i = \frac{1}{c\log\kappa}$ for some constant c and the total size of the parallel solver chain is $O(m+m_1+n\log^3\kappa\log n)$. Furthermore, the construction makes $O(\log\kappa)$ sequential batches of $O(\log n)$ parallel calls to $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}_1}$ plus an overhead of $O(\log^2\kappa\log\log\kappa\log n)$ depth and $O(m_1\log^2\kappa\log n + n\log^5\kappa\log^3 n)$ work.*

Proof We will show by induction that we can construct the levels of the solver chain along with the corresponding SDDM matrices and splitting $\mathbf{M}_i = \mathbf{D}_i - \mathbf{A}_i$. The base case of $i = 1$ follows from the first level of the parallel solver chain being given.

For the inductive case, suppose we have level i of the chain. Since Corollary 3.2.5 limits the depth of the chain to $O(\log\kappa)$, $i \leq c'\log\kappa$ for some fixed constant c'. Therefore an appropriate choice of c gives $(1 + \frac{4}{c\log\kappa})^i\frac{11}{10} \leq \frac{3}{2}$ and $(1 - \frac{4}{c\log\kappa})^i\frac{9}{10} \geq \frac{3}{4}$. Applying Lemma 3.4.4 then gives $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ whose running time consists of one invocation of $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}}$ plus an overhead of $O(\log\kappa\log\log\kappa + \log n)$ depth and $O(m_1 + i\cdot n\log^2\kappa\log\log\kappa\log n) \leq O(m_1 + n\log^3\kappa\log\log\kappa\log n)$ work. Furthermore, $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}(\mathbf{b})$ evaluates to a symmetric linear operator on $\mathbf{b}$ and if the matrix corresponds to this operator is $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$, then the guarantees of Lemma 3.4.4

gives:

$$\frac{3}{4}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2} \preceq \frac{3}{2}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2\right)^{-1}$$

Then we can proceed in the same way as our proof of Lemma 3.3.7. Let the SDDM matrix corresponding to $\mathbf{I}-\boldsymbol{\mathcal{A}}_i$ be $\mathbf{M}_i = \mathbf{D}_i - \mathbf{A}_i$. Let $\hat{\mathbf{M}}$ denote the SDDM matrix corresponding the next level in the exact setting, $\hat{\mathbf{M}} = \mathbf{D}_i - \mathbf{A}_i\mathbf{D}^{-1}\mathbf{A}_i$. We first invoke Lemma 3.3.6 with error $\frac{\epsilon_i}{4}$ to generate a crude sparsifier of $\hat{\mathbf{M}}$, $\hat{\mathbf{M}}'$. The number of non-zeros in this matrix, $\hat{m}_i'$, can be bounded by $O(m_i \log n \epsilon^{-2})$.

$$\left(1-\frac{\epsilon_i}{4}\right)\hat{\mathbf{M}} \preceq \hat{\mathbf{M}}' \preceq \left(1+\frac{\epsilon_i}{4}\right)\hat{\mathbf{M}}$$

Since $\mathbf{M} = \mathbf{D}_i^{-1/2}(\mathbf{I}-\boldsymbol{\mathcal{A}}_i)\mathbf{D}_i^{-1/2}$, combining this with the guarantees of $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ and Lemma 1.6.6 gives:

$$\left(1-\frac{\epsilon_i}{4}\right)\frac{3}{4}\hat{\mathbf{M}}'^{-1} \preceq \mathbf{D}_i^{1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}\mathbf{D}_i^{1/2} \preceq \left(1+\frac{\epsilon_i}{4}\right)\frac{3}{2}\hat{\mathbf{M}}'^{-1}$$

Since $\epsilon_i \leq 1$, this means that $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ with multiplications by $\mathbf{D}^{1/2}$ before and after suffices as an approximate solver of $\hat{\mathbf{M}}'$ for the sparsification process provided in Lemma 3.4.1. Using it with error $\frac{\epsilon_i}{4}$ then gives $\mathbf{M}_{i+1}$ that's a sparsifier for $\hat{\mathbf{M}}'$, and in turn $\hat{\mathbf{M}}$. The splitting and normalization are then given by Lemma 3.3.3. Therefore we can construct level $i+1$ of the solver chain as well.

This construction makes $O(\log n)$ calls to $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ plus an overhead of $O(\log n)$ depth and $O(\hat{m}'_i \log n)$ work. When $i = 1$, this work overhead is $O(m_i \log^2 \kappa \log n)$, while when $i > 1$, it is $O(n \log^4 \kappa \log^3 n)$. Combining this with the $O(\log \kappa)$ bound on the number of levels and the costs of the calls to $\textsc{Solve}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_i^2}$ then gives the total. ■

It remains to obtain this approximate inverse for $\mathbf{I}-\boldsymbol{\mathcal{A}}_1$. We do so by reducing it to a smaller system that's spectrally close, and use an approximate inverse for that as an approximate inverse to $\mathbf{I}-\boldsymbol{\mathcal{A}}_1$. To construct a parallel solver chain for that system, we need to in turn recurse on an even smaller system. This leads to an algorithm that generates a sequence of matrices similar to the sequential preconditioner chain given in Definition 2.3.1, with the main difference being that the spectral conditions between levels are much more relaxed. This is because the solver for $\mathbf{I}-\boldsymbol{\mathcal{A}}^{(i+1)}$ is only used in the construction of the parallel solver chain for $\mathbf{I}-\boldsymbol{\mathcal{A}}^{(i)}$.

Definition 3.4.6 *A parallel solver construction chain for $\boldsymbol{\mathcal{A}}$ is two sequences of matrices $\boldsymbol{\mathcal{A}}^{(0)} = \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(l)}$ and $\boldsymbol{\mathcal{A}}_1^{(0)}, \boldsymbol{\mathcal{A}}_1^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(l-1)}$, size bounds $m^{(0)}, m^{(1)}, \ldots, m^{(l)}$, approximation factor κ_{apx}, and bound on eigenvalues κ such that:*

1. *For all $j < l$, $\boldsymbol{\mathcal{A}}_1^{(j)}$ is the first level of a parallel solver chains with $\boldsymbol{\mathcal{A}}_0 = \boldsymbol{\mathcal{A}}^{(j)}$ and $\epsilon_0 = \frac{1}{10}$.*

2. *The number of non-zero entries in $\boldsymbol{\mathcal{A}}^{(j)}$ and $\boldsymbol{\mathcal{A}}_1^{(j)}$ can be bounded by $m^{(j)}$ and $O(m^{(j)} \log n)$ respectively.*

3. *For all $j < l$, $m^{(j+1)} \leq \frac{1}{10} m^{(j)}$ and $m^{(l)}$ is less than a fixed constant.*

4. *All eigenvalues of $\mathcal{A}^{(j)}$ are in the range $\left[-1 + \frac{1}{\kappa}, 1 - \frac{1}{\kappa}\right]$.*

5. *If* $\text{SOLVE}_{\boldsymbol{I}-\mathcal{A}^{(j+1)}}$ *is a routine corresponding to a matrix $\mathbf{Z}_{\boldsymbol{I}-\mathcal{A}^{(j+1)}}$ such that:*

$$\kappa_{\min}(\boldsymbol{I} - \mathcal{A}^{(j+1)})^{-1} \preceq \mathbf{Z}_{\boldsymbol{I}-\mathcal{A}^{(j+1)}} \preceq \kappa_{\max}(\boldsymbol{I} - \mathcal{A}^{(j+1)})^{-1}$$

   Then we can obtain a routine $\text{SOLVE}_{\boldsymbol{I}-\mathcal{A}_1^{(j)}}$ *corresponding to a matrix $\mathbf{Z}_{\boldsymbol{I}-\mathcal{A}_1^{(j)}}$ such that:*

$$\kappa_{\min}\left(\boldsymbol{I} - \mathcal{A}_1^{(j)}\right)^{-1} \preceq \mathbf{Z}_{\boldsymbol{I}-\mathcal{A}_1^{(j)}} \preceq \kappa_{apx} \cdot \kappa_{\max}\left(\boldsymbol{I} - \mathcal{A}_1^{(j)}\right)^{-1}$$

   and $\text{SOLVE}_{\boldsymbol{I}-\mathcal{A}_1^{(i)}}$ *involves one call to* $\text{SOLVE}_{\boldsymbol{I}-\mathcal{A}^{(i+1)}}$ *plus an overhead of $O(\log n)$ depth and $O(m_i \log n)$ work.*

Such a solver construction chain allows us to construct parallel solver chain backwards starting from the last level.

Lemma 3.4.7 *Given a parallel solver construction chain for $\mathcal{A}$, we can obtain in $O(\sqrt{\kappa_{apx}} \log^3 \kappa \log n + \log^3 \kappa \log\log \kappa \log^2 n)$ depth and $O(m(\sqrt{\kappa_{apx}} \log^4 \kappa \log^2 n + \log^5 \kappa \log^3 n))$ work a parallel solver chain with $\mathcal{A}_0 = \mathcal{A}$, total size $O(m \log^3 \kappa \log n)$, $\epsilon_0 = \frac{1}{5}$, and $\epsilon_i = \frac{1}{c \log^3 \kappa}$ for all $i \geq 1$.*

Proof We show by induction backwards on the levels j that we can build a parallel solver chain with $\mathcal{A}_0 = \mathcal{A}^{(j)}$, total size $O(m_i \log^3 \kappa \log n)$, and the required errors in $O(\sqrt{\kappa_{apx}} \log^2 \kappa \log n + \log^2 \kappa \log\log \kappa \log^2 n)$ depth and $O(m^{(j)}(\sqrt{\kappa_{apx}} \log^4 \kappa \log^2 n + \log^5 \kappa \log^3 n))$ work

The base case follows from m_i being smaller than a fixed constant. For the inductive case, suppose the result is true for $j+1$. Then the inductive hypothesis gives parallel solver chain with $\mathcal{A}_0^{(j+1)} = \mathcal{A}^{(j+1)}$ with total size size $O(m \log^3 \kappa \log n)$. Lemma 3.2.2 then gives a solver algorithm $\text{SOLVE}_{\mathbf{I}-\mathcal{A}^{(j+1)}}$ corresponding to a matrix $\mathbf{Z}_{\mathbf{I}-\mathcal{A}^{(j+1)}}$ such that

$$\frac{1}{2}\left(\mathbf{I} - \mathcal{A}^{(j+1)}\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\mathcal{A}^{(j+1)}} \preceq 10\left(\mathbf{I} - \mathcal{A}^{(j+1)}\right)^{-1}$$

and $\text{SOLVE}_{\mathbf{I}-\mathcal{A}^{(j+1)}}(\mathbf{b})$ runs in $O(\log \kappa \log n)$ depth and $O(m^{(j+1)} \log^3 \kappa \log n)$ work. By the solver conversion condition given as Part 5 of Definition 3.4.6, this leads a routine $\text{SOLVE}'_{\mathbf{I}-\mathcal{A}_1^{(j)}}$ corresponding to a matrix $\mathbf{Z}'_{\mathbf{I}-\mathcal{A}_1^{(j)}}$ such that:

$$\frac{1}{2}\left(\mathbf{I} - \mathcal{A}_1^{(j)}\right)^{-1} \preceq \mathbf{Z}'_{\mathbf{I}-\mathcal{A}_1^{(j)}} \preceq 10\kappa_{apx}\left(\mathbf{I} - \mathcal{A}_1^{(j)}\right)^{-1}$$

Preconditioned Chebyshev iteration as stated in Lemma 2.6.3 then allows us improve the quality of this solver to any constant in $O(\sqrt{\kappa_{apx}})$ iterations. This gives a routine $\text{SOLVE}_{\mathbf{I}-\mathcal{A}_1^{(j)}}$ corresponding to a matrix $\mathbf{Z}'_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}}$ such that:

$$\frac{9}{10}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}\right)^{-1} \preceq \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}} \preceq \frac{11}{10}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}\right)^{-1}$$

and for any vector $\mathbf{b}$, $\text{SOLVE}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}}(\mathbf{b})$ runs in $O(\sqrt{\kappa_{apx}}\log\kappa\log n)$ depth and $O(\sqrt{\kappa_{apx}}(m^{(j)}\log n + m^{(j+1)}\log^3\kappa\log n)) \leq O(\sqrt{\kappa_{apx}}m^{(j)}\log^3\kappa\log n)$ work. This routine meets the requirement of Lemma 3.4.5, and thus we can construct the rest of the parallel solver chain with $\boldsymbol{\mathcal{A}}_0^{(j)} = \boldsymbol{\mathcal{A}}^{(j)}$. The total work/depth can then be obtained by combining the number of calls made to $\text{SOLVE}_{\mathbf{I}-\boldsymbol{\mathcal{A}}_1^{(j)}}(\mathbf{b})$ and the work/depth overhead.

Summing over the $\log n$ levels of the construction chain and using the fact that m_is are geometrically decreasing gives the overall bounds.

■

Building the parallel solver construction chain is done using the following lemma:

Lemma 3.4.8 (Crude Sparsification) *Given any weighted graph $G = (V, E, \boldsymbol{w})$, and any parameter γ, we can find in $O(\gamma^{-2}\log^3 n)$ depth and $O(m\log n)$ work a graph H on the same vertex set with $n-1+\gamma m$ edges such that:*

$$\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq O\left(\gamma^{-2}\log^3 n\right)\boldsymbol{L}_G$$

The proof of this lemma requires the construction of low-stretch subgraphs, which in turn relies on parallel low diameter partition schemes. We will give a detailed exposition of this in Chapter 4, specifically Lemma 3.4.8 will be proven in Section 4.3. We also need to reduce $\mathbf{L}_H$ to one of size $O(\gamma m)$ using greedy-elimination. A parallelization of this procedure was given as Lemma 26 in [BGK$^+$13].

Lemma 3.4.9 (Parallel Greedy Elimination) *There is a procedure* PARGREEDYELIMINATION *that given an $n \times n$ SDDM matrix $\boldsymbol{M} = \boldsymbol{D} - \boldsymbol{A}$ corresponding to a graph on n vertices and $m = n - 1 + m'$ edges, produces with high probability in $O(n+m)$ work and $O(\log n)$ depth a factorization:*

$$\boldsymbol{P}^T\boldsymbol{M}\boldsymbol{P} = \boldsymbol{U}^T\begin{bmatrix}\boldsymbol{I} & \mathbf{0}^T\\ \mathbf{0} & \boldsymbol{M}'\end{bmatrix}\boldsymbol{U}$$

Where $\boldsymbol{M}'$ is a SDDM matrix with at most $O(m')$ vertices and edges, $\boldsymbol{P}$ is a permutation matrix and $\boldsymbol{U}$ is an upper triangular matrix such that:

- *The lower-right $n' \times n'$ block of $\boldsymbol{U}$ corresponding to the entries in $\boldsymbol{M}'$ is the identity matrix.*
- *The diagonal of $\boldsymbol{M}'$ is at most the corresponding diagonal entries in $\boldsymbol{P}^T\boldsymbol{M}\boldsymbol{P}$.*

- *Matrix-vector products with* $\boldsymbol{U}$, $\boldsymbol{U}^T$, $\boldsymbol{U}^{-1}$ *and* $\boldsymbol{U}^{-T}$ *can be computed in* $O(\log n)$ *depth and* $O(n)$ *work.*

We need an additional bound on the effect on eigenvalues of this greedy elimination procedure.

Lemma 3.4.10 *Let* $\boldsymbol{M}'$ *be a matrix outputted by greedy elimination procedure given in Lemma 3.4.9 above, and let* $\boldsymbol{M}' = \boldsymbol{D}' - \boldsymbol{A}'$ *be a splitting obtained by taking its diagonal. The minimum eigenvalue of the normalized matrix* $\boldsymbol{I} - \boldsymbol{\mathcal{A}}' = \boldsymbol{D}'^{-1/2}\boldsymbol{M}'\boldsymbol{D}'^{-1/2}$ *is at least the minimum eigenvalue of* $\boldsymbol{I} - \boldsymbol{\mathcal{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{M}\boldsymbol{D}^{-1/2}$.

Proof Since eigenvalues are invariant under simultaneous permutations of rows/columns, we may assume during the rest of this proof that $\mathbf{P} = \mathbf{I}$. The Courant Fischer theorem given in Lemma 3.3.9 gives that the minimum eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}'$ is:

$$\min_{\mathbf{x}'} \frac{\mathbf{x}'^T (\mathbf{I} - \boldsymbol{\mathcal{A}}')\, \mathbf{x}'}{\mathbf{x}'^T\mathbf{x}'} = \min_{\mathbf{y}'} \frac{\mathbf{y}'^T\mathbf{M}'\mathbf{y}'}{\mathbf{y}'\mathbf{D}'\mathbf{y}'}$$

Similarly, for any vector $\mathbf{y}$, $\frac{\mathbf{y}^T\mathbf{M}\mathbf{y}}{\mathbf{y}\mathbf{D}\mathbf{y}}$ gives an upper bound on the minimum eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}$. Therefore, it suffices to show that for any vector $\mathbf{y}'$, we can obtain a $\mathbf{y}$ with a smaller quotient.

For a vector $\mathbf{y}'$, consider the vector:

$$\mathbf{y} = \mathbf{U}^{-1} \begin{bmatrix} \mathbf{0} \\ \mathbf{y}' \end{bmatrix}$$

We have:

$$\begin{aligned} \mathbf{y}^T\mathbf{M}\mathbf{y} &= \mathbf{y}^T\mathbf{U}^T \begin{pmatrix} \mathbf{I} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{M}' \end{pmatrix} \mathbf{U}\mathbf{y} \\ &= \begin{bmatrix} \mathbf{0} \\ \mathbf{y}' \end{bmatrix}^T \begin{bmatrix} \mathbf{I} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{M}' \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{y}' \end{bmatrix} \\ &= \mathbf{y}'^T\mathbf{M}'\mathbf{y}' \end{aligned}$$

Also, since the $\mathbf{U}$ is the identity matrix in the lower-right block corresponding to $\mathbf{M}'$, its inverse is the identity in this block as well. Therefore the entries in $\mathbf{y}$ corresponding to $\mathbf{y}$ are the same. Since $\mathbf{D}$ is non-negative and its entries corresponding to $\mathbf{D}'$ are larger, we have $\mathbf{y}^T\mathbf{D}\mathbf{y} \geq \mathbf{y}'\mathbf{D}'\mathbf{y}'$. Combining these gives:

$$\frac{\mathbf{y}^T\mathbf{M}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \leq \frac{\mathbf{y}'^T\mathbf{M}'\mathbf{y}'}{\mathbf{y}'^T\mathbf{D}'\mathbf{y}'}$$

Therefore the minimum eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}'$ is greater or equal to the minimum eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}$. ■

These two lemmas allows us to construct the entire sequence of sparsifiers needed to build the parallel solver chain. Since we are incurring spectral distortions of $O\left(\mathbf{poly}\left(\log n\right)\right)$ for $O\left(\log n\right)$ steps, a direct calculation leads to a distortion of $\log^{O(\log n)} n$ to the spectrum. We can do better by only controlling the minimum eigenvalue and using the fact that all eigenvalues of $\boldsymbol{\mathcal{A}}$ are between $[-1, 1]$.

Lemma 3.4.11 *If a SDDM matrix $\boldsymbol{M}$ with splitting $\boldsymbol{D} - \boldsymbol{A}$ such that all eigenvalues of $\boldsymbol{I} - \boldsymbol{\mathcal{A}}$ are at least λ, then the matrix $\boldsymbol{M}' = (1 + \lambda)\boldsymbol{D} - \boldsymbol{A}$ satisfies:*

1. *All eigenvalues of $\boldsymbol{\mathcal{A}}'$ are in the range $\left[-1 + \lambda, 1 - \frac{1}{2}\lambda\right]$.*
2. *$\frac{1}{2}\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}\right) \preceq \boldsymbol{I} - \boldsymbol{\mathcal{A}}' \preceq 2\left(\boldsymbol{I} - \boldsymbol{\mathcal{A}}\right)$.*

Proof Normalizing $\mathbf{M}'$ using the splitting $(1 + \lambda)\mathbf{D} - \mathbf{A}$ gives:

$$\begin{aligned}\boldsymbol{\mathcal{A}}' &= \left(\left(1+\lambda\right)\mathbf{D}\right)^{-1/2}\mathbf{A}\left(\left(1+\lambda\right)\mathbf{D}\right)^{-1/2}\\ &= \frac{1}{1+\lambda}\boldsymbol{\mathcal{A}}\end{aligned}$$

Since $(-1+\lambda)\mathbf{I} \preceq \boldsymbol{\mathcal{A}} \preceq \mathbf{I}$, $(-1+\lambda)\mathbf{I} \preceq \frac{1}{1+\lambda}\boldsymbol{\mathcal{A}} \preceq \frac{1}{1+\lambda}\mathbf{I}$. The lower eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}'$ is at least λ and the upper eigenvalue is at most $1 + \frac{1}{1+\lambda} = 2 - \frac{\lambda}{1+\lambda} \leq 2 - \frac{\lambda}{2}$ since $\lambda \leq 1$.

We now show the relation between $\mathbf{I} - \boldsymbol{\mathcal{A}}$ and $\mathbf{I} - \boldsymbol{\mathcal{A}}'$. The LHS side follows from $\mathbf{I} - \mathbf{A}' \preceq \frac{1}{1+\lambda}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)$ and $\frac{1}{1+\lambda} \geq \frac{1}{2}$. The RHS can be obtained by adding the following two inequalities:

$$\begin{aligned}\frac{1}{1+\lambda}\mathbf{I} - \boldsymbol{\mathcal{A}}' &= \frac{1}{1+\lambda}\left(\mathbf{I} - \boldsymbol{\mathcal{A}}\right)\\ &\preceq \mathbf{I} - \boldsymbol{\mathcal{A}}\\ \frac{\lambda}{1+\lambda}\mathbf{I} &\preceq \lambda\mathbf{I} \preceq \mathbf{I} - \boldsymbol{\mathcal{A}}\end{aligned}$$

■

This method of adding a small identity matrix can also be used to increase the minimum eigenvalue of the matrix returned by the crude sparsification algorithm given in Lemma 3.4.8. This process leads to the following size reduction step for constructing the next level of a parallel solver construction chain.

Lemma 3.4.12 *Given a SDDM matrix $\boldsymbol{M}$ with splitting $\boldsymbol{D} - \boldsymbol{A}$ such that the minimum eigenvalue of $\boldsymbol{I} - \boldsymbol{\mathcal{A}}$ is at least λ. For any parameter γ we can construct in $O(m \log n)$ work and $O(\gamma^{-2} \log^3 n)$ depth. a SDDM matrix $\boldsymbol{M}' = \boldsymbol{D}' - \boldsymbol{A}'$ with γm vertices and edges such that:*

1. *The minimum eigenvalue of $\boldsymbol{I} - \boldsymbol{\mathcal{A}}'$ is at least $\frac{1}{2}\lambda$.*

2. *Suppose* $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}'}$ *is a routine corresponding to a matrix* $\mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}'}$ *such that:*

$$\kappa_{\min}\left(\boldsymbol{I}-\boldsymbol{\mathcal{A}}'\right)^{-1} \preceq \mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}'} \preceq \kappa_{\max}\left(\boldsymbol{I}-\boldsymbol{\mathcal{A}}'\right)^{-1}$$

Then we can obtain a routine $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}$ *corresponding to a matrix* $\mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}$ *such that:*

$$\kappa_{\min}\left(\boldsymbol{I}-\boldsymbol{\mathcal{A}}\right)^{-1} \preceq \mathbf{Z}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}} \preceq O\left(\gamma^{-2}\log^3 n\right)\cdot\kappa_{\max}\left(\boldsymbol{I}-\boldsymbol{\mathcal{A}}\right)^{-1}$$

and $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}}$ *involves one call to* $\textsc{Solve}_{\boldsymbol{I}-\boldsymbol{\mathcal{A}}'}$ *plus an overhead of* $O(\log n)$ *depth and* $O(n\log n)$ *work.*

Proof Let G be the graph corresponding to $\mathbf{A}$, $\mathbf{D}_G$ its diagonal (aka. weighted degrees of $\mathbf{A}$) and $\mathbf{D}_+$ be $\mathbf{D}-\mathbf{D}_G$, the extra diagonal elements in $\mathbf{M}$. Applying Lemma 3.4.8 gives H with $n-1+\gamma m$ edges such that

$$\mathbf{L}_G \preceq \mathbf{L}_H \preceq \kappa_{apx}\mathbf{L}_G$$

Where $\kappa_{apx} = O\left(\gamma^{-2}\log^3 n\right)$. The eigenvalue bound on $\boldsymbol{\mathcal{A}}$ and the fact that $-\mathbf{A} \preceq \mathbf{D}$ gives $\lambda\mathbf{D} \preceq \mathbf{M} \preceq 2\mathbf{D}$. Therefore if we let $\hat{\mathbf{M}} = \mathbf{D}_+ + \mathbf{L}_H + \kappa_{apx}\lambda\mathbf{D}$, we have:

$$\mathbf{M} \preceq \hat{\mathbf{M}} \preceq \kappa_{apx}\mathbf{M} + \kappa_{apx}\lambda\mathbf{D}$$

So $\mathbf{M}$ and $\hat{\mathbf{M}}$ are within a factor of $2\kappa_{apx}$ of each other. We now lower bound the minimum eigenvalue of $\mathbf{I}-\hat{\boldsymbol{\mathcal{A}}}$. Since $\mathbf{L}_H \preceq \kappa_{apx}\mathbf{L}_G$, using the indicator vectors as test vectors gives $\mathbf{D}_H \preceq \kappa_{apx}\mathbf{D}_G$ So we have:

$$\begin{aligned}\hat{\mathbf{D}} &\preceq \kappa_{apx}\mathbf{D}_G + \mathbf{D}_+ + \kappa_{apx}\lambda\mathbf{D}\\ &\preceq 2\kappa_{apx}\mathbf{D} \qquad \text{Since } \lambda \leq 1\end{aligned}$$

On the other hand since $\mathbf{L}_H$ and $\mathbf{D}_+$ are both positive semi-definite, we also have $\kappa_{apx}\lambda\mathbf{D} \preceq \hat{\mathbf{M}}$. Hence $\frac{1}{2}\lambda\hat{\mathbf{D}} \preceq \mathbf{M}$ and the minimum eigenvalue of $\mathbf{I}-\hat{\boldsymbol{\mathcal{A}}}$ is at least $\frac{1}{2}\lambda$.

We can then obtain $\mathbf{M}'$ by the greedy elimination procedure given in Lemma 3.4.9 on $\hat{\mathbf{M}}$. $\boldsymbol{\mathcal{A}}'$ can be in turn be obtained by splitting $\mathbf{M}'$ directly using its diagonal. The bound on its minimum eigenvalue follows from Lemma 3.4.10.

For the conversion of solver guarantees, the factorization given by Lemma 3.4.9 gives:

$$\begin{aligned}\mathbf{P}^T\hat{\mathbf{M}}\mathbf{P} &= \mathbf{U}^T\begin{bmatrix}\mathbf{I} & \mathbf{0}^T\\ \mathbf{0} & \mathbf{M}'\end{bmatrix}\mathbf{U}\\ \hat{\mathbf{M}}^{-1} &= \mathbf{P}\mathbf{U}^{-1}\begin{bmatrix}\mathbf{I} & \mathbf{0}^T\\ \mathbf{0} & \mathbf{M}'^{-1}\end{bmatrix}\mathbf{U}^{-T}\mathbf{P}^T\\ &= \mathbf{P}\mathbf{U}^{-1}\begin{bmatrix}\mathbf{I} & \mathbf{0}^T\\ \mathbf{0} & \mathbf{D}'^{-1/2}\left(\mathbf{I}-\boldsymbol{\mathcal{A}}'\right)^{-1}\mathbf{D}'^{-1/2}\end{bmatrix}\mathbf{U}^{-T}\mathbf{P}^T\end{aligned}$$

Replacing $(\mathbf{I} - \boldsymbol{\mathcal{A}}'^{-1})$ by $\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}'}$ gives the following approximate solver for $\hat{\mathbf{M}}$:

$$\mathbf{Z}_{\hat{\mathbf{M}}} = \mathbf{P}\mathbf{U}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{D}'^{-1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}'}\mathbf{D}'^{-1/2} \end{bmatrix} \mathbf{U}^{-T}\mathbf{P}^T$$

Applying Lemma 1.6.6 with $\mathbf{V} = \begin{bmatrix} \mathbf{I} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{D}'^{-1/2} \end{bmatrix} \mathbf{U}^{-T}\mathbf{D}^{1/2}$ gives:

$$\kappa_{\min}\hat{\mathbf{M}}^{-1} \preceq \mathbf{Z}_{\hat{\mathbf{M}}} \preceq \kappa_{\max}\hat{\mathbf{M}}^{-1}$$

Inverting the spectral bounds between $\mathbf{M}$ and $\hat{\mathbf{M}}$ gives $\mathbf{M}^{-1} \preceq 2\kappa_{apx}\hat{\mathbf{M}}^{-1} \preceq 2\kappa_{apx}\mathbf{M}^{-1}$. Combining this with the fact that $(\mathbf{I} - \boldsymbol{\mathcal{A}})^{-1} = \mathbf{D}^{1/2}\mathbf{M}\mathbf{D}^{1/2}$ and applying Lemma 1.6.6 once again gives:

$$\kappa_{\min}\mathbf{M}^{-1} \preceq 2\kappa_{apx}\mathbf{D}^{1/2}\mathbf{Z}_{\hat{\mathbf{M}}}\mathbf{D}^{1/2} \preceq 2\kappa_{apx} \cdot \kappa_{\max}\mathbf{M}^{-1}$$

Hence one possible linear operator to evaluate is:

$$\begin{aligned} \mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}} &= 2\kappa_{apx}\mathbf{D}^{1/2}\mathbf{Z}_{\hat{\mathbf{M}}}\mathbf{D}^{1/2} \\ &= 2\kappa_{apx}\mathbf{D}^{1/2}\mathbf{P}\mathbf{U}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{0}^T \\ \mathbf{0} & \mathbf{D}'^{-1/2}\mathbf{Z}_{\mathbf{I}-\boldsymbol{\mathcal{A}}'}\mathbf{D}'^{-1/2} \end{bmatrix} \mathbf{U}^{-T}\mathbf{P}^T\mathbf{D}^{1/2} \end{aligned}$$

Note that matrix-vector products involving $\mathbf{P}$, $\mathbf{P}^T$, $\mathbf{D}^{1/2}$, $\mathbf{D}'^{-1/2}$ $\mathbf{U}^{-1}$, and $\mathbf{U}^{-T}$ can all be evaluated in $O(\log n)$ depth and $O(n)$ work. Therefore we can obtain a routine $\text{SOLVE}_{\hat{\mathbf{M}}}$ that evaluates the linear operator given by $\mathbf{Z}_{\hat{\mathbf{M}}}$ using one call to $\text{SOLVE}_{\mathbf{I}-\boldsymbol{\mathcal{A}}'}$ plus an overhead of $O(\log n)$ depth and $O(n)$ work. ■

Lemma 3.4.13 *Given an $n \times n$ SDDM matrix $\boldsymbol{M}$ and splitting $\boldsymbol{D} - \boldsymbol{A}$ such that all eigenvalues of $\boldsymbol{\mathcal{M}}^{(i)}$ are in the range $[-1 + \frac{1}{\kappa^{(0)}}, 1 - \frac{1}{\kappa^{(0)}}]$. We can construct in $O\left(\log^5 n\right)$ depth and $O\left(m \log n\right)$ work a parallel solver construction chain for $\boldsymbol{\mathcal{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ with $\kappa_{apx} = O\left(\log^5 n\right)$ and $\kappa \leq m\kappa^{(0)}$.*

Proof Consider the following algorithm that sets $\boldsymbol{\mathcal{A}}^{(0)} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and iteratively generates $\boldsymbol{\mathcal{A}}^{(j)}$ and spectrum bounds $\kappa^{(j)}$ until the size of $\boldsymbol{\mathcal{A}}^{(j)}$ is smaller than some constant.

- Let $\mathbf{M}^{(j)} = \mathbf{D}^{(j)} - \mathbf{A}^{(j)}$ be the SDDM matrix corresponding to $\mathbf{I} - \boldsymbol{\mathcal{A}}^{(j)}$.
- Compute an approximation to $\mathbf{D}^{(j)} - \mathbf{A}^{(j)}\mathbf{D}^{(j)-1}\mathbf{A}^{(j)}$ using the square sparsifier in Lemma 3.3.6 with error $\epsilon_0 = \frac{1}{10}$, $\mathbf{M}_1^{(j)}$.
- Set $\boldsymbol{\mathcal{A}}_1^{(j)}$ to the splitting of $\mathbf{M}_1^{(j)}$ given by Lemma 3.3.3.
- Apply Lemma 3.4.12 to $\mathbf{M}_1^{(j)}$ with $\gamma = 1/O\left(\log n\right)$ for some suitable (fixed) constant. to obtain $\tilde{\mathbf{M}}^{(j+1)}$ with corresponding $\tilde{\boldsymbol{\mathcal{A}}}^{(j+1)}$.

- Apply Lemma 3.4.11 to $\tilde{\boldsymbol{\mathcal{A}}}^{(j+1)}$ with $\lambda = \frac{1}{\kappa^{(j)}}$ to create $\boldsymbol{\mathcal{A}}^{(j+1)}$. Set $\kappa^{(j+1)}$ to $4\kappa^{(j)}$.

We first show that the size of $\boldsymbol{\mathcal{A}}^{(j)}$ decreases geometrically. Since ϵ_0 was set to a constant, the guarantee of Lemma 3.3.6 gives that $\boldsymbol{\mathcal{A}}_1^{(j)}$ has $O(m \log n)$ edges. This meets the requirement of Definition 3.4.6, and the spectral guarantees of $\boldsymbol{\mathcal{A}}_1^{(j)}$ follows from Lemma 3.3.3. Also, the guarantees of Lemma 3.4.8 then gives that $\tilde{\mathbf{M}}_1^{(j)}$ has $n - 1 + O(\gamma m \log n)$ edges, which is then reduced to $O(\gamma m \log n)$ via. greedy elimination. Therefore setting $\gamma = 1/O(\log n)$ for the right choice of constants gives the desired reduction factor, which in turn implies $l = O(\log n)$.

The conversion of solver routines follows from Lemma 3.4.12 and the spectral guarantees of Lemma 3.4.11. It remains to lower bound the spectrum of the normalized matrices generated. Suppose all eigenvalue of $\boldsymbol{\mathcal{A}}^{(j)}$ are in the range $\left[-1 + \lambda^{(j)}, 1 - \lambda^{(j)}\right]$. Then Lemma 3.2.4 gives that the minimum eigenvalue of $\mathbf{I} - \boldsymbol{\mathcal{A}}_1^{(j)}$ is at least $\lambda^{(j)}$. The guarantees of the size reduction routine from Lemma 3.4.12 ensures that minimum eigenvalue of $\mathbf{I} - \tilde{\boldsymbol{\mathcal{A}}}^{(j+1)}$ is at least $\frac{1}{2}\lambda^{(j)}$. This can only be further halved the routine from Lemma 3.4.11, so all eigenvalues of $\mathbf{I} - \boldsymbol{\mathcal{A}}^{(j+1)}$ are in the range $[-1 + \frac{1}{4}\lambda^{(j)}, 1 - \frac{1}{4}\lambda^{(j)}]$ and we can set $\kappa^{(j+1)}$ to $4\kappa^{(j)}$. Since the sizes of the levels decrease by factors of 10, $\kappa \leq m^{-1}\kappa^{(0)}$ suffices as an overall bound.

■

We can now put these together to prove the main result of this chapter.

Proof of Theorem 3.0.1: By Lemma 3.3.10, it suffices to solve a linear system $\mathbf{I} - \boldsymbol{\mathcal{A}}$ where all eigenvalues of $\boldsymbol{\mathcal{A}}$ are in the range $\left[-1 + \frac{1}{2n^3U}, 1 - \frac{1}{2n^3U}\right]$. We first build a parallel solver construction chain using Lemma 3.4.13 with $\kappa_{apx} = O\left(\log^5 n\right)$ and $\kappa = 2n^5U$ in $O\left(\log^5 n\right)$ depth and $O(m \log n)$ work.

Invoking Lemma 3.4.7 then gives a parallel solver chain in $O(\sqrt{\kappa_{apx}} \log^3 \kappa \log n + \log^3 \kappa \log\log \kappa \log^2 n) = O(\log^3 \kappa \log^{7/2} n + \log^3 \kappa \log\log \kappa \log^2 n)$ depth and $O(m(\sqrt{\kappa_{apx}} \log^4 \kappa \log^2 n + \log^5 \kappa \log^3 n)) = O(m(\log^4 \kappa \log^{9/2} n + \log^5 \kappa \log^3 n))$ work. The resulting chain has total size $O(m \log^3 \kappa \log n)$. Invoking this chain with Lemma 3.2.2 then gives the depth/work bounds for a solver with constant relative error. This error can then be reduced using preconditioned iterative methods such as Chebyshev iteration as stated in Lemma 2.1.1, and the resulting guarantee can also be converted using Lemma 1.6.7. ■

The factorization used in our construction should also extend to give solvers. The only additional step needed is to evaluate we evaluate the operator $(\mathbf{I} + \boldsymbol{\mathcal{A}})^{1/2}$, which can be done in ways similar to ApxPwr. We omit presenting this since the resulting bound is are more than the variant using the factorization given in Fact 3.1.1. However, we believe further developments of this approach may allow us to compute a wider range of functions based on the eigenvalues of the matrix, such as matrix exponentials/logarithms, and log determinants.

Chapter 4

Construction of Low-Stretch Subgraphs

The sparsifiers and preconditioners in the previous two chapters rely on upper bounding effective resistances, or equivalently edge sampling probabilities, using a tree. These trees were first used in subgraph preconditioners by Boman and Hendrickson [BH01], and played crucial roles in all subsequent constructions. Due to connections to metric embeddings, these probabilities are often known as stretch. In this view, both the graph and the tree can be viewed as defining metrics given by their shortest path distance. For a tree T, we use $\text{dist}_T(u, v)$to denote the length of the shortest path between vertices u and v in the tree. Using this notation the stretch of an edge $e = uv$ can be defined as:

Definition 4.0.1 *Given a graph $G = (V, E)$, edge lengths **l**, and a tree T, the **stretch** of an edge $e = uv$ is:*

$$\boldsymbol{str}_T(e) = \frac{dist_T(u, v)}{\boldsymbol{l}_e}$$

In metric embedding definitions, the denominator term is often written as $\text{dist}_G(u, v)$. These two terms are equivalent when G is a metric, and this modification is needed since our input graphs may not have this property.

Intuitively, $\mathbf{str}_T(e)$ corresponds to the factor by which the edge $e = uv$ is 'stretched' by rerouting it into the tree. In comparison to our definition of stretch from Section 2.1, they are identical in the unweighted setting since $\frac{1}{\mathbf{w}_e} = 1$. In the weighted case, they are equivalent by setting the length of an edge to be the inverse of its weight.

$$\mathbf{l}_e = \frac{1}{\mathbf{w}_e}$$

This change of variables is in some sense natural because weights correspond to conductance, and it is the inverse of them, resistances, that are additive along tree paths.

The use of low stretch spanning trees and the exact stretch values in combinatorial preconditioning is continuously evolving. They were first used as ways of explicitly rerouting the edges by Boman and Hendrickson [BH01] and Spielman and Teng [ST06]. More recently, stretch has been used as sampling probabilities by Koutis et al. [KMP11] (see Chapter 2) and Kelner et al. [KOSZ13]. One property of stretch remains the same in applications of low stretch embeddings and is unlikely to change:

smaller total stretch = better performance.

Due to the direct dependency of solver running times on total stretch, and the relation to metric embedding, algorithms for generating low stretch spanning trees have been extensively studied. All the algorithms for generating low stretch spanning trees to date have two main components: an inner loop that partitions the graph into pieces; and an outer loop that stitches these partitions together to form a good spanning tree. The outer loops controls the overall flow of the algorithm, as well as the form of the tree generated. As a result, they have been the main focus in almost all the low-stretch spanning tree constructions to date. A brief overview of the trade-offs made in these algorithms is in Section 4.1. Our improvements to this framework relies on the fact that for the construction of sparsifiers, the stretch of a small fraction of edges can be very large. This is the main insight in our parallel algorithm for constructing low stretch subgraphs in Section 4.3 as well as low $\ell_{1/2+\alpha}$-moment spanning trees in Section 4.4.

The partition routines were first introduced in the setting of distributed algorithms by Awerbuch et al. [Awe85, ALGP89]. Their aggregated bound across multiple iterations were improved by Seymour [Sey95], but otherwise these subroutines have remained largely unchanged in various low stretch embedding algorithms. One drawback of this routine is that it's highly sequential: each piece in the decomposition relies on the result of all previous ones. As a result, one of the main components of the Blelloch et al. parallel solver algorithm [BGK+13] was an alternate parallel decomposition scheme. In Section 4.2 we show a more refined version of this scheme that obtains the same parameters as the sequential algorithm. The low stretch subgraph algorithm in Section 4.3 is a direct consequence of this algorithm and techniques introduced by Blelloch et al. [BGK+13]. However, in order to understand the goal of these decompositions, it is helpful to give a picture of the overall algorithm.

4.1 Low Stretch Spanning Trees and Subgraphs

A direct approach for generating trees with small total stretch would be to find trees such that every edge have small stretch. Unfortunately this is not possible for the simplest non-tree graph: the cycle on n vertices as shown in Figure 4.1. As a tree contains $n-1$ edges, each possible tree omits one edge from the cycle. If all the edges on the cycle have the same weight, this edge then has a stretch of $n-1=\Omega(n)$.

As a result, it is crucial that only the total stretch is small. Or in other words, the edges have small stretch on average. This requirement is still significantly different than other definitions of special spanning trees such as the minimum weight spanning tree. Here an illustrative example is the $\sqrt{n} \times \sqrt{n}$ square grid with unit edge weights shown in Figure 4.2. The tree on the left is both a minimum spanning tree and a shortest path tree (under appropriate tie breaking schemes). However, it leads to a high total stretch. For each vertical edge beyond the middle column, at least

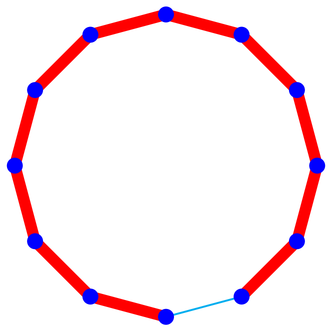


Figure 4.1: Cycle on n vertices. Any tree T can include only $n-1$ of these edges. If the graph is unweighted, the remaining edge has stretch $n-1$.

$\sqrt{n}$ horizontal edges are needed to travel between its endpoints. Therefore its stretch is at least $\sqrt{n}$. So the $n/2$ edges in the right half of the square grid contribute a total stretch of $n^{1.5}$.

In the tree on the right, all edges along the middle row and column still have stretch $O(\sqrt{n})$. However, the middle row and column only have $O(\sqrt{n})$ edges and so they contribute only $O(n)$ to the total stretch. Recall that all we need is a low total stretch, so a small number of high-stretch edges is permitted. Having accounted for the edges in the middle row and column, the argument can then be repeated on the 4 smaller subgraphs of size $n/4$ formed by removing the middle row and column. These pieces have trees that are constructed similarly, leading to the recurrence

$$\text{TotalStretch}(n) = 4 \cdot \text{TotalStretch}(n/4) + O(n).$$

which solves to $\text{TotalStretch}(n) = O(n \log n)$.

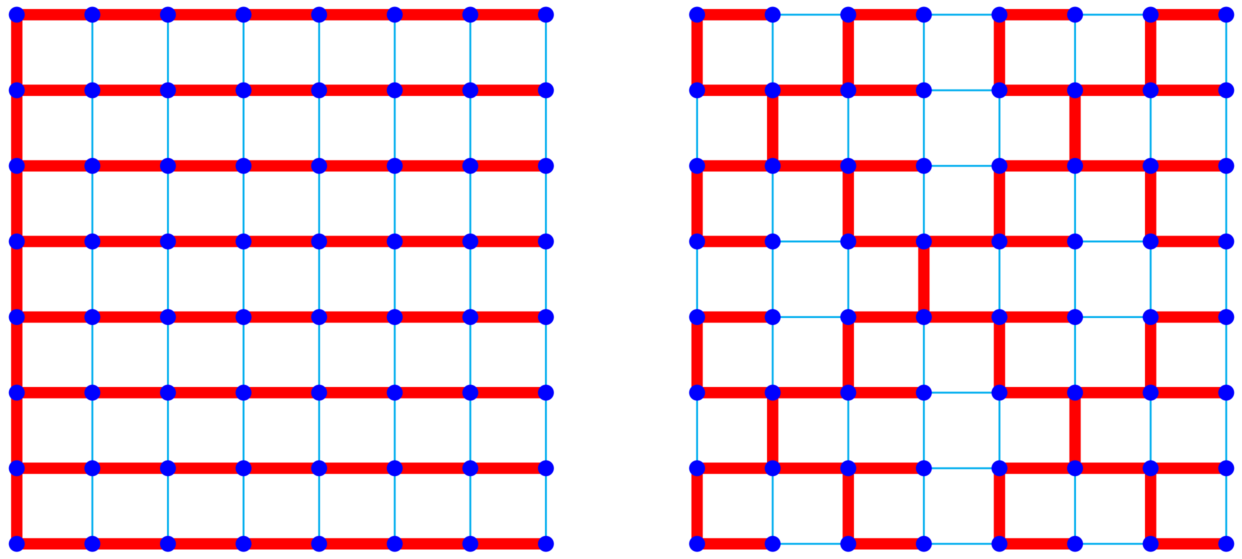


Figure 4.2: Two possible spanning trees of the unit weighted square grid, shown with red edges.

A generalization of this type of trade-off, which keeps the number of high stretch edges small, forms the basis of all low stretch spanning tree algorithms. Constructions of such trees for general graphs were first proposed in the context of online algorithms by Alon et al. [AKPW95], leading to the AKPW low stretch spanning trees. In these settings, a more crucial quantity is the expected stretch of a single edge. This is a stronger requirement since applying linearity of expectation over it also allows us to bound the total stretch. On the other hand, here it is not crucial for the tree to be a spanning tree. As a result, subsequent works generated trees that are not necessarily subtrees, and the problem was more commonly formulated as embedding arbitrary metrics. A series of results by Bartal [Bar96, Bar98], and Fakcharoenphol et al. [FRT04] led to an expected stretch of $O(\log n)$ per edge, which was shown to be optimal by Alon et al. [AKPW95].

However, to date it's only known how to use subtrees to generate sparsifiers and preconditioners. The first nearly-linear time solver used the AKPW low stretch spanning trees, which had $O\left(m \exp\left(\sqrt{\log n \log\log n}\right)\right)$ total stretch [ST06]. As a result, subsequent improvements in SDD linear system solvers have often gone hand in hand with improvements in algorithms for finding low stretch spanning trees. A series of works by Elkin et al. [EEST08], Abraham et al. [ABN08] and Abraham and Neiman [AN12] led to sequential algorithms for finding trees with $O(m \log n \log\log n)$ total stretch. The nearly $m \log n$ running time of the solver algorithm presented in Chapter 2 also made the runtime of finding the tree one of the bottlenecks. Koutis et al. [KMP11] addressed this by modifying the Abraham et al. [ABN08] low stretch spanning tree algorithm, giving a running time of $O\left(n \log n \log\log n + m \log n\right)$. This modification is also used in the current best low stretch spanning tree result by Abraham and Neiman [AN12], a nearly-optimal total stretch of $O\left(m \log n \log\log n\right)$ in $O\left(n \log n \log\log n + m \log n\right)$ time.

The situation is similar for parallel solvers. Finding a low stretch spanning tree is one of the main technical components in the first nearly-linear work, sublinear depth algorithm by Blelloch et al. [BGK$^+$13]. The analysis of this algorithm utilized a quantity more specialized for solvers. Because the tree is only used to reduce the number of edges by a small factor (a constant or **poly** $(\log n)$), it is acceptable for a small fraction of edges to have very high stretch. One way to view this, as described in Section 2.5 is that edge's sampling probability can be upper bounded by 1. Alternatively, they can be viewed as extra edges in addition to the tree, and the sum of them forms a low stretch subgraph. One example of such a subgraph is shown in Figure 4.3.

For a grid, the number of edges of stretch s is about n/s. Therefore, if we treat all edges of stretch more than **poly** $(logn)$ as 'extra' edges, we obtain a total of $n \log n/$**poly** $(logn) = n/$**poly** $(logn)$ extra edges. On the other hand, since stretches grow geometrically, there are only $\log\log n$ 'groups' with stretch less than **poly** $(logn)$. As the total stretch from each such group is $O(n)$, their total is $O(n \log\log n)$.

Another way to view this is that we begin by partitioning the grid into $n_0 \times n_0$ sized sub-grids, and applying the recursive-C construction to each grid. The total stretch in each of these sub-grid is $n_0^2 \log n_0$, which when summed over the $\left(\frac{\sqrt{n}}{n_0}\right)^2$ sub-grids gives $O\left(n \log n_0\right)$. On the other hand, the number of edges between them is $O\left(\frac{n}{n_0}\right)$. Setting $n_0 =$ **poly** $(\log n)$ then gives both the total stretch and number of extra edges.

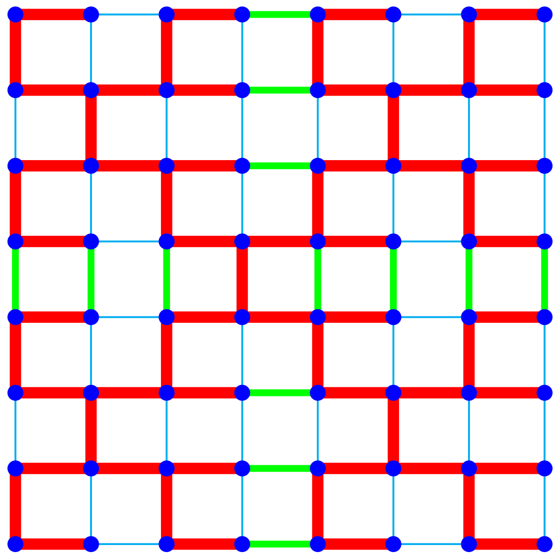


Figure 4.3: Low stretch spanning tree for the square grid (in red) with extra edges of high stretch (narrower, in green). These edges can be viewed either as edges to be resolved later in the recursive algorithm, or as forming a subgraph along with the tree

Blelloch et al. showed that including these extra edges leads to nearly-linear work solvers using the AKPW low stretch spanning trees [BGK$^+$13]. As such trees have $O\left(m\exp\left(\sqrt{\log n\log\log n}\right)\right)$ total stretch, it suggests that higher stretch edges need to be accounted differently. The results in this chapter are based around implications of such treatments of high stretch edges that are closely related to the solvers shown in Chapters 2 and 3. Specifically we show:

1. A simplified version of the low stretch subgraph algorithm by Blelloch et al. [BGK$^+$13] that suffices for the parallel edge reduction component from Section 3.4. Here high stretch edges are kept as extras, leading to subgraphs instead of spanning trees.
2. We show that slight modifications to the low stretch spanning tree algorithm by Abraham and Neiman [AN12] meets the requirements of Lemma 2.5.1 for a $O\left(m\log n\log\left(1/\epsilon\right)\right)$ time solver. Here stretch is measured under a different function that's smaller for higher stretch values.

The low stretch subgraph routine shown in Section 4.3 also requires a partition routine known as low (strong) diameter decomposition. Such routines are used at the core of most tree embedding algorithms in ways similar to partitioning the grid into smaller ones. Parallel low (strong) diameter decomposition algorithms were first introduced by Blelloch et al. [BGK$^+$13]. We will start by showing an improved version of it that obtains the same parameters as sequential algorithms.

4.2 Parallel Partition Routine

At the core of various low stretch embedding schemes are routines that partition the vertices of a graph into low diameter subgraphs with few edges between them. Such routines are also used at the core for a number of algorithms such as: approximations to sparsest cut [LR99, She09]; construction of spanners [Coh98]; parallel approximations of shortest path in undirected graphs [Coh00].

In order to formally specify the diameter of a piece, it is crucial to emphasize the distinction between weak and strong diameter. The diameter of a piece $S \subseteq V$ can be defined in two ways, weak and strong diameter. Both of them set diameter to the maximum length of a shortest path between two vertices in S, while the difference is in the set of allowed paths. **Strong diameter** restricts the shortest path between two vertices in S to only use vertices in S, while **weak diameter** allows for shortcuts through vertices in $V \setminus S$. The optimal tree metric embedding algorithm relies on weak diameter [FRT04]. It has been parallelized with polylog work overhead by Blelloch et al. [BGT12], although the total work remains quadratic.

A trend in algorithms that use weak diameter is that their running time tends to be super-linear. This is also the case with many parallel algorithms for computing low diameter decompositions such as the ones by Awerbuch et al. [ABCP92] and Linial and Sax [LS93] [1]. To date, nearly-linear work algorithms for finding tree embedding use strong diameter instead. While this leads to more difficulties in bounding diameters, the overall work is easier to bound since each piece certifies its own diameter, and therefore does not need to examine other pieces. For generating low stretch spanning trees, strong diameter is also crucial because the final tree (which the graph embeds into) is formed by combining the shortest path tree in each of the pieces. As a result, we will use diameter of a piece to denote **strong diameter**, and define a low diameter decomposition as follows:

Definition 4.2.1 *Given an undirected, unweighted graph $G = (V, E)$, a (β, d) decomposition is a partition of V into subsets $S_1 \dots S_k$ such that:*

- *The (strong) diameter of each S_i is at most d.*
- *The number of edges uv with vertices belonging to different pieces is at most βm.*

A standard choice of parameters for such decompositions is $(\beta, O(\frac{\log n}{\beta}))$, which are in some sense optimal. The simpler bottom-up schemes for constructing low stretch embeddings set β to $\log^{-c} n$ for some constant c. As a result, a dependency in β^{-1} in the depth of the decomposition algorithm is acceptable. This makes the depth of these algorithms more than the $\mathcal{NC}$ algorithms for large values of β. However, they suffice for the purpose of generating tree/subgraph embedding in polylog depth, as well as low work parallel graph algorithms.

Obtaining these decompositions in the sequential setting can be done via. a process known as ball growing. This process starts with a single vertex, and repeatedly adds the neighbors of the current set into the set. It terminates when the number of edges on the boundary is less than a β fraction of the edges within, which is equivalent to the piece having small conductance. Once the first piece is found, the algorithm discards its vertices and repeats on the remaining graph. The final bound of βm edges between the pieces can in turn be obtained by summing this bound over all

[1]The Linial and Sax algorithm [LS93] adapted to low diameter decompositions could be improved to $O(m \cdot \text{diameter})$ work

the pieces. Using a consumption argument, one could prove that the diameter of a piece does not exceed $O(\frac{\log n}{\beta})$. Because the depth can depend on $1/\beta$, and the piece's diameter can be bounded by $O(\frac{\log n}{\beta})$, finding a single piece is easy to parallelize. However, the strong diameter requirement means that we cannot start finding the second ball until we're done finding the first. This leads to a chain of sequential dependencies that may be as long as $\Omega(n)$, and is the main challenge in obtaining a parallel decomposition algorithm.

An algorithm for low diameter decompositions was given by Blelloch et al. [BGK$^+$13]. It gave a $\left(\beta, O\left(\frac{\log^4 n}{\beta}\right)\right)$ decomposition in $O(\frac{\log^3 n}{\beta})$ depth and $O(m \log^2 n)$ work. This algorithm showed that some of the ball growing steps can be performed simultaneously in parallel, leading to balls which have small overlap. Then a randomly shifted shortest path routine is used to resolve these overlaps. We will show an improved version of these ideas that combines these two steps into a simple, global routine. This leads to a simple algorithm that picks random shifts in a more intricate way and assigns vertices to pieces using one shortest path invocation. Our algorithm can also be viewed as a parallel version of the probabilistic decompositions due to Bartal [Bar96]. It is also related to a decomposition algorithm by Gupta and Talwar [GT13]. To describe our partition scheme, we first need to define some notations.

4.2.1 Notations

We begin by stating some standard notations. Given a graph G, we use $\text{dist}(u, v)$ to denote the length of the shortest path from u to v. As with earlier works on tree embedding, we will pick a special vertex in each piece, and use the distance to the farthest vertex from it as an estimate for the diameter. This simplification can be made since the graph is undirected and the final bound allows for constant factors (specifically 2). We will denote this special vertex the center of the piece, and denote the piece centered at u using S_u.

As the number of pieces in the final decomposition may be large (e.g. the line graph), a parallel algorithm needs to construct a number of pieces simultaneously. On the other hand, for closely connected graphs such as the complete graph, a single piece may contain the entire graph. As a result, if too many pieces are grown independently, the total work may become quadratic. The decomposition algorithm from [BGK$^+$13] addressed this trade-off by gradually increasing the number of centers picked iteratively. It was motivated by the (β, W) decompositions used in an algorithm by Cohen for approximating shortest paths in undirected graphs [Coh00]. By running iterations with gradually more centers, it can be shown that the resulting pieces at each iteration have small overlap. This overlap is in turn resolved using a shifted shortest path algorithm, which introduces shifts (denoted by δ) at the centers and assigns vertex v to S_u that minimizes the shifted distance:

$$\text{dist}_{-\delta}(u, v) = \text{dist}(u, v) - \delta_u. \tag{4.1}$$

It was shown that by picking shifts uniformly from a sufficiently large range, a $(\beta, O(\frac{\log^c n}{\beta}))$ decomposition can be obtained.

Our algorithm can be viewed as a more streamlined algorithm that combines these two components. The gradual increase in the number of centers can be emulated by adding a large step-like increase to the shifts of centers picked in earlier iterations. Furthermore, the need to have exponentially decreasing number of centers in the iterations suggests that the exponential distribution can be used in place of the (locally) uniform distribution. This distribution has been well-studied, and we will rely on some of its properties that are often used to estimate fault tolerance [Tri02]. For a parameter γ, this distribution is defined by the density function:

$$f_{Exp}(x, \gamma) = \begin{cases} \gamma \exp(-\gamma x) & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

We will denote it using *Exp*(γ) and will also make use of its cumulative density function:

$$F_{Exp}(x, \gamma) = \mathbf{Pr}\left[Exp(\gamma) \leq x\right] = \begin{cases} 1 - \exp(-\gamma x) & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

4.2.2 Partition Routine

We now describe our partition routine. Aside from being parallelizable, this routine can be viewed as an alternative to the repeated ball-growing arguments used in sequential algorithms. Pseudocode of a simplified version of our algorithm is shown in Algorithm 5. As states, it is a quadratic algorithm. However, we will describe efficient versions of it in Sections 4.2.3.

Algorithm 5 Partition Algorithm Using Exponentially Shifted Shortest Paths

```
PARTITION
Input: Undirected, unweighted graph $G = (V, E)$, parameter $\beta$ and parameter $d$ indicating failure probability.
Output: $(\beta, O(\log n/\beta))$ decomposition of $G$ with probability at least $1 - n^{-d}$.
1: For each vertex $u$, pick $\delta_u$ independently from $Exp(\beta)$
2: Compute $S_u$ by assigning each vertex $v$ to the vertex that minimizes $\text{dist}_{-\delta}(u, v)$, breaking ties lexicographically
3: return $\{S_u\}$
```

We start by showing that the assignment process readily leads to bounds on strong diameter. Specifically, the strong diameter of S_u can be measured using distances from u in the original graph.

Lemma 4.2.2 *If $v \in S_u$ and v' is the last vertex on the shortest path from u to v, then $v' \in S_u$ as well.*

Proof The proof is by contradiction, suppose v' belongs to $S_{u'}$ for some $u' \neq u$. The fact that v' is the vertex before v on the shortest path from u implies $\text{dist}_{-\delta}(u, v) = \text{dist}_{-\delta}(u, v') + 1$. Also, as

v' is adjacent to v, we also have $\text{dist}_{-\delta}(u', v) \leq \text{dist}_{-\delta}(u', v') + 1$. Since v' belongs to $S_{u'}$ instead of S_u, we must have one of the following two cases:

1. v' is strictly closer to u' than u in terms of shifted distance. In this case we have $\text{dist}_{-\delta}(u', v') < \text{dist}_{-\delta}(u, v')$, which when combined with the conditions above gives:

$$\begin{aligned}\text{dist}_{-\delta}(u', v) \leq & \text{dist}_{-\delta}(u', v') + 1\\ < & \text{dist}_{-\delta}(u, v') + 1\\ = & \text{dist}_{-\delta}(u, v).\end{aligned}$$

   So v is strictly closer to u' than u as well, which implies that v should not be assigned to S_u.

2. The shifted distances are the same, and u' is lexicographically earlier than u. Here a similar calculation gives $\text{dist}_{-\delta}(u', v) \leq \text{dist}_{-\delta}(u, v)$. If the inequality holds strictly, we are back to the case above. In case of equality, the assumption that u' is lexicographically earlier than u means v should not be in S_u as well.

■

Note that the second case is a zero probability event, and its proof is included to account for roundings in implementations that we will describe in Section 4.2.3.

To bound the strong diameter of the pieces, it suffices to bound the distance from a vertex to the center of the piece that it is assigned to. Since any vertex $v \in S_u$ could have been potentially included in S_v, the shift value of the center δ_u serves as an upper bound on the distance to any vertex in S_u. Therefore, $\delta_{\max} = \max_u \delta_u$ serves as an upper bound for the diameter of each piece.

Lemma 4.2.3 *Furthermore, with high probability, $\delta_u \leq O(\frac{\log n}{\beta})$ for all vertices u.*

Our proof below proof closely following the presentation in Chapter 1.6. of [Fel71].

Proof By the cumulative distribution function of the exponential distribution, the probability of $\delta_u \geq (d+1) \cdot \frac{\ln n}{\beta}$ is:

$$\begin{aligned}\exp\left(-(d+1) \cdot \beta \frac{\ln n}{\beta}\right) = & \exp(-(d+1)\ln n)\\ \leq & n^{-(d+1)}.\end{aligned}$$

Applying union bound over the n vertices then gives the bound. ■

The other property that we need to show is that few edges are between the pieces. We do so by bounding the probability of two endpoints of an edge being assigned to two different pieces. In order to keep symmetry in this argument, it is helpful to consider shifted distances from a vertex to the midpoint of an edge. This slight generalization can be formalized by replacing an edge uv with two length $1/2$ edges, uw and wv. We first show that an edge's end points can be in different pieces only if there are two different vertices whose shifted shortest path to its midpoint are within 1 of the minimum.

Lemma 4.2.4 *Let uv be an edge with midpoint w such that when partitioned using shift values δ, $u \in S_{u'}$ and $v \in S_{v'}$. Then both dist$_{-\delta}(u', w)$ and dist$_{-\delta}(v', w)$ are within 1 of the minimum shifted distance to w.*

Proof Let the pieces that contain u and v be $S_{u'}$ and $S_{v'}$ respectively ($u' \neq v'$). Let the minimizer of $\text{dist}_{-\delta}(x, w)$ be w'. Since w is distance $1/2$ from both u and v, we have

$$\text{dist}_{-\delta}(w', u), \text{dist}_{-\delta}(w', v) \leq \text{dist}_{-\delta}(w', w) + 1/2.$$

Suppose $\text{dist}_{-\delta}(u', w) > \text{dist}_{-\delta}(w', w) + 1$, then we have:

$$\begin{aligned}\text{dist}_{-\delta}(u', u) &\geq \text{dist}_{-\delta}(u', w) - 1/2\\ &> \text{dist}_{-\delta}(w', w) + 1/2\\ &\geq \text{dist}_{-\delta}(w', u),\end{aligned}$$

a contradiction with u' being the minimizer of $\text{dist}_{-\delta}(x, u)$. The case with v follows similarly. ■

An even more accurate characterization of this situation can be obtained using the additional constraint that the shortest path from w' to w must go through one of u or v. However, this lemma suffices for abstracting the situation further to applying random decreases $\delta_1, \delta_2 \ldots \delta_n$ to a set of numbers $d_1 \ldots d_n$ corresponding to dist(x, w). We now turn our attention to analyzing the probability of another shifted value being close to the minimum when shifts are picked from the exponential distribution.

The memoryless property of the exponential distribution gives an intuitive way to bound this probability. Instead of considering the vertices picking their shift values independently, consider them as light bulbs with lifetime distributed according to *Exp*(β), and the d_is indicate the time each light bulb is being turned on. Then $\min_i d_i - \delta_i$ corresponds to the time when the last light bulb burns out, and we want to bound the time between that and the second last. In this setting, the memoryless property of exponentials gives that when the second to last light bulb fails, the behavior of the last light bulb does not change and its lifetime after that point still follows the same distribution. Therefore, the probability that the difference between these two is less than c can be bounded using the cumulative distribution function:

$$\begin{aligned}1 - \exp(-c\beta) &\approx 1 - (1 - c\beta) && \text{(When } c\beta \text{ is small)}\\ &= c\beta.\end{aligned}$$

The only case that is not covered here is when the last light bulb has not been turned on yet when the second last failed. However, in that case this probability can only be less. Below we give an algebraic proof of this.

Lemma 4.2.5 *Let $d_1 \leq \ldots \leq d_n$ be arbitrary values and $\delta_1 \ldots \delta_n$ be independent random variables picked from Exp(β). Then the probability that between the smallest and the second smallest values of $d_i - \delta_i$ are within c of each other is at most $O(\beta c)$.*

Proof

Let $\mathcal{E}$ denote the number of indices i such that:

$$d_i - \delta_i \leq d_j - \delta_j + c \,\forall j.$$

For each vertex i, let $\mathcal{E}_i$ be an indicator variable for the event that:

$$d_i - \delta_i \leq d_j - \delta_j + c \,\forall j.$$

We will integrate over the value of $t = d_i - \delta_i$. For a fixed value of t, $\mathcal{E}_i$ occurs if and only if $\delta_j \leq d_j - t + c$ for each j. As the shift values are picked independently, we can multiply the cumulative distribution functions for *Exp*(β) and get:

$$\begin{aligned}&\mathbf{Pr}\left[\mathcal{E}_i\right]\\ =&\int_{t=-\infty}^{\infty} f_{Exp}(d_i - t, \beta) \prod_{j \neq i} F_{Exp}(d_j - t + c, \beta)\end{aligned}$$

When $t > d_1 + c$, $d_1 - t + c < 0$ and $f_{Exp}(d_1 - t, \beta) = F_{Exp}(d_1 - t + c, \beta) = 0$. So it suffices to evaluate this integral up to $t = d_1 + c$. Also, we may use $\exp(-\beta x)$ as an upper bound as $f_{Exp}(x, \beta)$, and arrive at:

$$\begin{aligned}&\mathbf{Pr}\left[\mathcal{E}_i\right]\\ \leq&\int_{t=-\infty}^{d_1+c} \beta \exp(-\beta(d_i - t)) \prod_{j \neq i} F_{Exp}(d_j - t + c)\\ \leq&\int_{t=-\infty}^{d_1+c} \beta \exp(-\beta(d_i - t)) \prod_{j \neq i} (1 - \exp(-\beta(d_j - t + c))))\end{aligned}$$

We now bound $\mathbf{E}[\mathcal{E}] = \mathbf{E}[\sum_i \mathcal{E}_i]$. By linearity of expectation we have:

$$\begin{aligned}
\mathbf{E}\left[\sum_i \mathcal{E}_i\right] &\leq \sum_i \int_{t=-\infty}^{d_1+c} \beta \exp\left(-\beta(d_i - t)\right) \\
&\qquad \prod_{j \neq i} \left(1 - \exp(-\beta(d_j - (t - c)))\right) \\
&= \exp(\beta c) \int_{t=-\infty}^{d_1+c} \beta \sum_i \exp\left(-\beta(d_i - t + c)\right) \\
&\qquad \prod_{j \neq i} \left(1 - \exp(-\beta(d_j - t + c))\right)
\end{aligned}$$

Observe that the expression being integrated is the derivative w.r.t. t of:

$$-\prod_i \left(1 - \exp(-\beta(d_i - t + c))\right)$$

Therefore we get:

$$\mathbf{E}[\mathcal{E}] \leq -\exp(\beta c) \prod_i \left(1 - \exp(-\beta(d_i - t + c))\right)\Bigg|_{t=-\infty}^{t=d_1+c}$$

When $t \to -\infty$, $-\beta(d_i - t + c) \to -\infty$. Therefore $\exp(-\beta(d_i - t + c)) \to 0$, and the overall product tends to $-\exp(\beta c)$.

When $t = d_1 + c$, we have:

$$\begin{aligned}
&-\exp(\beta c) \prod_i \left(1 - \exp(-\beta(d_i - (d_1 + c) + c))\right) \\
&= -\exp(\beta c) \prod_i \left(1 - \exp(-\beta(d_i - d_1))\right) \\
&\leq -\exp(\beta c) \prod_i \left(1 - \exp(0)\right) = 0 \qquad \text{Since } d_i \geq d_1
\end{aligned}$$

Combining these two gives $\mathbf{E}[\mathcal{E}] \leq \exp(\beta c)$.

By Markov's inequality the probability of there being another vertex being within c of the minimum is at most $\exp(\beta c) - 1 \leq O(\beta c)$ for $c = 1$. ■

Using this Lemma with $c = 1$ and applying linearity of expectation gives the bound on the number of edges between pieces.

Corollary 4.2.6 *The probability of an edge $e = uv$ having u and v in different pieces is bounded by $O(\beta)$, and the expected number of edges between pieces is $O(\beta m)$.*

4.2.3 Parallel Implementation in Unweighted Case

Our partition routine as described in Algorithm 5 requires computing $\text{dist}_{-\delta}(u, v)$ for all pairs of vertices u and v. Standard modifications allow us to simplify it to the form shown in Algorithm 6, which computes BFS involving small integer distances.

Algorithm 6 Parallel Partition Algorithm

```
PARALLEL-PARTITION
Input: Undirected, unweighted graph $G = (V, E)$, parameter $0 < \beta < 1$ and parameter $d$ indicating
failure probability.
Output: $(\beta, O(\log n/\beta))$ decomposition of $G$ with probability at least $1 - n^{-d}$.
1: IN PARALLEL each vertex $u$ picks $\delta_u$ independently from an exponential distribution with
   mean $1/\beta$.
2: IN PARALLEL compute $\delta_{\max} = \max\{\delta_u \mid u \in V\}$
3: Perform PARALLEL BFS, with vertex $u$ starting when the vertex at the head of the queue has
   distance more than $\delta_{\max} - \delta_u$.
4: IN PARALLEL Assign each vertex $u$ to point of origin of the shortest path that reached it in the
   BFS.
```

The first observation is that the $-\delta_u$ shift at vertex u can be simulated by introducing a super source s with distance $-\delta_u$ to each vertex u. Then if we compute single source shortest path from s to all vertices, the component that v belongs to is given by the first vertex on the shortest path from s to it. Two more observations are needed to transform this shortest path setup to a BFS. First, the negative lengths on edges leaving s can be fixed by adding $\delta_{\max} = \max_u \delta_u$ to all these weights. Second, note that the only edges with non-integral lengths are the ones leaving s. In this shortest path algorithm, the only time that we need to examine the non-integer parts of lengths is when we compare two distances whose integer parts are tied. So the fractional parts can be viewed as tie-breakers for equal integer distances, and all distances with the same integer part can be processed in parallel. We'll show below that these tie breakers can also be replaced by a random permutation of integers.

Therefore, the algorithm is equivalent to computing shortest path when all edge lengths are integer, with an extra tie breaking rule for comparing distances. In order to use unweighted BFS, it remains to handle the edges with non-unit lengths leaving s, and we do so by processing the those edges in a delayed manner. An edge from s to v only causes v to be added to the BFS queue when the frontier of the search has reached a distance larger than the length of that edge and v has not been visited yet. So it suffices to check all vertices v with a length L edge to s when the BFS frontier moves to distance L, and add the unvisited ones to that level.

The exact cost of running a parallel breadth first search depends on the model of parallelism. There has been much practical work on such routines when the graph has small diameter [LS10, BAP12, SB13]. For simplicity we will use the $O(\Delta \log n)$ depth and $O(m)$ work bound in the PRAM model by Klein and Subramanian [KS97]. Here Δ is the maximum distance that we run the BFS to, and can be bounded by $O(\frac{\log n}{\beta})$. In the PRAM model, our result can be described by the following theorem:

Theorem 4.2.7 *There is an algorithm* PARTITION *that takes an unweighted graph with n vertices, m edges, a parameter $\beta \leq 1/2$ and produces a $(\beta, O(\frac{\log n}{\beta}))$ decomposition in expected $O(\frac{\log^2 n}{\beta})$ depth and $O(m)$ work.*

Proof Consider running PARTITION using the BFS based implementation described above, and repeating until we have an $(\beta, O(\frac{\log n}{\beta}))$ partition. Since the δ_us are generated independently, they can be computed in $O(n)$ work and $O(1)$ depth in parallel. The rest of the running time comes from assigning vertices to pieces using shifted shortest path. As the maximum distance from a vertex to the center of its piece is $O(\frac{\log n}{\beta})$ (or we could stop the algorithm at this point), this BFS can be done in $O(m)$ work and $O(\frac{\log^2 n}{\beta})$ depth using parallel BFS algorithms. The resulting decomposition can also be verified in $O(\log n)$ depth and $O(m)$ time.

It remains to bound the success probability of each iteration. Lemma 4.2.2 gives that the shortest path from u to any $v \in S_u$ is contained in S_u, so $\max_{v \in S_u} \text{dist}(u, v)$ is an upper bound for the strong diameter of each subset. For each vertex $v \in S_u$, since $\text{dist}_{-\delta}(v, v) = d(v, v) - \delta_v \leq 0$ is a candidate, $\text{dist}_{-\delta}(u, v) \leq -\delta_v$. Lemma 4.2.3 then allows us to bound this value by $O(\log n/\beta)$ with high probability. The expected number of edges between pieces follows from Corollary 4.2.6, so with constant probability we meet both requirements of a $(\beta, O(\frac{\log n}{\beta}))$ partition. Therefore, we are expected to iterate a constant number of times, giving the expected depth and work bounds. ■

4.3 Parallel Construction of Low Stretch Subgraphs

We now give a parallel algorithm for constructing the low-stretch subgraphs needed for the low-quality sparsifiers from Section 3.4. These subgraphs certify that most of the edges have small stretch w.r.t. a forest. It computes this forest $T \subseteq G$ along with stretch bounds of the edges that do have small stretch w.r.t. T. It also identifies the edges whose stretch we cannot bound, aka. a subset of "extra" edges $\bar{E}$. In other words, it produces a vector $\widehat{\text{str}} : E \setminus \bar{E} \rightarrow \mathbb{R}_+$, for the stretch of edges in $E \setminus \bar{E}$.

In general, the exact values of stretches with respect to a forest T can be computed in parallel using the tree contraction framework from [MR89], but having these upper bounds allows us to simplify the presentation of our overall algorithm.

Our algorithm is directly motivated by the AKPW low-stretch spanning tree algorithm [AKPW95], but is significantly simpler. As a motivating example, note that if all edges are unweighted, performing one step of low diameter decompositions as given in Theorem 4.2.7 gives:

- $O(\beta m)$ edges between the pieces
- All edges within pieces have stretch $O(\beta^{-1} \log n)$.

In order to handle varying weights, note that the diameter of pieces after one level of partition is also $O(\beta^{-1} \log n)$. Therefore if all pieces are shrunk into vertices, the cost of traversing a vertex is identical to traversing an edge of weight $O(\beta^{-1}\log n)$. Therefore, we can consider all edges in geometrically increasing buckets of weights, and at each level consider edges whose weights are comparable to the diameter of our 'vertices'.

Pseudocode of our algorithm for generating low stretch subgraphs is given in Algorithm 7 is a crude version of the AKPW algorithm. In words, iteration t of Algorithm 7 looks at a graph $(V^{(t)}, E^{(t)})$ which is a minor of the original graph (because components were contracted in previous iterations, and because it only considers the edges in the first j weight classes). It uses $\textsc{Partition}(V^{(t)}, E_t, \gamma)$ to decompose this graph into components such at most $\gamma|E_t|$ edges are between pieces. These edges are then added to the list of extra edges $\bar{E}$. Each such piece is then shrunk into a single node while adding a BFS tree on that component to T, and the algorithm iterates once again on the resulting graph. Adding these BFS trees maintains the invariant that the set of original nodes that have been contracted into a (super-)node in the current graph are connected in T. This means upon termination T is a forest.

Algorithm 7 Low-stretch Spanning Subgraph Algorithm

```
LSSG-AKPW
Input: Weighted graph $G = (V, E, \mathbf{w})$ with $n$ vertices, edge reduction factor $\gamma$. Routine $\textsc{Partition}$ that produces a $(\beta, y\beta^{-1})$-low diameter decomposition for graphs with $n$ vertices, where $y$ is a known parameter dependent on $n$ ($O(\log n)$).
Output: Forest $T$, extra edges $\bar{E}$ and upper bounds for stretch w.r.t. $T$, $\widehat{\mathsf{str}}$ for edges in $E \setminus \bar{E}$.

  Normalize the edges so that $\min\{\mathbf{w}_e : e \in E\} = 1$.
  $z \leftarrow 4y\gamma^{-1}$
  Initialize $T \leftarrow \emptyset$, $\bar{E} \leftarrow \emptyset$
  Divide $E$ into $E_1, E_2, \ldots$, where $E_i = \{e \in E \mid \mathbf{w}_e \in [z^{i-1}, z^i)\}$
  for $t = 1, 2, \ldots$, until the graph is exhausted do
    Create $E_i^{(t)}$ from $E_i$ by taking into account the contractions used to form $V^{(t)}$.
    $(C_1, C_2, \ldots, C_p) = \textsc{Partition}((V^{(t)}, E_i^{(t)}), z/4)$.
    Add a BFS tree of each component $C_i$ to $T$.
    Set $\widehat{\mathsf{str}}_e$ to $2z^2$ for all edges with endpoints in the same piece
    Add all edges with endpoints in different pieces to $\bar{E}$
    Create vertex set $V^{(t+1)}$ by contracting all edges within the components and removing all self-loops (but keeping parallel edges).
  end for
  Output forest $T$, extra edges $\bar{E}$ and stretch upper bounds $\widehat{\mathsf{str}}$.
```

We begin the analysis of the total stretch and running time by proving two useful facts:

Fact 4.3.1

$$\left|\bar{E}\right| \leq \gamma\left|E\right|$$

Proof The guarantee of PARTITION gives that at most γ fraction of the edges of E_t remain between the pieces after iteration t. As these are the only edges added to $\bar{E}$, summing over the iterations gives the bound. ■

Therefore, it suffices to bound the stretch of edges of $E_t^{(t)}$ that are contained in a single piece. The hop radius of the piece can be bounded by $y\gamma^{-1} \leq z/4$, and because all edges in E_t have weights within a factor of z of each other, there is hope that a $O(z^2)$ bound on stretch can suffice. However, each vertex in G^t corresponds to a set of vertices in G, and distances between them are omitted in this calculation. As a result, we need to start by bounding the weighted radius of each of the pieces.

Fact 4.3.2 *In iteration t, the weighted diameter of a component in the expanded-out graph is at most z^{t+1}.*

Proof The proof is by induction on j. First, note that each of the clusters computed in any iteration j has edge-count radius at most $z/4$. Now the base case $j = 1$ follows by noting that each edge in E_1 has weight less than z, giving a radius of at most $z^2/4 < z^{j+1}$. Now assume inductively that the radius in iteration $j - 1$ is at most z^j. Now any path with $z/4$ edges from the center to some node in the contracted graph will pass through at most $z/4$ edges of weight at most z^j, and at most $z/4 + 1$ supernodes, each of which adds a distance of $2z^j$; hence, the new radius is at most $z^{j+1}/4 + (z/4 + 1)2z^j \leq z^{j+1}$ as long as $z \geq 8$. ■

Coupling this with the lower bound on edge weights in bucket E_t allows to prove that $\widehat{\mathsf{str}}$ in fact upper bounds edge stretch:

Lemma 4.3.3 *For any edge e, $\textbf{str}_T(e) \leq \widehat{\mathsf{str}}(e)$*

Proof Let e be an edge in E_t which is contained in a piece contracted during iteration t The initial partition gives $w(e) > z^{t-1}$. By Fact 4.3.2, the path connecting the two endpoints of e in F has distance at most $2z^{t+1}$. Thus, $\textbf{str}_T(e) \leq 2z^{t+1}/z^{t-1} = 2z^2$. ■

Combining these facts means that LSSG-AKPW obtains a low-stretch subgraph with $n - 1 + \gamma m$ edges such that all off tree edges have stretch $z^2 = O\left(\gamma^{-2} \log^2 n\right)$. A drawback of this algorithm is that it performs $\log \Delta$ calls to PARTITION, where Δ is the ratio between the maximum and minimum edges weights in $\mathbf{w}$. We next show that by ignoring some of the buckets, we can run this algorithm on the bucketing scheme in parallel to reduce the depth.

4.3.1 Reducing Depth to Polylog

The depth of the LSSG-AKPW algorithm still depends on $\log \Delta$, and the reason is straightforward: the graph $G^{(t)}$ used in iteration t is built by taking $G^{(1)}$ and contracting edges in each iteration—hence, it depends on all previous iterations. However, note that for edges that are lighter by a factor of n can only affect the final stretch by a factor of $1/n$. Therefore, if a significant range of weight classes are empty, we can break this chain of dependencies. To create such empty weight classes, it suffices to identify edge classes totaling fewer than γm edges, and move them to $\bar{E}$ directly.

Consider a graph $G = (V, E, \mathbf{w})$ with edge weights $\mathbf{w}(e) \geq 1$, and let $E_i(G) := \{e \in E(G) \mid \mathbf{w}(e) \in [z^{i-1}, z^i)\}$ be the weight classes. Then, G is called (γ, τ)-*well-spaced* if there is a set of *special* weight classes $\{E_i(G)\}_{i \in I}$ such that for each $i \in I$, (a) there are at most γ weight classes before the following special weight class $\min\{i' \in I \cup \{\infty\} \mid i' > i\}$, and (b) the τ weight classes $E_{i-1}(G), E_{i-2}(G), \ldots, E_{i-\tau}(G)$ preceding i are all empty.

Lemma 4.3.4 *Given any graph $G = (V, E)$, $\tau \in \mathbb{Z}_+$, and $\gamma \leq 1$, there exists a graph $G' = (V, E')$ which is $(4\tau/\gamma, \tau)$-well-spaced, and $|E \setminus E'| \leq \gamma \cdot |E|$. Moreover, G' can be constructed in $O(m)$ work and $O(\log n)$ depth.*

Proof Let $\delta = \frac{\log \Delta}{\log z}$; note that the edge classes for G are $E_1, \ldots, E_\delta$, some of which may be empty. Denote by E_J the union $\cup_{i \in J} E_i$. We construct G' as follows: Divide these edge classes into disjoint groups $J_1, J_2, \ldots \subseteq [\delta]$, where each group consists of $\lceil \tau/\gamma \rceil$ consecutive classes. Within a group J_i, by an averaging argument, there must be a range $L_i \subseteq J_i$ of τ *consecutive* edge classes that contains at most a γ fraction of all the edges in this group, i.e., $|E_{L_i}| \leq \gamma \cdot |E_{J_i}|$ and $|L_i| \geq \tau$. We form G' by removing these the edges in all these groups L_i's from G, i.e., $G' = (V, E \setminus (\cup_i E_{L_i}))$. This removes only a γ fraction of all the edges of the graph.

We claim G' is $(4\tau/\gamma, \tau)$-well-spaced. Indeed, if we remove the group L_i, then we designate the smallest $j \in [\delta]$ such that $j > \max\{j' \in L_i\}$ as a special bucket (if such a j exists). Since we removed the edges in E_{L_i}, the second condition for being well-spaced follows. Moreover, the number of buckets between a special bucket and the following one is at most

$$2\lceil \tau/\gamma \rceil - (\tau - 1) \leq 4\tau/\gamma.$$

Finally, these computations can be done in $O(m)$ work and $O(\log n)$ depth using standard techniques such as the ones described in [JáJ92] and [Lei92]. ■

Therefore, we will construct LSSGs separately on each of the intervals in parallel, and merge the results together. This gives our main result for low-stretch subgraphs:

Theorem 4.3.5 (Low-Stretch Subgraphs) *There is an algorithm* LSSUBGRAPH(G, γ) *that for any weighted graph $G = (V, E, \boldsymbol{w})$, and any parameter γ, finds a forest T and a subset of edges $\bar{E} \subseteq E(G)$ along with upper bounds on stretch $\widehat{\mathsf{str}}$ for all edges $e \in E(G) \setminus \bar{E}$ such that*

1. *$|\bar{E}| \leq O(\gamma m)$.*
2. *For each edge $e \in E(G) \setminus \bar{E}$, $\boldsymbol{str}_T(e) \leq \widehat{\mathsf{str}}_e$; and*
3. *$\sum_{e \in E(G) \setminus \bar{E}} \widehat{\mathsf{str}}(e) \leq O\left(m\gamma^{-2} \log^2 n\right)$,*

Moreover, the procedure runs in $O(m)$ work and $O(\gamma^{-2} \log^3 n)$ depth.

Proof We set $\tau = 3 \log n$ and apply Lemma 4.3.4 to delete at most γm edges to form G'. This leaves us with a $(3 \log n\gamma^{-1}, 3 \log n)$-well-spaced graph G'.

For each special bucket i, we implicitly find the graph formed by contracting all edges in earlier buckets. Then we run LSSG-AKPW on this graph consisting of edges until the next special bucket. Afterwards, we add some of the extra edges to the forest so that its connected components match the one that results from contracting all edges up until the next special bucket. The first step can be done using a minimum spanning tree of the entire graph. The second step is done by performing $\log n\gamma^{-1}$ iterations of PARTITION one after another, each involving a different set of edges. Combining these over all edges gives $O(m)$ work and $O(\gamma^{-2} \log^3 n)$ depth. The final step is akin to finding connected components, and can be done in $O(m)$ work and $O(\log^2 n)$ depth, which is a lower order term.

To bound stretch, note that the bucketing scheme means that edges contracted have weight at most $z^{-3 \log n} \leq n^{-2}$ times the weight of an edge processed in this batch. As the path in the resulting tree contains at most n edges, these edges can result in an additive stretch of at most n^{-1}. So the total stretch can still be bounded by $O\left(m\gamma^{-2} \log^2 n\right)$.

Finally, the number of edges in $\bar{E} = \bar{E}' \cup (E(G) \setminus E(G'))$ is at most $2\gamma m$ by summing the two bounds. ■

For different choices of parameters, this algorithm was used to to generate the AKPW low-stretch spanning trees [AKPW95]. The resulting algorithm has a logarithmic dependency on Δ, as well as an n^{ϵ} term. It was presented along with the first polylog depth low diameter decomposition algorithms in [BGK$^+$13], and can be found in more details in [Tan11].

This particular embedding is needed in Section 3.4 to reduce the number of edges in the graph. Combining it with the ultrasparsifier construction gives the following crude sparsification:

Lemma 3.4.8 (Crude Sparsification) *Given any weighted graph $G = (V, E, \boldsymbol{w})$, and any parameter γ, we can find in $O(\gamma^{-2} \log^3 n)$ depth and $O(m \log n)$ work a graph H on the same vertex set with $n - 1 + \gamma m$ edges such that:*

$$\boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq O\left(\gamma^{-2} \log^3 n\right) \boldsymbol{L}_G$$

The proof of this Lemma can be readily obtained by combining Theorem 4.3.5 with a parallelization of the ultrasparsifiers from Theorem 2.2.4.

Theorem 2.2.4 (Ultra-Sparsifier Construction) *There exists a setting of constants in* ULTRASPARSIFY *such that given a graph G and a spanning tree T,* ULTRASPARSIFY*(G, T) returns in $O(m \log n + \frac{\boldsymbol{str}_T(G) \log^2 n}{\kappa})$ time a graph H that is with high probability a $\left(\kappa, O\left(\frac{\boldsymbol{str}_T(G)}{m} \log n\right)\right)$-ultra-sparsifier for G.*

Proof of Lemma 3.4.8:

Applying Theorem 4.3.5 with $\frac{\gamma}{2}$ gives set of extra edges $\bar{E}$ and a spanning tree T such that $\mathbf{str}_T(E \setminus \bar{E}) \leq m\gamma^{-2}\log^2 n$. Since the sampling steps in ULTRASPARSIFY are performed independently, it parallelizes to $O\left(\log^2 n\right)$ depth without work overhead. Running it on $E \setminus \bar{E}$ for a κ to be determined gives a graph H with $n-1+\frac{\mathbf{str}_T(E \setminus \bar{E})}{\kappa}\log n = n-1+\frac{4\gamma^{-2}\log^3 n}{\kappa}$ edges such that:

$$\mathbf{L}_G \preceq \mathbf{L}_H \preceq \kappa\mathbf{L}_G$$

Adding $\bar{E}$ to H then leads to a graph with $n-1+\left(\frac{\gamma}{2}+\frac{4\gamma^{-2}\log^3 n}{\kappa}\right)m$. Therefore if we set κ to $8\gamma^{-3}\log^3 n$, the number of edges in this graph is $n-1+\gamma m$. The bound between G and H also follows from the guarantees of Theorem 2.2.4. ■

4.4 Construction of low ℓ_α-stretch spanning trees

We now give a sketch for generating low stretch spanning tree that leads to the $O\left(m\log n\log\left(1/\epsilon\right)\right)$ time solver algorithm from Section 2.5. The construction of low stretch spanning trees received significant attention due to their use in the nearly-linear time solvers for SDD linear systems [ST06]. The state of the art both in stretch and running time have been continuously improved [EEST08, ABN08, AN12]. As a result, a self-contained presentation of modifications to the algorithm falls outside of the scope of this thesis. Instead, we will sketch the necessary modifications needed for obtaining a spanning tree with small ℓ_α stretch.

Claim 4.4.1 *Let α be any constant such that $0 \leq \alpha < 1$. Given weighted graph $G=(V,E,\boldsymbol{w})$, we can find in $O\left(m\log n\log\log n\right)$ time a spanning tree T such that:*

$$\sum_e\left(\frac{\boldsymbol{str}_T(e)}{\log n}\right)^\alpha \leq O(m)$$

To date, all algorithms that generate spanning trees with polylog average stretch rely on the star-decomposition framework [EEST08, ABN08, AN12]. The framework, introduced by Elkin et al. [EEST08], aims to partition into a graph into pieces whose diameters are smaller by a constant factor. As a result, the diameter of the pieces, Δ decreases geometrically. Furthermore, contracting edges shorter than $\frac{\Delta}{n^3}$ ensures that each edge is only involved in $O(\log n)$ levels of partitioning [EEST08]. The stretch bound of edges cut then relies on two steps:

1. Let E_{cut} be the set of edges between the pieces. One needs to show that $\sum_{e\in E_{cut}}\frac{1}{\mathbf{w}_e} \leq \frac{p\cdot m}{\Delta}$, where m is the number of edges involved.

2. The final diameter of the tree produced on a graph with diameter Δ is $O(\Delta)$.

The second step is the main focus in the recent advancements by Abraham et al. [ABN08, AN12]. Assuming this step, it's easy to see that the total stretch can be bounded by $O(p\log nm)$. This $O(\log n)$ factor from all the levels was reduced to $O(\log\log n)$ using an approach by Seymour [Sey95]. However, using simpler schemes such as probabilistic decompositions [Bar96], one can obtain an algorithm with $p = O(\log n)$ that does not have the guarantees across all the layers. This leads to a total stretch of $O(m \log^2 n)$. However, probabilistic decompositions have the even stronger property that an edge e with weight $\mathbf{w}_e$ is cut with probability at most:

$$\frac{\mathbf{w}_e}{\Delta} \log n$$

We can show that as long as higher stretch values are discounted, this suffices for the required low moment stretch.

Lemma 4.4.2 *Let α be a constant that's strictly less than 1. Consider an edge weight $\boldsymbol{w}_e$ and a sequence of diameter values $\Delta_1 \ldots \Delta_l$ such that $\Delta_i \geq c^i \boldsymbol{w}_e \log n$ for some constant $c > 1$. If e is cut in the decomposition at level i with probability at most $\frac{\boldsymbol{w}_e}{\Delta_i} \log n$, then:*

$$\mathbf{E}\left[\left(\frac{\boldsymbol{str}_T(e)}{\log n}\right)^{\alpha}\right] \leq O\left(1\right)$$

Proof By linearity of expectation, we can sum over all the levels:

$$\begin{aligned}
\mathbf{E}\left[\left(\frac{\mathbf{str}_T(e)}{\log n}\right)^{\alpha}\right] &= \sum_{i=1}^{l} \left(\frac{O(\Delta_i)}{\mathbf{w}_e \log n}\right)^{\alpha} \mathbf{Pr}\left[e \text{ cut at level } i\right] \\
&\leq \sum_{i=1}^{l} \left(\frac{O(\Delta_i)}{\mathbf{w}_e \log n}\right)^{\alpha} \frac{\mathbf{w}_e}{\Delta_i} \log n \\
&= O\left(\sum_{i=1}^{l} \left(\frac{\mathbf{w}_e \log n}{\Delta_i}\right)^{1-\alpha}\right)
\end{aligned}$$

Note that due to the geometrical decrease in diameters, Δ_i satisfies $\Delta_i \geq \mathbf{w}_e \log n c^i$, or alternatively $\frac{\mathbf{w}_e \log n}{\Delta_i} \leq c^{-i}$. Substituting this gives:

$$\begin{aligned}
\mathbf{E}\left[\left(\frac{\mathbf{str}_T(e)}{\log n}\right)^{\alpha}\right] &\leq O\left(\sum_{i=1}^{l} \left(\frac{1}{c^{1-\alpha}}\right)^{i}\right) \\
&= O\left(1\right)
\end{aligned}$$

■

Therefore replacing the more sophisticated decomposition schemes with probabilistic decompositions leads to spanning trees with low moment stretch. The running time of the algorithm follows similarly to Section 7 of the algorithm by Abraham and Neiman [AN12].

Chapter 5

Applications to Image Processing

In this chapter we show that SDD linear system solvers can be used as a subroutine to solve a wide range of regression problems. These problems are motivated by the LASSO framework and have applications in machine learning and computer vision. We show that most of these problems can be formulated as a grouped least squares problem. Our main result shown in Section 5.4 is an $O\left(m^{4/3}\mathbf{poly}\left(\epsilon^{-1}\right)\right)$ time algorithm for computing $(1+\epsilon)$-approximate solutions to this formulation. We also present some preliminary experiments using it on image processing tasks.

The problem of recovering a clear signal from noisy data is an important problem in signal processing. One general approach to this problem is to formulate an objective based on required properties of the answer, and then return its minimizer via optimization algorithms. The power of this method was first demonstrated in image denoising, where the total variation minimization approach by Rudin, Osher and Fatemi [ROF92] had much success. More recent works on sparse recovery led to the theory of compressed sensing [Can06], which includes approaches such as the least absolute shrinkage and selection operator (LASSO) objective due to Tibshirani [Tib96]. These objective functions have proven to be immensely powerful tools, applicable to problems in signal processing, statistics, and computer vision. In the most general form, given vector $\mathbf{y}$ and a matrix $\mathbf{A}$, one seeks to minimize:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \left\|\mathbf{y}-\mathbf{A}\mathbf{x}\right\|_2^2 \\ \text{subject to:} \quad & \left\|\mathbf{x}\right\|_1 \leq c \end{aligned} \tag{5.1}$$

It can be shown to be equivalent to the following by introducing a Lagrangian multiplier, λ:

$$\min_{\mathbf{x}} \quad \left\|\mathbf{y}-\mathbf{A}\mathbf{x}\right\|_2^2 + \lambda\left\|\mathbf{x}\right\|_1 \tag{5.2}$$

Many of the algorithms used to minimize the LASSO objective in practice are first order methods [Nes07, BCG11], which update a sequence of solutions using well-defined vectors related to the gradient. These methods converge well when the matrix $\mathbf{A}$ is well-structured. The formal definition of this well-structuredness is closely related to the conditions required by the guarantees given in the compressed sensing literature [Can06] for the recovery of a sparse signal. As a result, these algorithms perform very well on problems where theoretical guarantees for solution quality are known. This good performance, combined with the simplicity of implementation, makes these algorithms the method of choice for most problems.

However, LASSO-type approaches have also been successfully applied to larger classes of problems. This has in turn led to the use of these algorithms on a much wider variety of problem instances. An important case is image denoising, where works on LASSO-type objectives predates the compressed sensing literature [ROF92]. The matrices involved here are based on the connectivity of the underlying pixel structure, which is often a $\sqrt{n} \times \sqrt{n}$ square mesh. Even in an unweighted setting, these matrices tend to be ill-conditioned. In addition, the emergence of non-local formulations that can connect arbitrary pairs of vertices in the graph also highlights the need to handle problems that are traditionally considered ill-conditioned. We will show in Section 5.6 that the broadest definition of LASSO problems include well-studied problems from algorithmic graph theory:

Fact 5.0.1 *Both the s-t shortest path and s-t minimum cut problems in undirected graphs can be solved by minimizing a LASSO objective.*

Although linear time algorithms for unweighted shortest path are known, finding efficient parallel algorithms for this has been a long-standing open problem. The current state of the art parallel algorithm for finding $1 + \epsilon$ approximate solutions, due to Cohen [Coh00], is quite involved. Furthermore, these reductions are readily parallelizable, efficient algorithms for LASSO minimization would also lead to efficient parallel shortest path algorithms. This suggests that algorithms for minimizing LASSO objectives, where each iteration uses simple, parallelizable operations, are also difficult. Finding a minimum s-t cut with nearly-linear running time is also a long standing open question in algorithm design. In fact, there are known hard instances where many algorithms do exhibit their worst case behavior [JM93]. The difficulty of these problems and the non-linear nature of the objective are two of the main challenges in obtaining fast run time guarantees.

Previous run time guarantees for minimizing LASSO objectives rely on general convex optimization routines [BV04], which take at least $\Omega(n^2)$ time. As the resolution of images are typically at least 256×256, this running time is prohibitive. As a result, when processing image streams or videos in real time, gradient descent or filtering based approaches are typically used due to time constraints, often at the cost of solution quality. The continuing increase in problem instance size, due to higher resolution of streaming videos, or 3D medical images with billions of voxels, makes the study of faster algorithms an increasingly important question.

While the connection between LASSO and graph problems gives us reasons to believe that the difficulty of graph problems also exists in minimizing LASSO objectives, it also suggests that techniques from algorithmic graph theory can be brought to bear. To this end, we draw upon recent developments in algorithms for maximum flow [CKM$^+$11] and minimum cost flow [DS08]. We show that relatively direct modifications of these algorithms allows us to solve a generalization of most LASSO objectives, which we term the **grouped least squares problem**. Our algorithm is similar to convex optimization algorithms in that each iteration of it solves a quadratic minimization problem, which is equivalent to solving a linear system. The speedup over previous algorithms come from the existence of much faster solvers for graph related linear systems [ST06], although our approaches are also applicable to situations involving other underlying quadratic minimization problems.

5.1 Background and Formulations

The formulation of our main problem is motivated by the total variation objective from image denoising. This objective has its origin in the seminal work by Mumford and Shah [MS89]. There are two conflicting goals in recovering a smooth image from a noisy one, namely that it must be close to the original image, while having very little noise. The Mumford-Shah function models the second constraint by imposing penalties for neighboring pixels that differ significantly. These terms decrease with the removal of local distortions, offsetting the higher cost of moving further away from the input. However, the minimization of this functional is computationally difficult and subsequent works focused on minimizing functions that are close to it.

The total variation objective is defined for a discrete, pixel representation of the image and measures noise using a smoothness term calculated from differences between neighboring pixels. This objective leads naturally to a graph $G = (V, E)$ corresponding to the image with pixels. The original (noisy) image is given as a vertex labeling **s**, while the goal of the optimization problem is to recover the 'true' image **x**, which is another set of vertex labels. The requirement of **x** being close to **s** is quantified by $\|\mathbf{x} - \mathbf{s}\|_2^2$, which is the square of the ℓ_2 norm of the vector that's identified as noise. To this is added the smoothness term, which is a sum over absolute values of difference between adjacent pixels' labels:

$$\|\mathbf{x} - \mathbf{s}\|_2^2 + \sum_{(u,v)\in E} |x_u - x_v| \tag{5.3}$$

This objective can be viewed as an instance of the fused LASSO objective [TSR$^+$05]. As the orientation of the underlying pixel grid is artificially imposed by the camera, this method can introduce rotational bias in its output. One way to correct this bias is to group the differences of each pixel with its 4 neighbors. For simplicity, we break symmetry by only consider the top and right neighbors, giving terms of the form:

$$\sqrt{(x_u - x_v)^2 + (x_u - x_w)^2} \tag{5.4}$$

where v and w are the horizontal and vertical neighbor of u.

Our generalization of these objectives is based on the key observation that $\sqrt{(x_u - x_v)^2 + (x_u - x_w)^2}$ and $|x_u - x_v|$ are both ℓ_2 norms of vectors consisting of differences of values between adjacent pixels. Each such difference can be viewed as an edge in the underlying graph, and the grouping gives a natural partition of the edges into disjoint sets $S_1 \dots S_k$:

$$\|\mathbf{x} - \mathbf{s}\|_2^2 + \sum_{1\le i\le k} \sqrt{\sum_{(u,v)\in S_i} (x_u - x_v)^2} \tag{5.5}$$

When each S_i contain a single edge, this formulation is identical to the objective in Equation 5.3 since $\sqrt{(x_u - x_v)^2} = |x_u - x_v|$. To make the first term resemble the other terms in our objective, we will take a square root of it – as we prove in Appendix 5.7, algorithms that give exact minimizers for this variant still captures the original version of the problem. Also, the terms inside the square roots can be written as quadratic positive semi-definite terms involving **x**. The simplified problem now becomes:

$$\|\mathbf{x} - \mathbf{s}\|_2 + \sum_{1 \le i \le k} \sqrt{\mathbf{x}^T \mathbf{L}_i \mathbf{x}} \tag{5.6}$$

We can also use $\|\cdot\|_{\mathbf{L}_i}$ to denote the norm induced by the PSD matrix $\mathbf{L}_i$, and rewrite each of the later terms as $\|\mathbf{x}\|_{\mathbf{L}_i}$. Fixed labels $\mathbf{s}_1 \dots \mathbf{s}_k$ can also be introduced for each of the groups, with roles similar to **s** (which becomes $\mathbf{s}_0$ in this notation). As the ℓ_2 norm is equivalent to the norm given by the identity matrix, $\|\mathbf{x} - \mathbf{s}_0\|_2$ is also a term of the form $\|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$. These generalizations allows us to define our main problem:

Definition 5.1.1 *The* **grouped least squares problem** *is:*

Input: $n \times n$ matrices $\boldsymbol{L}_1 \dots \boldsymbol{L}_k$ and fixed values $\boldsymbol{s}_1 \dots \boldsymbol{s}_k \in \mathbb{R}^n$.

Output:

$$\min_{\boldsymbol{x}} \mathcal{OBJ}(\boldsymbol{x}) = \sum_{1 \le i \le k} \|\boldsymbol{x} - \boldsymbol{s}_i\|_{\boldsymbol{L}_i}$$

Note that this objective allows for the usual definition of LASSO involving terms of the form $|\mathbf{x}_u|$ by having one group for each such variable with $\mathbf{s}_i = \mathbf{0}$. It is also related to group LASSO [YL06], which incorporates similar assumptions about closer dependencies among some of the terms. To our knowledge grouping has not been studied in conjunction with fused LASSO, although many problems such as the ones listed in Section 5.2 require this generalization.

5.1.1 Quadratic Minimization and Solving Linear Systems

Our algorithmic approach to the group least squares problem crucially depends on solving a related quadratic minimization problem. Specifically, we solve linear systems involving a weighted combination of the $\mathbf{L}_i$ matrices. Let $\mathbf{w}_1 \dots \mathbf{w}_k \in \mathbb{R}^+$ denote weights, where $\mathbf{w}_i$ is the weight on the ith group. Then the quadratic minimization problem that we consider is:

$$\min_{\mathbf{x}} \mathcal{OBJ}2(\mathbf{x}, \mathbf{w}) = \sum_{1 \le i \le k} \frac{1}{\mathbf{w}_i} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}^2 \tag{5.7}$$

We will use OPT2($\mathbf{w}$) to denote the minimum value that is attainable. This minimizer, **x**, can be obtained using the following Lemma:

Lemma 5.1.2 *$\mathcal{OBJ}2(\boldsymbol{x},\boldsymbol{w})$ is minimized for $\boldsymbol{x}$ such that*

$$\left(\sum_{1\le i\le k}\frac{1}{\boldsymbol{w}_i}\boldsymbol{L}_i\right)\boldsymbol{x}=\sum_{1\le i\le k}\frac{1}{\boldsymbol{w}_i}\boldsymbol{L}_i\boldsymbol{s}_i$$

Therefore the quadratic minimization problem reduces to a linear system solve involving $\sum_i \frac{1}{\mathbf{w}_i}\mathbf{L}_i$, or $\sum_i \alpha_i \mathbf{L}_i$ where α is an arbitrary set of positive coefficients. In general, this can be done in $O(n^{\omega})$ time where ω is the matrix multiplication constant [Str69, CW90, Wil12]. When $\mathbf{L}_i$ is symmetric diagonally dominant, which is the case for image applications and most graph problems, these systems can be approximately solved to ϵ accuracy in $\tilde{O}(m\log(1/\epsilon))$ time [1], where m is the total number of non-zero entries in the matrices [ST06, KMP11], and also in $\tilde{O}(m^{1/3+\theta}\log(1/\epsilon))$ parallel depth [BGK$^+$13]. There has also been work on extending this type of approach to a wider class of systems [AST09], with works on systems arising from well-spaced finite-element meshes [BHV08], 2-D trusses [DS07], and certain types of quadratically coupled flows [KMP12]. For the analysis of our algorithms, we treat this step as a black box with running time $T(n,m)$. Furthermore, to simplify our presentation we assume that the solvers return exact answers, as errors can be brought to polynomially small values with an extra $O(\log n)$ overhead. We believe analyses similar to those performed in [CKM$^+$11, KMP12] can be adapted if we use approximate solvers instead of exact ones.

5.2 Applications

A variety of problems ranging from computer vision to statistics can be formulated as grouped least squares. We describe some of them below, starting with classical problems from image processing.

5.2.1 Total Variation Minimization

As mentioned earlier, one of the earliest applications of these objectives was in the context of image processing. More commonly known as total variation minimization in this setting [CS05], various variants of the objective have been proposed with the anisotropic objective the same as Equation 5.3 and the isotropic objective being the one shown in Equation 5.5.

Obtaining a unified algorithm for isotropic and anisotropic TV was one of the main motivations for our work. Our results lead to an algorithm that approximately minimizes both variants in $\tilde{O}(m^{4/3}\epsilon^{-8/3})$ time. This guarantee does not rely on the underlying structure of the graph, so the algorithm readily applicable to 3-D images or non-local models involving the addition of edges across the image. However, when the neighborhoods are those of a 2-D image, a $\log n$ factor speedup can be obtained by using the optimal solver for planar systems given in [KM07].

5.2.2 Denoising with Multiple Colors

Most works on image denoising deals with images where each pixel is described using a single number corresponding to its intensity. A natural extension would be to colored images, where

[1]We use $\tilde{O}(f(m))$ to denote $\tilde{O}(f(m)\log^c f(m))$ for some constant c.

each pixel has a set of c attributes (in the RGB case, $c = 3$). One possible analogue of $|x_i - x_j|$ in this case would be $||x_i - x_j||_2$, and this modification can be incorporated by replacing a cluster involving a single edge with clusters over the c edges between the corresponding pixels.

This type of approach can be viewed as an instance of image reconstruction algorithms using Markov random fields. Instead of labeling each vertex with a single attribute, a set of c attributes are used instead and the correlation between vertices is represented using arbitrary PSD matrices. When such matrices have bounded condition number, it was shown in [KMP12] that the resulting least squares problem can still be solved in nearly-linear time by preconditioning with SDD matrices.

5.2.3 Poisson Image Editing

The Poisson Image Editing method of Perez, Gangnet and Blake [PGB03] is a popular method for image blending. This method aims to minimize the difference between the gradient of the image and a guidance field vector $\mathbf{v}$. We show here that the grouped least square problem can be used for minimizing objectives from this framework. The objective function as given in equation (6) of [PGB03] is:

$$\min_{f|\Omega} \sum_{(p,q)\cap\Omega\neq\emptyset} \left(f_p - f_q - v_{pq}\right)^2, \text{ with } f_p = f_p^* \; \forall p \in \partial\Omega$$

where Ω is the domain and $\partial\Omega$ is the boundary. It is mainly comprised of terms of the form:

$$\left(x_p - x_q - v_{pq}\right)^2$$

This term can be rewritten as $((x_p - x_q) - (v_{pq} - 0))^2$. So if we let $\mathbf{s}_i$ be the vector where $s_{i,p} = v_{pq}$ and $s_{i,q} = 0$, and $\mathbf{L}_i$ be the graph Laplacian for the edge connecting p and q, then the term equals to $||\mathbf{x} - \mathbf{s}_i||_{\mathbf{L}_i}^2$. The other terms on the boundary will have x_q as a constant, leading to terms of the form $||x_{i,p} - s_{i,p}||_2^2$ where $s_{i,p} = x_q$. Therefore the discrete Poisson problem of minimizing the sum of these squares is an instance of the quadratic minimization problem as described in Section 5.1.1. Perez et al. in Section 2 of their paper observed that these linear systems are sparse, symmetric and positive definite. We make the additional observation here that the systems involved are also symmetric diagonally dominant. The use of the grouped least squares framework also allows the possibility of augmenting these objectives with additional L_1 or L_2 terms.

5.2.4 Clustering

Hocking et al. [HVBJ11] recently studied an approach for clustering points in d dimensional space. Given a set of points $x_1 \dots x_n \in \mathbb{R}^d$, one method that they proposed is the minimization of the following objective function:

$$\min_{y_1 \dots y_n \in \mathbb{R}^d} \sum_{i=1}^{n} ||x_i - y_i||_2^2 + \lambda \sum_{ij} w_{ij} ||y_i - y_j||_2$$

Where w_{ij} are weights indicating the association between items i and j. This problem can be viewed in the grouped least squares framework by viewing each x_i and y_i as a list of d variables, giving that the $||x_i - y_i||_2$ and $||y_i - y_j||_2$ terms can be represented using a cluster of d edges. Hocking et al. used the Frank-Wolfe algorithm to minimize a relaxed form of this objective and observed fast behavior in practice. In the grouped least squares framework, this problem is an instance with $O(n^2)$ groups and $O(dn^2)$ edges. Combining with the observation that the underlying quadratic optimization problems can be solved efficiently allows us to obtain an $1 + \epsilon$ approximate solution in $\tilde{O}(dn^{8/3}\epsilon^{-8/3})$ time.

5.3 Related Algorithms

Due to the importance of optimization problems motivated by LASSO there has been much work on efficient algorithms for minimizing the objectives described in Section 5.1. We briefly describe some of the previous approaches for LASSO minimization below.

5.3.1 Second-order Cone Programming

To the best of our knowledge, the only algorithms that provide robust worst-case bounds for the entire class of grouped least squares problems are based on applying tools from convex optimization. In particular, it is known that interior point methods applied to these problems converge in $\tilde{O}(\sqrt{k})$ iterations with each iterations requiring solving a linear system [BV04, GY04]. Unfortunately, computing these solutions is computationally expensive – the best previous bound for solving one of these systems is $O(m^{\omega})$ where ω is the matrix multiplication exponent. This results in a total running time of $O(m^{1/2+\omega})$, which in well structures situations can be lowered to $O(m^2)$. However, for images with m in the millions, it is still cost-prohibitive.

5.3.2 Graph Cuts

For the anisotropic total variation objective shown in Equation 5.3, a minimizer can be found by solving a large number of almost-exact maximum flow calls [DS06, KZ04]. Although the number of iterations can be large, these works show that the number of problem instances that a pixel can appear in is small. Combining this reduction with the fastest known exact algorithm for the maximum flow problem by Goldberg and Rao [GR98] gives an algorithm that runs in $\tilde{O}(m^{3/2})$ time.

Both of these algorithms requires extracting the minimum cut in order to construct the problems for subsequent iterations. As a result, it is not clear whether recent advances on fast approximations of maximum flow and minimum s-t cuts [CKM$^+$11] can be directly used as a black box with these algorithms. Extending this approach to the non-linear isotropic objective also appears to be difficult.

5.3.3 Iterative Reweighted Least Squares

An approach similar to convex optimization methods, but has much better observed rates of convergence is the iterative reweighted least squares (IRLS) method. This method does a much more aggressive adjustment each iteration and to give good performances in practice [WR07].

5.3.4 First Order Methods

The method of choice in practice are first order methods such as [Nes07, BCG11]. Theoretically these methods are known to converge rapidly when the objective function satisfies certain Lipschitz conditions. Many of the more recent works on first order methods focus on lowering the dependency of ϵ under these conditions. As the grouped least squares problem is a significantly more general formulation, this direction is somewhat different than our goals,

5.4 Approximating Grouped Least Squares Using Quadratic Minimization

We show an approximate algorithm for the grouped least squares problem. The analyses of the algorithms is intricate, but is closely based on the approximate minimum cut algorithm given by Christiano et al. [CKM$^+$11]. It has better error dependencies, but also takes much more time for moderate number of groups. Both of these algorithms can be viewed as reductions to the quadratic minimization problems described in Section 5.1.1. As a result, they imply efficient algorithms for problems where fast algorithms are known for the corresponding least squares problems.

Recall that the minimum s-t cut problem - equivalent to an ℓ_1-minimization problem - is a special case of the grouped least squares problem where each edge belongs to its own group (i.e., $k = m$). As a result, it's natural to extend the approach of [CKM$^+$11] to the whole spectrum of values of k by treating each group as an edge.

One view of the cut algorithm from [CKM$^+$11] is that it places a weight on each group, and minimizes a quadratic, or ℓ_2^2 problem involving terms of the from $\frac{1}{\mathbf{w}_i} \left\| \mathbf{x} - \mathbf{s}_i \right\|_{\mathbf{L}_i}^2$. Their algorithm then adjusts the weights based on the flow on each edge using the multiplicative weights update framework [AHK05, LW94]. This flow is in turn obtained from the dual of the quadratic minimization problem. We simplify this step by showing that the energy of the groups from the quadratic minimization problems can be directly used. Pseudocode of the algorithm is shown in Algorithm 1.

The main difficulty of analyzing this algorithm is that the analysis of minimum s-t cut algorithm of [CKM$^+$11] relies strongly on the existence of a solution where $\mathbf{x}$ is either 0 or 1. Our analysis extends this potential function into the fractional setting using a potential function based on the Kulback-Liebler (KL) divergence [KL51]. To our knowledge the use of this potential with multiplicative weights was first introduced by Freund and Schapire [FS99], and is common in learning theory. This function can be viewed as measuring the KL-divergence between $\mathbf{w}_i^{(t)}$ and $\left\| \bar{\mathbf{x}} - \mathbf{s}_i \right\|_{\mathbf{L}_i}$ over all groups, where $\bar{\mathbf{x}}$ an optimum solution to the grouped least squares problem. This term, which we denote as D_{KL} is:

$$D_{KL} = \sum_i \left\| \bar{\mathbf{x}} - \mathbf{s}_i \right\|_{\mathbf{L}_i} \log \left(\frac{\left\| \bar{\mathbf{x}} - \mathbf{s}_i \right\|_{\mathbf{L}_i}}{\mathbf{w}_i^{(t)}} \right) \tag{5.8}$$

One way to interpret this function is that $\left\| \mathbf{x} - \mathbf{s}_i \right\|_{\mathbf{L}_i}$ and $\frac{1}{\mathbf{w}_i} \left\| \mathbf{x} - \mathbf{s}_i \right\|_{\mathbf{L}_i}^2$ are equal when $\mathbf{w}_i =$

Algorithm 1 Algorithm for the approximate decision problem of whether there exist vertex potentials with objective at most OPT

```
ApproxGroupedLeastSquares
Input: PSD matrices $\mathbf{L}_1 \dots \mathbf{L}_k$, fixed values $\mathbf{s}_1 \dots \mathbf{s}_k$ for each group. Routine Solve for solving linear systems, width parameter $\rho$ and error bound $\epsilon$.
Output: Vector $\mathbf{x}$ such that $\mathcal{OBJ}(\mathbf{x}) \le (1+10\epsilon)\text{OPT}$.

 1: Initialize $\mathbf{w}_i^{(0)} = 1$ for all $1 \le i \le k$
 2: $N \leftarrow 10\rho \log n \epsilon^{-2}$
 3: for $t = 1 \dots N$ do
 4:     $\mu^{(t-1)} \leftarrow \sum_i \mathbf{w}_i^{(t-1)}$
 5:     Use Solve as described in Lemma 5.1.2 to compute a minimizer for the quadratic minimization problem where $\alpha_i = \frac{1}{\mathbf{w}_i^{(t-1)}}$. Let this solution be $\mathbf{x}^{(t)}$
 6:     Let $\lambda^{(t)} = \sqrt{\mu^{(t-1)}\mathcal{OBJ}2(\mathbf{x}^{(t)})}$, where $\mathcal{OBJ}2(\mathbf{x}^{(t)})$ is the quadratic objective given in Equation 5.7
 7:     Update the weight of each group:
            $\mathbf{w}_i^{(t)} \leftarrow \mathbf{w}_i^{(t-1)} + \left(\frac{\epsilon}{\rho}\frac{\|\mathbf{x}^{(t)}-\mathbf{s}_i\|_{\mathbf{L}_i}}{\lambda^{(t)}} + \frac{2\epsilon^2}{k\rho}\right)\mu^{(t-1)}$
 8: end for
 9: $\bar{t} \leftarrow \arg\min_{0 \le t \le N} \mathcal{OBJ}(\mathbf{x}^{(t)})$
10: return $\mathbf{x}^{(\bar{t})}$
```

$\|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$. Therefore, this algorithm can be viewed as gradually adjusts the weights to become a scaled copy of $\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}$, and D_{KL} serves a way to measure this difference. It can be simplified by subtracting the constant term given by $\sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i} \log(\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i})$ and multiplying by $-1/\text{OPT}$. This gives us our key potential function, $\nu^{(t)}$:

$$\nu^{(t)} = \frac{1}{\text{OPT}} \sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i} \log(\mathbf{w}_i^{(t)}) \tag{5.9}$$

It's worth noting that in the case of the cut algorithm, this function is identical to the potential function used in [CKM+11]. We show the convergence of our algorithm by proving that if the solution produced in some iteration is far from optimal, $\nu^{(t)}$ increases substantially. Upper bounding it with a term related to the sum of weights, $\mu^{(t)}$ allows us to prove convergence.

To simplify the analysis, we that all entries of $\mathbf{s}$, the spectrum of $\mathbf{L}_i$, and OPT are polynomially bounded in n. That is, there exist some constant d such that $-n^d \le \mathbf{s}_{i,u} \le n^d$ and $n^{-d}\mathbf{I} \preceq \sum_i \mathbf{L}_i \preceq n^d\mathbf{I}$ where $\mathbf{A} \preceq \mathbf{B}$ means $\mathbf{B} - \mathbf{A}$ is PSD. Some of these assumptions can be relaxed via analyses similar to Section 2 of [CKM+11].

Theorem 5.4.1 *On input of an instance of $\mathcal{OBJ}$ with edges partitioned into k sets. If all parameters polynomially bounded between n^{-d} and n^d, running* ApproxGroupedLeastSquares *with*

$\rho = 2k^{1/3}\epsilon^{-2/3}$ *returns a solution* $\boldsymbol{x}$ *with such that* $\mathcal{OBJ}(\boldsymbol{x}) \leq \max\{(1+10\epsilon)OPT, n^{-d}\}$ *where OPT is the value of the optimum solution.*

The additive n^{-d} case is included to deal with the case where OPT $= 0$, or is close to it. We believe it should be also possible to handle this case by restricting the condition number of $\sum_i \mathbf{L}_i$.

For readers familiar with the analysis of the Christiano et al. algorithm [CKM+11], the following mapping of terminology in the analysis below might be useful:

- edge $e \rightarrow$ group i.
- flow along edge $e \rightarrow$ value of $\|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$.
- weight on edge $e \rightarrow$ weight of group i, $\mathbf{w}_i$.
- electrical flow problem $\rightarrow$ quadratic minimization problem (defined in Section 5.1.1)
- total energy of electrical flow / effective resistance $\rightarrow \mathcal{OBJ}2(\mathbf{w})$.

We first show that if $\lambda^{(t)}$ as defined on Line 6 of Algorithm 1 is an upper bound for $\mathcal{OBJ}(\mathbf{x}^{(t)})$. This is crucial in its use the normalizing factor in our update step on Line 7.

Lemma 5.4.2 *In all iterations we have:*

$$\mathcal{OBJ}(\boldsymbol{x}^{(t)}) \leq \lambda^{(t)}$$

Proof By the Cauchy-Schwarz inequality we have:

$$\begin{aligned}(\lambda^{(t)})^2 =& \left(\sum_i \mathbf{w}_i^{(t-1)}\right)\left(\sum_i \frac{1}{\mathbf{w}_i^{(t-1)}} \left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}^2\right) \\ \geq& \left(\sum_i \left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}\right)^2 \\ =& \mathcal{OBJ}(\mathbf{x}^{(t)})^2 \end{aligned} \tag{5.10}$$

Taking square roots of both sides completes the proof. ■

At a high level, the algorithm assigns weights $\mathbf{w}_i$ for each group, and iteratively reweighs them for N iterations. Recall that our key potential functions are $\mu^{(t)}$ which is the sum of weights of all groups, and:

$$\nu^{(t)} = \frac{1}{\text{OPT}} \sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i} \log(\mathbf{w}_i^{(t)}) \tag{5.11}$$

Where $\bar{\mathbf{x}}$ is a solution such that $\mathcal{OBJ}(\bar{\mathbf{x}}) = \text{OPT}$. We will show that if $\mathcal{OBJ}(\mathbf{x}^{(t)})$, or in turn $\lambda^{(t)}$ is large, then $\nu^{(t)}$ increases at a rate substantially faster than $\log(\mu^{(t)})$. These bounds, and the relations between $\mu^{(t)}$ and $\nu^{(t)}$ are summarized below:

Lemma 5.4.3 *1.*

$$\nu^{(t)} \leq \log(\mu^{(t)}) \tag{5.12}$$

2.

$$\mu^{(t)} \leq \left(1 + \frac{\epsilon(1+2\epsilon)}{\rho} t\right) \mu^{(t-1)} \tag{5.13}$$

and

$$\log(\mu^{(t)}) \leq \frac{\epsilon(1+2\epsilon)}{\rho} t + \log k \tag{5.14}$$

3. If in iteration t, $\lambda^{(t)} \geq (1+10\epsilon)OPT$ *and* $\|\boldsymbol{x} - \boldsymbol{s}_i\|_{\boldsymbol{L}_i} \leq \rho \frac{\boldsymbol{w}_i^{(t-1)}}{\mu^{(t-1)}} \lambda^{(t)}$ *for all groups i, then:*

$$\nu^t \geq \nu^{(t-1)} + \frac{\epsilon(1+9\epsilon)}{\rho} \tag{5.15}$$

The relationship between the upper and lower potentials can be established using the fact that $\mathbf{w}_i$ is non-negative:

Proof of Lemma 5.4.3, Part 1:

$$\begin{aligned}
\nu^{(t)} =& \frac{1}{\text{OPT}} \sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i} \log(\mathbf{w}_i^{(t)}) \\
\leq& \frac{1}{\text{OPT}} \sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i} \log\left(\sum_j \mathbf{w}_j^{(t)}\right) \\
=& \log(\mu^{(t)}) \left(\frac{1}{\text{OPT}} \sum_i \|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\right) \\
\leq& \log(\mu^{(t)})
\end{aligned} \tag{5.16}$$

■

Part 2 follows directly from the local behavior of the $\log$ function:

Proof of Lemma 5.4.3, Part 2: The update rules gives:

$$
\begin{aligned}
\mu^{(t)} =& \sum_i \mathbf{w}_i^{(t)} \\
=& \sum_i \mathbf{w}_i^{(t-1)} + \left(\frac{\epsilon}{\rho}\frac{\left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}}{\lambda^{(t)}} + \frac{2\epsilon^2}{k\rho}\right)\mu^{(t-1)} && \text{by update rule on Line 7} \\
=& \mu^{(t-1)} + \frac{\epsilon}{\rho}\frac{\sum_i \left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}}{\lambda^{(t)}}\mu^{(t-1)} + \sum_i \frac{2\epsilon^2}{k\rho}\mu^{(t-1)} \\
=& \mu^{(t-1)} + \frac{\epsilon}{\rho}\frac{\mathcal{OBJ}(\mathbf{x}^{(t)})}{\lambda^{(t)}}\mu^{(t-1)} + \frac{2\epsilon^2}{\rho}\mu^{(t-1)} \\
\leq& \mu^{(t-1)} + \frac{\epsilon}{\rho}\mu^{(t-1)} + \frac{2\epsilon^2}{\rho}\mu^{(t-1)} && \text{By Lemma 5.4.2} \\
=& \left(1 + \frac{\epsilon(1+2\epsilon)}{\rho}\right)\mu^{(t-1)} && (5.17)
\end{aligned}
$$

Using the fact that $1 + x \leq \exp(x)$ when $x \geq 0$ we get:

$$
\begin{aligned}
\mu^{(t)} \leq& \exp\left(\frac{\epsilon(1+2\epsilon)}{\rho}\right)\mu^{(t-1)} \\
\leq& \exp\left(t\frac{\epsilon(1+2\epsilon)}{\rho}\right)\mu^{(0)} \\
=& \exp\left(t\frac{\epsilon(1+2\epsilon)}{\rho}\right)k
\end{aligned}
$$

Taking logs of both sides gives Equation 5.14. ■

This upper bound on the value of μ^t also allows us to show that the balancing rule keeps the w_i^ts reasonably balanced within a factor of k of each other. The following corollary can also be obtained.

Corollary 5.4.4 *The weights at iteration* t *satisfy* $\mathbf{w}_i^{(t)} \geq \frac{\epsilon}{k}\mu^{(t)}$.

Proof

The proof is by induction on t. When $t = 0$ we have $\mathbf{w}_i^{(0)} = 1$, $\mu^{(0)} = k$ and the claim follows from $\frac{\epsilon}{k}k = \epsilon < 1$. When $t > 1$, we have:

$$\begin{aligned}
\mathbf{w}_i^{(t)} &\geq \mathbf{w}_i^{(t-1)} + \frac{2\epsilon^2}{k\rho}\mu^{(t-1)} && \text{By line 7} \\
&\geq \left(\frac{\epsilon}{k} + \frac{2\epsilon^2}{k\rho}\right)\mu^{(t-1)} && \text{By the inductive hypothesis} \\
&= \frac{\epsilon}{k}\left(1 + \frac{2\epsilon}{\rho}\right)\mu^{(t-1)} \\
&\geq \frac{\epsilon}{k}\left(1 + \frac{\epsilon(1+2\epsilon)}{\rho}\right)\mu^{(t-1)} \\
&\geq \frac{\epsilon}{k}\mu^{(t)} && \text{By Lemma 5.4.3, Part 2}
\end{aligned} \tag{5.18}$$

■

The proof of Part 3 is the key part of our analysis. The first order change of $\nu^{(t)}$ is written as a sum of products of $\mathbf{L}_i$ norms, which we analyze via the fact that $\mathbf{x}^{(t)}$ is the solution of a linear system from the quadratic minimization problem.

Proof of Lemma 5.4.3, Part 3: We make use of the following known fact about the behavior of the log function around 1:

Fact 5.4.5 *If* $0 \leq x \leq \epsilon$, *then* $\log(1+x) \geq (1-\epsilon)x$.

$$\begin{aligned}
\nu^{(t)} - \nu^{(t-1)} &= \frac{1}{\text{OPT}}\sum_{1\leq i\leq k}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\log\left(\mathbf{w}_i^{(t)}\right) - \frac{1}{\text{OPT}}\sum_{1\leq i\leq k}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\log\left(\mathbf{w}_i^{(t-1)}\right) \\
&\qquad \text{By Equation 5.11} \\
&= \frac{1}{\text{OPT}}\sum_{1\leq i\leq k}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\log\left(\frac{\mathbf{w}_i^{(t)}}{\mathbf{w}_i^{(t-1)}}\right) \\
&\geq \frac{1}{\text{OPT}}\sum_{1\leq i\leq k}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\log\left(1 + \frac{\epsilon}{\rho}\frac{\left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}}{\lambda^{(t)}}\frac{\mu^{(t-1)}}{\mathbf{w}_i^{(t-1)}}\right) && \text{By the update rule on line 7} \\
&\geq \frac{1}{\text{OPT}}\sum_{1\leq i\leq k}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\frac{\epsilon(1-\epsilon)}{\rho}\frac{\left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}}{\lambda^{(t)}}\frac{\mu^{(t-1)}}{\mathbf{w}_i^{(t-1)}} && \text{By Fact 5.4.5} \\
&= \frac{\epsilon(1-\epsilon)\mu_i^{(t-1)}}{\rho\text{OPT}\lambda^{(t)}}\sum_{1\leq i\leq k}\frac{1}{\mathbf{w}_i^{(t-1)}}\|\bar{\mathbf{x}} - \mathbf{s}_i\|_{\mathbf{L}_i}\left\|\mathbf{x}^{(t)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}
\end{aligned} \tag{5.19}$$

Since $\mathbf{L}_i$ forms a P.S.D norm, by the Cauchy-Schwarz inequality we have:

$$\begin{aligned}\left\|\bar{\mathbf{x}}-\mathbf{s}_i\right\|_{\mathbf{L}_i}\left\|\mathbf{x}^{(t)}-\mathbf{s}_i\right\|_{\mathbf{L}_i} &\geq (\bar{\mathbf{x}}-\mathbf{s}_i)^T\mathbf{L}_i(\mathbf{x}^{(t)}-\mathbf{s}_i)\\ &= \left\|\mathbf{x}^{(t)}-\mathbf{s}_i\right\|_{\mathbf{L}_i}^2 + (\bar{\mathbf{x}}-\mathbf{x})^T\mathbf{L}_i(\mathbf{x}^{(t)}-\mathbf{s}_i) \qquad (5.20)\end{aligned}$$

Recall from Lemma 5.1.2 that since $\mathbf{x}^{(t)}$ is the minimizer to $\mathcal{OBJ}2(\mathbf{w}^{(t-1)})$, we have:

$$\left(\sum_i \frac{1}{\mathbf{w}_i^{(t-1)}}\mathbf{L}_i\right)\mathbf{x}^{(t)} = \sum_i \frac{1}{\mathbf{w}^{(t-1)}}\mathbf{s}_i \qquad (5.21)$$

$$\left(\sum_i \frac{1}{\mathbf{w}_i^{(t-1)}}\mathbf{L}_i\right)(\mathbf{x}^{(t)}-\mathbf{s}_i) = \mathbf{0} \qquad (5.22)$$

$$(\bar{\mathbf{x}}-\mathbf{x}^{(t)})^T\left(\sum_i \frac{1}{\mathbf{w}_i^{(t-1)}}\mathbf{L}_i\right)(\mathbf{x}^{(t)}-\mathbf{s}_i) = 0 \qquad (5.23)$$

Substituting this into Equation 5.19 gives:

$$\begin{aligned}\nu^{(t)}-\nu^{(t-1)} &\geq \frac{\epsilon(1-\epsilon)\mu_i^{(t-1)}}{\rho\text{OPT}\lambda^{(t)}}\sum_i \frac{1}{\mathbf{w}_i^{(t-1)}}\left\|\mathbf{x}^{(t)}-\mathbf{s}_i\right\|_{\mathbf{L}_i}^2\\ &= \frac{\epsilon(1-\epsilon)}{\rho\text{OPT}\lambda^{(t)}}\mu^{(t-1)}\mathcal{OBJ}2(\mathbf{w},\mathbf{x}^{(t)})\\ &= \frac{\epsilon(1-\epsilon)}{\rho\text{OPT}\lambda^{(t)}}(\lambda^{(t)})^2 \qquad \text{By definition of } \lambda^{(t)} \text{ on Line 6}\\ &\geq \frac{\epsilon(1-\epsilon)(1+10\epsilon)}{\rho} \qquad \text{By assumption that } \lambda^{(t)} > (1+10\epsilon)\text{OPT}\\ &\geq \frac{\epsilon(1+8\epsilon)}{\rho} \qquad (5.24)\end{aligned}$$

Since the iteration count largely depends on ρ, it suffices to provide bounds for ρ over all the iterations. The proof makes use of the following lemma about the properties of electrical flows, which describes the behavior of modifying the weights of a group S_i that has a large contribution to the total energy. It can be viewed as a multiple-edge version of Lemma 2.6 of [CKM+11].

Lemma 5.4.6 *Assume that $\epsilon^2\rho^2 < 1/10k$ and $\epsilon < 0.01$ and let $\boldsymbol{x}^{(t-1)}$ be the minimizer for $\mathcal{OBJ}2(\boldsymbol{w}^{(t-1)})$. Suppose there is a group i such that $\left\|\boldsymbol{x}^{(t-1)}-\boldsymbol{s}_i\right\|_{\boldsymbol{L}_i} \geq \rho\frac{\boldsymbol{w}_i^{(t-1)}}{\mu^{(t-1)}}\lambda^{(t)}$, then*

$$\mathit{OPT2}(\boldsymbol{w}^{(t)}) \leq \exp\left(-\frac{\epsilon^2\rho^2}{2k}\right)\mathit{OPT2}(\boldsymbol{w}^{(t-1)})$$

Proof

We first show that group i contributes a significant portion to $\mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)})$. Squaring both sides of the given condition gives:

$$
\begin{aligned}
\left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}^2 &\geq \rho^2 \frac{(\mathbf{w}_i^{(t-1)})^2}{(\mu^{(t-1)})^2} (\lambda^{(t)})^2 \\
&= \rho^2 \frac{(\mathbf{w}_i^{(t-1)})^2}{(\mu^{(t-1)})^2} \mu^{(t-1)} \mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)}) && (5.25) \\
\frac{1}{\mathbf{w}_i^{(t-1)}} \left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i} &\geq \rho^2 \frac{\mathbf{w}_i^{(t-1)}}{\mu^{(t-1)}} \mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)}) \\
&\geq \frac{\epsilon \rho^2}{k} \mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)}) \qquad \text{By Corollary 5.4.4} && (5.26)
\end{aligned}
$$

Also, by the update rule we have $\mathbf{w}_i^{(t)} \geq (1+\epsilon)\mathbf{w}_i^{(t-1)}$ and $\mathbf{w}_j^{(t)} \geq \mathbf{w}_j^{(t-1)}$ for all $1 \leq j \leq k$. So we have:

$$
\begin{aligned}
\text{OPT2}(\mathbf{w}^{(t)}) &\leq \mathcal{OBJ}2(\mathbf{w}^{(t)}, \mathbf{x}^{(t-1)}) \\
&= \mathcal{OBJ}2(\mathbf{w}^{(t)}, \mathbf{x}^{(t-1)}) - (1 - \frac{1}{1+\epsilon}) \left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}^2 \\
&\leq \mathcal{OBJ}2(\mathbf{w}^{(t)}, \mathbf{x}^{(t-1)}) - \frac{\epsilon}{2} \left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i}^2 \\
&\leq \mathcal{OBJ}2(\mathbf{w}^{(t)}, \mathbf{x}^{(t-1)}) - \frac{\epsilon^2 \rho^2}{2k} \mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)}) \\
&\leq \exp\left(-\frac{\epsilon^2 \rho^2}{2k}\right) \mathcal{OBJ}2(\mathbf{w}^{(t-1)}, \mathbf{x}^{(t-1)}) && (5.27)
\end{aligned}
$$

■

This means the value of the quadratic minimization problem can be used as a second potential function. We first show that it's monotonic and establish rough bounds for it.

Lemma 5.4.7 *OPT2($\mathbf{w}^{(0)}$) $\leq n^{3d}$ and OPT2($\mathbf{w}^{(t)}$) is monotonically decreasing in t.*

Proof By the assumption that the input is polynomially bounded we have that all entries of $\mathbf{s}$ are at most n^d and $\mathbf{L}_i \preceq n^d \mathbf{I}$. Setting $\mathbf{x}_u = 0$ gives $\|\mathbf{x} - \mathbf{s}_i\|_2 \leq n^{d+1}$. Combining this with the spectrum bound then gives $\|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i} \leq n^{2d+1}$. Summing over all the groups gives the upper bound.

The monotonicity of OPT2($\mathbf{w}^{(t)}$) follows from the fact that all weights are decreasing. ■

Combining this with the fact that OPT2($\mathbf{w}^{(N)}$) is not low enough for termination gives our bound on the total iteration count.

Proof of Theorem 5.4.1: The proof is by contradiction. Suppose otherwise, since $\mathcal{OBJ}(\mathbf{x}^{(N)}) > \epsilon$ we have:

$$\begin{aligned}\lambda^{(N)} \geq& (1+10\epsilon)n^{-d}\\ &\geq 2n^{-d}\text{OPT} \qquad (5.28)\end{aligned}$$

$$\sqrt{\mu^{(N)}\text{OPT2}(\mathbf{w}^{(t)})} \geq 2n^{-d} \qquad (5.29)$$

$$\text{OPT2}(\mathbf{w}^{(t)}) \geq \frac{4}{n^{-2d}\mu^{(N)}} \qquad (5.30)$$

Which combined with $\text{OPT2}(\mathbf{w}^{(0)}) \leq n^{3d}$ from Lemma 5.4.7 gives:

$$\frac{\text{OPT2}(\mathbf{w}^{(0)})}{\text{OPT2}(\mathbf{w}^{(N)})} \leq n^{5d}\mu^{(N)} \qquad (5.31)$$

By Lemma 5.4.3 Part 2, we have:

$$\begin{aligned}\log(\mu^{(N)}) &\leq \frac{\epsilon(1+\epsilon)}{\rho}N + \log k\\ &\leq \frac{\epsilon(1+\epsilon)}{\rho}10d\rho\log n\epsilon^{-2} + \log n \qquad \text{By choice of } N = 10d\rho\log n\epsilon^{-2}\\ &= 10(1+\epsilon)\epsilon^{-1}\log n + \log n\\ &\leq 10d(1+2\epsilon)\epsilon^{-1}\log n \qquad \text{when } \epsilon < 0.01 \qquad (5.32)\end{aligned}$$

Combining with Lemma 5.4.6 implies that the number of iterations where $\left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i} \geq \rho\frac{\mathbf{w}_i^{(t-1)}}{\mu^{(t-1)}}\lambda^{(t)}$ for i is at most:

$$\begin{aligned}\log\left(\mu^{(N)}n^{5d}\right)/\left(\frac{\epsilon^2\rho^2}{2k}\right) &= 10d(1+3\epsilon)\epsilon^{-1}\log n/\left(\frac{2\epsilon^{2/3}}{k^{1/3}}\right) \qquad \text{By choice of } \rho = 2k^{1/3}\epsilon^{-2/3}\\ &= 8d\epsilon^{-5/3}k^{1/3}\log n\\ &= 4d\epsilon^{-1}\rho\log n \leq \epsilon N \qquad (5.33)\end{aligned}$$

This means that we have $\left\|\mathbf{x}^{(t-1)} - \mathbf{s}_i\right\|_{\mathbf{L}_i} \leq \rho\frac{\mathbf{w}_i^{(t-1)}}{\mu^{(t-1)}}\lambda^{(t)}$ for all $1 \leq i \leq k$ for at least $(1-\epsilon)N$ iterations and therefore by Lemma 5.4.3 Part 3:

$$\nu^{(N)} \geq \nu^{(0)} + \frac{\epsilon(1+8\epsilon)}{\rho}(1-\epsilon)N > \mu^{(N)} \qquad (5.34)$$

Giving a contradiction. ■

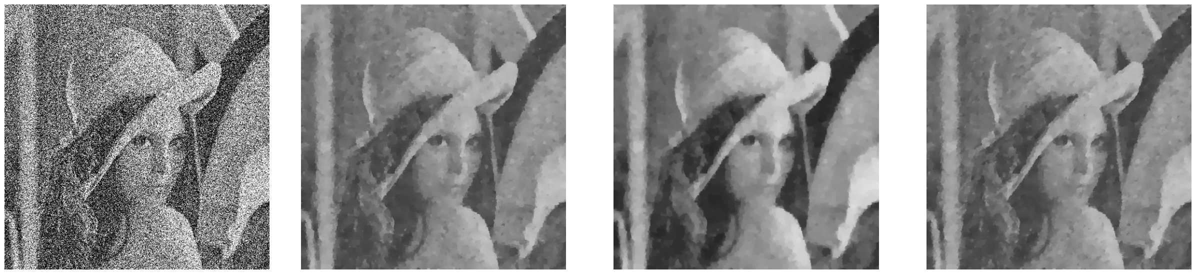
5.5 Evidence of Practical Feasibility

We performed a series of experiments using the algorithm described in Section 5.4 in order to demonstrate its practical feasibility. Empirically, our running times are slower than the state of the art methods, but are nonetheless reasonable. This suggests the need of further experimental works on a more optimized version, which is outside of the scope of this paper.

The SDD linear systems that arise in the quadratic minimization problems were solved using the combinatorial multigrid (CMG) solver [KM09b, KMT09]. One side observation confirmed by these experiments is that for the sparse SDD linear systems that arise from image processing, the CMG solver yields good results both in accuracy and running time.

5.5.1 Total Variational Denoising

Total Variational Denoising is the concept of applying Total Variational Minimization as denoising process. This was pioneered by Rudin, Osher and Fatemi [ROF92] and is commonly known as the ROF image model. Our formulation of the grouped least squares problems yields a simple way to solve the ROF model and most of its variants. In Figure 5.1, we present a simple denoising experiment using the standard image processing data set, ‘Lenna’. The main goal of the experiment is to show that our algorithm is competitive in terms of accuracy, while having running times comparable to first-order methods. On a 512×512 grayscale image, we introduce Additive White Gaussian Noise (AWGN) at a measured Signal to Noise Ratio (SNR) of 2. AWGN is the most common noise model in photon capturing sensors from consumer cameras to space satellites systems. We compare the results produced by our algorithm with those by the Split Bregman algorithm from [GO09] and the Gauss-Seidel variation of the fixed point algorithm from [MSX11]. These methods minimize an objective with ℓ_2^2 fidelity term given in Equation 5.3 while we used the variant with ℓ_2 fidelity shown in Equation 5.6. Also, the parameters in these algorithms were picked to give the best results for the objective functions being minimized. As a result, for measuring the qualities of output images we only use the ℓ_2 and ℓ_1 norms of pixel-wise differences with the original image.



Noisy Version	Split Bregman	Grouped Least Squares	Fixed Point
29.29E3, 8.85E6	6.70E3, 1.89E6	5.99E3, 1.39E6	6.64E3, 1.87E6

Figure 5.1: Outputs of various denoising algorithms on image with AWGN noise. From left to right: noisy version, Split Bregman [GO09], Fixed Point [MSX11], and Grouped Least Squares. Errors listed below each figure from left to right are ℓ_2 and ℓ_1 norms of differences with the original.

Our experiments were conducted on a single core 64-bit Intel(R) Xeon(R) E5440 CPU @ 2.83GHz. The non-solver portion of the algorithm was implemented in Matlab(R). On images of size 256×256, 512×512 and 1024×1024, the average running times are 2.31, 9.70 and 47.61 seconds respectively. These running times are noticeably slower than the state of the art. However, on average only 45% of the total running time is from solving the SDD linear systems using the CMG solver. The rest is mostly from reweighting edges and MATLAB function calls, which should be much faster in more optimized versions. More importantly, in all of our experiments the weights are observed to converge in under 15 iterations, even for larger images of size up to 3000×3000.

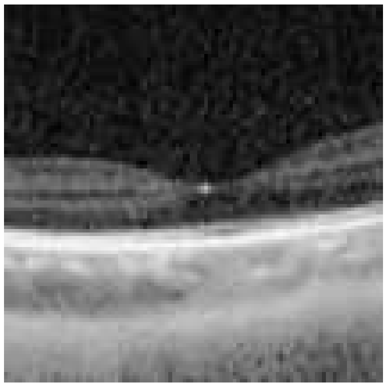
5.5.2 Image Processing

As exemplified by the denoising with colors application discussed in Section 5.2.2, the grouped least squares framework can be used for a wide range of image processing tasks. Some examples of such applications are shown in Figure 5.2. Our denoising algorithm can be applied as a preprocessing step to segmenting images of the retina obtained from Optical Coherence Tomography (OCT) scans. Here the key is to preserve the sharpness between the nerve fiber layers and this is achieve by using a ℓ_1 regularization term. Variations of this formulation allows one to emulate a large variety of established image preprocessing applications. For example, introducing additional ℓ_2 terms containing differences of neighboring pixels of a patch leads to the removal of boundaries, giving an overall blurring effect. On our examples, this leads to results similar to methods that apply a filter over the image, such as Gaussian blurring. Introducing such effects using an objective function has the advantage that it can be used in conjunction with other terms. By mixing and matching penalty terms on the groups, we can preserve global features while favoring the removal of small artifacts introduced by sensor noise.

An example of Poisson Image Editing mentioned in Section 5.2.3 are shown in Figure 5.3. The application is seamless cloning as described in Section 3 of [PGB03], which aims to insert complex objects into another image. Given two images, they are blended by solving the discrete poisson equation based on a mix of their gradients and boundary values. We also added ℓ_2 constraints on different parts of the image to give a smoother result. The input consists of locations of the foreground pictures over the background, along with boundaries (shown in red) around the objects in the foreground. These rough boundaries makes the blending of surrounding textures the main challenge, and our three examples (void/sky, sea/pool, snow/sand) are some representative situations. These examples also show that our approaches can be extended to handle multichannel images (RGB or multi-spectral) with only a few modifications.

5.6 Relations between Graph Problems and Minimizing LASSO Objectives

We now give formal proofs that show the shortest path problem is an instance of LASSO and the minimum cut problem is an instance of fused-LASSO. Our proofs do not guarantee that the answers returned are a single path or cut. In fact, when multiple solutions have the same value it's possible for our algorithm to return a linear combination of them. However, we can ensure that the optimum solution is unique using the Isolation Lemma of Mulmuley, Vazarani and Vazarani


OCT scan of retina

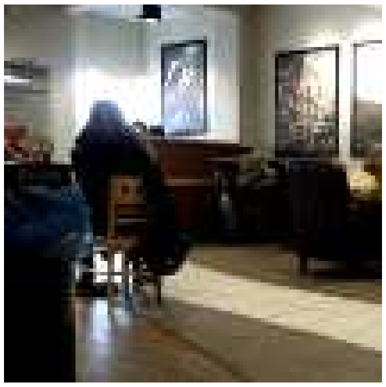
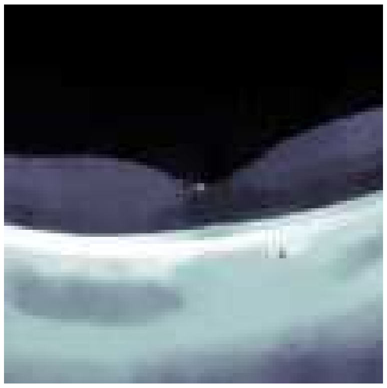
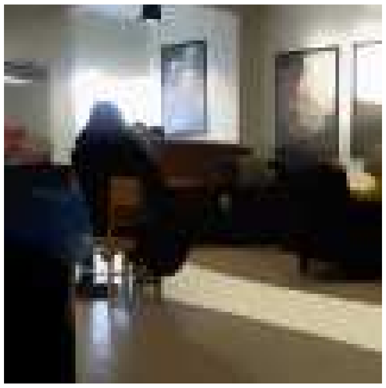


Photo of room

Newspaper


Global features


Scan after preprocessing

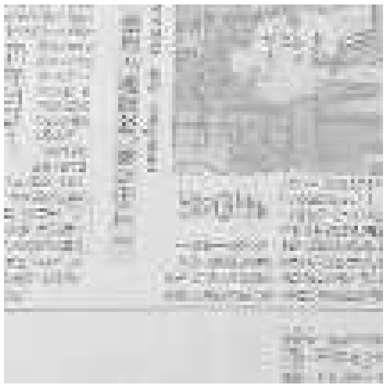

Decolored version

Figure 5.2: Applications to various image processing tasks. From left to right: image segmentation, global feature extraction / blurring, and decoloring.

[MVV87] while only incurring polynomial increase in edge lengths/weights. This analysis is similar to the one in Section 3.5 of [DS08] for finding a unique minimum cost flow, and is omitted here.

We prove the two claims in Fact 5.0.1 about shortest path and minimum cut separately in Lemmas 5.6.1 and 5.6.2.

Lemma 5.6.1 *Given a s-t shortest path instance in an undirected graph where edge lengths $\boldsymbol{l}$: $E \rightarrow \mathbb{R}^+$ are integers between 1 and n^d. There is a LASSO minimization instance where all entries are bounded by $n^{O(d)}$ such that the value of the LASSO minimizer is within 1 of the optimum answer.*

Proof Our reductions rely crucially on the edge-vertex incidence matrix, which we denote using ∂. Entries of this matrix are defined as follows:

$$\partial_{e,u} = \begin{cases} -1 & \text{if u is the head of e} \\ 1 & \text{if u is the tail of e} \\ 0 & \text{otherwise} \end{cases} \tag{5.35}$$

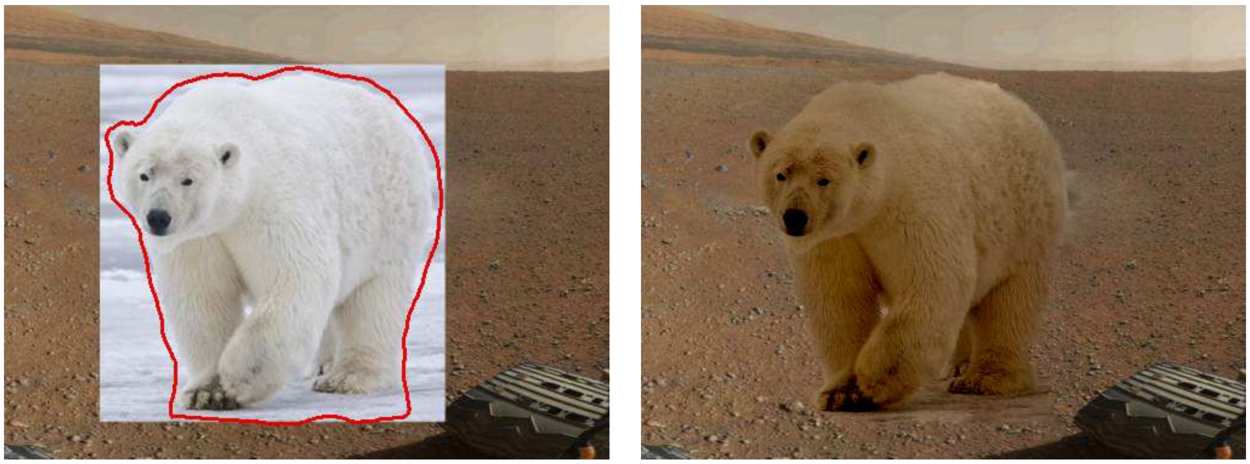

Marked polar bear Polar Bear on Mars [2]

Figure 5.3: Example of seamless cloning using Poisson Image Editing

We first show the reduction for shortest path. Then a path from s to t corresponds to a flow value assigned to all edges, $\mathbf{f} : E \rightarrow \mathbb{R}$ such that $\partial^T \mathbf{f} = \chi_{s,t}$. If we have another flow $\mathbf{f}'$ corresponding to any path from s to t, then this constraint can be written as:

$$\left\| \partial^T \mathbf{f} - \chi_{s,t} \right\|_2 = \mathbf{0} \tag{5.36}$$

$$\left\| \partial^T (\mathbf{f} - \mathbf{f}') \right\|_2 = \mathbf{0}$$

$$\| \mathbf{f} - \mathbf{f}' \|_{\partial \partial^T} = \mathbf{0} \tag{5.37}$$

The first constraint is closer to the classical LASSO problem while the last one is within our definition of grouped least squares problem. The length of the path can then be written as $\sum_e l_e |f_e|$. Weighting these two terms together gives:

$$\min_{\mathbf{f}} \lambda \| \mathbf{f} - \mathbf{f}' \|^2_{\partial \partial^T} + \sum_e l_e |f_e| \tag{5.38}$$

Where the maximum entry is bounded by $\max\{n^d, n^2 \lambda\}$. Clearly its objective is less than the length of the shortest path, let this solution be $\bar{\mathbf{f}}$. Then since the total objective is at most n^{d+1}, we have that the maximum deviation between $\partial^T \bar{\mathbf{f}}$ and $\chi_{s,t}$ is at most n^{d+1}/λ. Then given a spanning tree, each of these deviations can be routed to s or t at a cost of at most n^{d+1} per unit of flow. Therefore we can obtain $\mathbf{f}'$ such that $\partial^T \mathbf{f}' = \chi_{s,t}$ whose objective is bigger by at most n^{2d+2}/λ. Therefore setting $\lambda = n^{2d+2}$ guarantees that our objective is within 1 of the length of the shortest path, while the maximum entry in the problem is bounded by $n^{O(d)}$. ■

We now turn our attention to the minimum cut problem, which can be formulated as finding a vertex labeling $\mathbf{x}^{(vert)}$ where $\mathbf{x}_s^{(vert)} = 0$, $\mathbf{x}_t^{(vert)} = 1$ and the size of the cut, $\sum_{uv \in E} |\mathbf{x}_u^{(vert)} - \mathbf{x}_v^{(vert)}|$. Since the ℓ_1 term in the objective can incorporate single variables, we use an additional vector

$\mathbf{x}^{(edge)}$ to indicate differences along edges. The minimum cut problem then becomes minimizing $|\mathbf{x}^{(edge)}|$ subject to the constraint that $\mathbf{x}^{(edge)} = \partial' \mathbf{x}^{(vert)'} + \partial \chi_t$, where ∂' and $\mathbf{x}^{(vert)'}$ are restricted to vertices other than s and t and χ_t is the indicator vector that's 1 on t. The equality constraints can be handled similar to the shortest path problem by increasing the weights on the first term. One other issue is that $|\mathbf{x}^{(vert)'}|$ also appears in the objective term, and we handle this by scaling down $\mathbf{x}^{(vert)'}$, or equivalently scaling up ∂'.

Lemma 5.6.2 *Given a s-t minimum cut instance in an undirected graph where edge weights $\mathbf{w} : E \to \mathbb{R}^+$ are integers between 1 and n^d. There is a LASSO minimization instance where all entries are bounded by $n^{O(d)}$ such that the value of the LASSO minimizer is within 1 of the minimum cut.*

Proof

Consider the following objective, where λ_1 and λ_2 are set to n^{d+3} and n^2:

$$\lambda_1 ||\lambda_2 \partial' \mathbf{x}^{(vert)'} + \partial \chi_t - \mathbf{x}^{(edge)}||_2^2 + |\mathbf{x}^{(vert')}|_1 + |\mathbf{x}^{(edge)}|_1 \tag{5.39}$$

Let $\bar{\mathbf{x}}$ be an optimum vertex labelling, then setting $\mathbf{x}^{(vert')}$ to the restriction of $n^{-2}\bar{\mathbf{x}}$ on vertices other than s and t and $\mathbf{x}^{(edge)}$ to $\partial \mathbf{x}^{(vert)}$ makes the first term 0. Since each entry of $\bar{\mathbf{x}}$ is between 0 and 1, the additive increase caused by $|\mathbf{x}^{(vert')}|_1$ is at most $1/n$. Therefore this objective's optimum is at most $1/n$ more than the size of the minimum cut.

For the other direction, consider any solution $\mathbf{x}^{(vert)'}, \mathbf{x}^{(edge)'}$ whose objective is at most $1/n$ more than the size of the minimum cut. Since the edge weights are at most n^d and s has degree at most n, the total objective is at most $n^{d+1} + 1$. This gives:

$$\begin{aligned} ||\lambda_2 \partial' \mathbf{x}^{(vert)'} + \partial \chi_t - \mathbf{x}^{(edge)}||_2^2 &\leq n^{-1} \\ ||\lambda_2 \partial' \mathbf{x}^{(vert)'} + \partial \chi_t - \mathbf{x}^{(edge)}||_1 & \\ \leq ||\lambda_2 \partial' \mathbf{x}^{(vert)'} + \partial \chi_t - \mathbf{x}^{(edge)}||_2 &\leq n^{-1/2} \end{aligned} \tag{5.40}$$

Therefore changing $\mathbf{x}^{(edge)}$ to $\lambda_2 \partial' \mathbf{x}^{(vert)'} + \partial \chi_t$ increases the objective by at most $n^{-1/2} < 1$. This gives a cut with weight within 1 of the objective value and completes the proof. ■

We can also show a more direct connection between the minimum cut problem and the fused LASSO objective, where each absolute value term may contain a linear combination of variables. This formulation is closer to the total variation objective, and is also an instance of the problem formulated in Definition 5.1.1 with each edge in a group.

Lemma 5.6.3 *The minimum cut problem in undirected graphs can be written as an instance of the fused LASSO objective.*

Proof Given a graph $G = (V, E)$ and edge weights $\mathbf{p}$, the problem can be formulated as finding a vertex labeling x such that $x_s = 0$, $x_t = 1$ and minimizing:

$$\sum_{uv \in E} p_{uv} |x_u - x_v| \tag{5.41}$$

■

5.7 Other Variants

Although our formulation of $\mathcal{OBJ}$ as a sum of ℓ_2 objectives differs syntactically from some common formulations, we show below that the more common formulation involving quadratic, or ℓ_2^2, fidelity term can be reduced to finding exact solutions to $\mathcal{OBJ}$ using 2 iterations of ternary search. Most other formulations differs from our formulation in the fidelity term, but more commonly have ℓ_1 smoothness terms as well. Since the anisotropic smoothness term is a special case of the isotropic one, our discussion of the variations will assume anisotropic objectives.

5.7.1 ℓ_2^2 fidelity term

In practice, total variation objectives more commonly involve ℓ_2^2 fidelity terms. This term can be written as $\|\mathbf{x} - \mathbf{s}_0\|_2^2$, which corresponds to the norm defined by $\mathbf{I} = \mathbf{L}_0$. This gives:

$$\min_{\mathbf{x}} \quad \|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0}^2 + \sum_{1 \le i \le k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$$

We can establish the value of $\|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0}^2$ separately by guessing it as a constraint. Since the t^2 is convex in t, the following optimization problem is convex in t as well:

$$\min_{\mathbf{x}} \quad \sum_{1 \le i \le k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i} \; \|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0}^2 \le t^2$$

Also, due to the convexity of t^2, ternary searching on the minimizer of this plus t^2 would allow us to find the optimum solution by solving $O(\log n)$ instances of the above problem. Taking square root of both sides of the $\|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0}^2 \le t^2$ condition and taking its Lagrangian relaxation gives:

$$\min_{\mathbf{x}} \max_{\lambda \ge 0} \sum_{i=1}^{k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i} + \lambda(\|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0} - t)$$

Which by the min-max theorem is equivalent to:

$$\max_{\lambda \ge 0} -\lambda t + \left(\min_{\mathbf{x}} \sum_{i=1}^{k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i} + \lambda \|\mathbf{x} - \mathbf{s}_0\|_{\mathbf{L}_0} \right)$$

The term being minimized is identical to our formulation and its objective is convex in λ when $\lambda \ge 0$. Since $-\lambda t$ is linear, their sum is convex and another ternary search on λ suffices to optimize the overall objective.

5.7.2 ℓ_1 fidelity term

Another common objective function to minimize is where the fidelity term is also under ℓ_1 norm. In this case the objective function becomes:

$$\|\mathbf{x} - \mathbf{s}_0\|_1 + \sum_i \sum_{1 \le i \le k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$$

This can be rewritten as a special case of $\mathcal{OBJ}$ as:

$$\sum_u \sqrt{(x_u - s_u)^2} + \sum_i \sum_{1 \le i \le k} \|\mathbf{x} - \mathbf{s}_i\|_{\mathbf{L}_i}$$

Which gives, instead, a grouped least squares problem with $n + k$ groups.

5.8 Remarks

We believe that the ability of our algorithm to encompass many of the current image processing algorithms represents a major advantage in practice. It allows the use of a common data structure (the underlying graph) and subroutine (linear system solvers) for many different tasks in the image processing pipeline. Theoretically, the grouped least squares problem is also interesting as it represents an intermediate problem between linear and quadratic optimization.

The performances of our algorithms given in Section 5.4 depend on k, which is the number of groups in the formulation given in Definition 5.1.1. Two settings of k are useful for understanding the relation of the grouped least squares problem with other problems. When $k = 1$, the problem becomes the electrical flow problem, and the running time of our algorithm is similar to directly solving the linear system. This is also the case when there is a small (constant) number of groups. The other extremum is when each edge belongs to its own group, aka. $k = m$. Here our approximate algorithm is the same as the minimum s-t cut algorithm given in [CKM$^+$11]. This means although the problem with smaller number of groups is no longer captured by linear optimization, the minimum s-t cut problem – that still falls within the framework of linear optimization – is in some sense the hardest problem in this class. Therefore the grouped least squares problem is a natural interpolation between the L_1 and L_2^2 optimization. Theoretical consequences of this interpolation for undirected multicommodity flow [KMP12] and flows on separable graphs [MP13] have been explored. However, these algorithms have been superceeded by subsequent works using different approaches [KLOS13, She13].

The preliminary experimental results from Section 5.5 show that more aggressive reweightings of edges lead to much faster convergence than what we showed for our two algorithms. Although the running time from these experiments are slower than state of the art methods, we believe the results suggest that more thorough experimental studies with better tuned algorithms are needed. Also, the Mumford-Shah functional can be better approximated by non-convex functions [MS89]. Objectives as hinged loss often lead to better results in practice [RDVC$^+$04], but few algorithmic guarantees are known for them. Designing algorithms with strong guarantees for minimizing these objectives is an interesting direction for future work.

Chapter 6

Conclusion and Open Problems

We showed highly efficient algorithms for solving symmetric diagonally dominant linear systems. The ever increasing number of theoretical applications using these solvers as subroutines, as well as extensive practical works on solving such systems makes the practicality of our approaches a pressing question. Existing combinatorial preconditioning packages such as the combinatorial multigrid solver [KMST09] have much in common with the algorithm from Chapter 2. This package performs well on many real world graphs such as the ones derived from images in Chapter 5. Its performance are comparable to other methods such as LAPACK [ABB$^+$99], algebraic multigrid packages, and recent solvers specialized for images by Krishnan et al. [KFS13]. However, we believe several theoretical questions need to be addressed in order to bring the algorithms from Chapters 2 and 3 closer to practice.

6.1 Low Stretch Spanning Trees

The main difficulty of implementing the algorithm from Chapter 2 is the difficulty of generating a good low stretch subgraph/spanning tree. Although the algorithm for constructing these trees were greatly simplified by Abraham and Neiman [AN12], their algorithm, as well as previous ones, relied on a partition routine by Seymour [Sey95]. A close examination of these routines, as well as the overall algorithm shows several parameters that needs to be set carefully. The running time of the algorithm at $\Omega(n \log n \log \log n)$ is also the overall runtime bottleneck when the graph is sparse and ϵ is relatively large (e.g. constant). Their parallelization are also difficult due to performing shortest path computations across the entire graph. As a result, results with sub-optimal parameters (compared to sequential algorithms) such as the low stretch subgraph algorithm in Section 4.3 are needed for the parallel algorithm.

The observation that higher stretch edges can be treated differently in Lemma 2.5.1 and Section 4.4 represents a way to replace this routine with simpler low diameter decomposition routines such as the one shown in Section 4.2.

The highest level structure in recent low stretch spanning tree algorithms is known as a star decomposition. One of the main intricacies in this decomposition is the need to bound the overall diameter growth incurred by subsequent recursive calls. This is in turn due to the need to bound the stretch of edges cut, especially ones of much smaller weight. The use of $\ell_{1/2+\alpha}$ moment means the contribution of such edges to total stretch is much smaller. As a result, closer examinations of the scheme by Abraham and Neiman [AN12] in conjunction with such weights are likely to lead to faster, simpler, and more practical algorithms for generating low stretch spanning trees.

6.2 Interaction between Iterative Methods and Sparsification

The routine central to the parallel algorithm from Chapter 3 is repeatedly squaring a matrix and approximating it. The final error is in turn given by the product of these errors of approximation. As a result, for k levels of this scheme, an error of $O\left(\frac{1}{k}\right)$ is needed per step to ensure a constant overall error. This results in a size increase by a factor of $O\left(k^2\right)$ in the sizes of the approximations. In our algorithm, k was set to $O\left(\log\left(nU\right)\right)$ where U is the ratio between maximum and minimum edge weights,. If this dependency can be replaced by a constant, the work of this algorithm will reduce from $O\left(n \log n \log^3\left(nU\right)\right)$ to $O\left(n \log n \log\left(nU\right)\right)$, which is much closer to being feasible in practice.

From the viewpoint of squaring a scalar and approximating the result at each step, the accumulation of error is almost unavoidable. However, the final purpose of this accumulation is to approximate the inverse. The algorithm can be viewed as trying to do two things at once: simulate an iterative method given by an infinite power series; while maintaining the sparsity of the matrices via. spectral sparsification. The former reduces error but increases problem size, while the later does the opposite. The relatively direct manner in which we trade between these two is likely the source of the accumulation of errors.

This leads to the question of whether a better middle ground is possible. Specifically, whether it is possible to analyze iterative methods in conjunction with spectral sparsification. Our rudimentary iterative method behaves very differently on eigenvalues based on their magnitude. In numerical analysis, a useful technique is to analyze the eigenvalues instead of proving sweeping bounds on the entire possible range (see e.g. [TB97]). These approaches are also related to the construction of deterministic spectral sparsifiers with optimal parameters by Batson et al. [BSS09] and Kolla et al. [KMST10]. Combining these approaches may lead to iterative methods where errors from repeated sparsification accumulate in a more controllable manner.

Bibliography

[ABB$^+$99] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *Lapack users' guide (third ed.)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. 6

[ABCP92] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg, *Low-diameter graph decomposition is in NC*, Proceedings of the Third Scandinavian Workshop on Algorithm Theory (London, UK, UK), SWAT '92, Springer-Verlag, 1992, pp. 83–93. 4.2

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman, *Nearly tight low stretch spanning trees*, Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '08, IEEE Computer Society, 2008, pp. 781–790. 2.2, 2.2, 4.1, 4.4, 4.4, 4.4

[ACL06] Reid Andersen, Fan Chung, and Kevin Lang, *Local graph partitioning using pagerank vectors*, Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '06, IEEE Computer Society, 2006, pp. 475–486. 3.3

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale, *The multiplicative weights update method: a meta algorithm and applications*, Tech. report, Princeton University, 2005. 5.4

[AKPW95] N. Alon, R. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput. **24** (1995), no. 1, 78–100. 1.2, 2.2, 4.1, 4.3, 4.3.1

[ALGP89] B. Awerbuch, M. Luby, A. V. Goldberg, and S. A. Plotkin, *Network decomposition and locality in distributed computation*, Proceedings of the 30th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), SFCS '89, IEEE Computer Society, 1989, pp. 364–369. 4

[AM85] Noga Alon and V. D. Milman, *lambda$_1$, isoperimetric inequalities for graphs, and superconcentrators*, J. Comb. Theory, Ser. B **38** (1985), no. 1, 73–88. 1.4

[AN12] Ittai Abraham and Ofer Neiman, *Using petal decompositions to build a low stretch spanning tree*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 395–406. 2.2, 2.2, 4.1, 2, 4.4, 4.4, 4.4, 4.4, 6.1

[AP09] Reid Andersen and Yuval Peres, *Finding sparse cuts locally using evolving sets*, Proceedings of the 41st annual ACM symposium on Theory of computing (New York, NY, USA), STOC '09, ACM, 2009, pp. 235–244. 3.3

[AST09] Haim Avron, Gil Shklarski, and Sivan Toledo, *On element SDD approximability*, CoRR **abs/0911.0547** (2009). 5.1.1

[AW02] Rudolf Ahlswede and Andreas Winter, *Strong converse for identification via quantum channels*, IEEE Transactions on Information Theory **48** (2002), no. 3, 569–579. 1.6.4, B

[Awe85] Baruch Awerbuch, *Complexity of network synchronization*, J. ACM **32** (1985), no. 4, 804–823. 4

[Axe94] Owe Axelsson, *Iterative solution methods*, Cambridge University Press, New York, NY, 1994. 1.2, 1.5.1, 1.6.2, 2.1, 3.3

[BAP12] Scott Beamer, Krste Asanović, and David Patterson, *Direction-optimizing breadth-first search*, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (Los Alamitos, CA, USA), SC '12, IEEE Computer Society Press, 2012, pp. 12:1–12:10. 4.2.3

[Bar96] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, 1996, pp. 184–193. 2.2, 4.1, 4.2, 4.4

[Bar98] Yair Bartal, *On approximating arbitrary metrices by tree metrics*, Proceedings of the thirtieth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '98, ACM, 1998, pp. 161–168. 2.2, 4.1

[BCG11] Stephen R. Becker, Emmanuel J. Candes, and Michael Grant, *Templates for convex cone problems with applications to sparse signal recovery*, MATHEMATICAL PROGRAMMING COMPUTATION **3** (2011), 165. 5, 5.3.4

[BGK$^+$13] GuyE. Blelloch, Anupam Gupta, Ioannis Koutis, GaryL. Miller, Richard Peng, and Kanat Tangwongsan, *Nearly-linear work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs*, Theory of Computing Systems (2013), 1–34 (English). 1.2, 1.3.4, 1.4, 3, 3.4, 4, 4.1, 4.1, 1, 4.1, 4.2, 4.2.1, 4.3.1, 5.1.1

[BGT12] Guy E. Blelloch, Anupam Gupta, and Kanat Tangwongsan, *Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design*, Proceedinbgs of the

24th ACM symposium on Parallelism in algorithms and architectures (New York, NY, USA), SPAA '12, ACM, 2012, pp. 205–213. 4.2

[BH01] Eric Boman and Bruce Hendrickson, *On spanning tree preconditioners*, Manuscript, Sandia National Lab., 2001. 1.2, 1.5.1, 4, 4

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick, *A multigrid tutorial: second edition*, Society for Industrial and Applied Mathematics, 2000. 1.5.2

[BHV08] Erik G. Boman, Bruce Hendrickson, and Stephen A. Vavasis, *Solving elliptic finite element systems in near-linear time with support preconditioners*, SIAM J. Numerical Analysis **46** (2008), no. 6, 3264–3284. 1.4, 5.1.1

[BK96] András A. Benczúr and David R. Karger, *Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time*, Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '96, ACM, 1996, pp. 47–55. 1.6.4

[Ble96] Guy E. Blelloch, *Programming parallel algorithms*, Commun. ACM **39** (1996), no. 3, 85–97. 3

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava, *Twice-Ramanujan sparsifiers*, Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 2009, pp. 255–262. 6.2

[BSST13] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng, *Spectral sparsification of graphs: theory and algorithms*, Commun. ACM **56** (2013), no. 8, 87–94. 1.6.4

[BV04] S. Boyd and L. Vandenberghe, *Convex optimization*, Camebridge University Press, 2004. 5, 5.3.1

[Can06] J. Candés, E, *Compressive sampling*, Proceedings of the International Congress of Mathematicians (2006). 5, 5

[Che70] J. Cheeger, *A lower bound for the smallest eigenvalue of the laplacian*, Problems in Analysis (1970), 195–199. 1.4

[Chu97] F.R.K. Chung, *Spectral graph theory*, Regional Conference Series in Mathematics, vol. 92, American Mathematical Society, 1997. 1.4

[CKM$^+$11] Paul Christiano, Jonathan A. Kelner, Aleksander Mądry, Daniel Spielman, and Shang-Hua Teng, *Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs*, Proceedings of the 43^{rd} ACM Symposium on Theory of Computing (STOC), 2011. 1.3.5, 1.4, 5, 5.1.1, 5.3.2, 5.4, 5.4, 5.4, 5.4, 5.4, 5.8

[CMMP13] Hui Han Chin, Aleksander Mądry, Gary L. Miller, and Richard Peng, *Runtime guarantees for regression problems*, Proceedings of the 4th conference on Innovations in Theoretical Computer Science (New York, NY, USA), ITCS '13, ACM, 2013, pp. 269–282. 1.3.5

[Coh98] Edith Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, SIAM J. Comput. **28** (1998), no. 1, 210–236. 4.2

[Coh00] ______, *Polylog-time and near-linear work approximation scheme for undirected shortest paths*, J. ACM **47** (2000), no. 1, 132–166. 4.2, 4.2.1, 5

[CS05] Tony Chan and Jianhong Shen, *Image processing and analysis: Variational, PDE, wavelet, and stochastic methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005. 5.2.1

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson, *Introduction to algorithms*, 2nd ed., McGraw-Hill Higher Education, 2001. 1.5, 1.5.2, 1.6.2

[CW90] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetical progressions*, J. Symbolic Computation **9** (1990), 251–280. 1.2, 5.1.1

[DS06] Jérôme Darbon and Marc Sigelle, *Image restoration with discrete constrained total variation part i: Fast and exact optimization*, Journal of Mathematical Imaging and Vision **26** (2006), no. 3, 261–276. 5.3.2

[DS07] Samuel I. Daitch and Daniel A. Spielman, *Support-graph preconditioners for 2-dimensional trusses*, CoRR **abs/cs/0703119** (2007). 1.1, 5.1.1

[DS08] Samuel I. Daitch and Daniel A. Spielman, *Faster approximate lossy generalized flow via interior point algorithms*, Proceedings of the 40th annual ACM symposium on Theory of computing (New York, NY, USA), STOC '08, ACM, 2008, pp. 451–460. 1.4, 5, 5.6

[EEST08] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng, *Lower-stretch spanning trees*, SIAM Journal on Computing **38** (2008), no. 2, 608–628. 2.2, 2.2, 4.1, 4.4, 4.4

[Fel71] William Feller, *An introduction to probability theory and its applications. Vol. II.*, Second edition, John Wiley & Sons Inc., New York, 1971. 4.2.2

[Fie73] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Mathematical Journal **23** (1973), no. 98, 298–305. 1.4

[Fos53] F. G. Foster, *On the stochastic matrices associated with certain queueing processes*, Ann. Math. Statistics **24** (1953), 355–360. 1.6.4

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. Syst. Sci. **69** (2004), no. 3, 485–497. 2.2, 4.1, 4.2

[FS99] Yoav Freund and Robert E. Schapire, *Adaptive game playing using multiplicative weights*, Games and Economic Behavior **29** (1999), no. 1-2, 79–103. 5.4

[GMZ94] K.D. Gremban, G.L. Miller, and M. Zagha, *Performance evaluation of a parallel preconditioner*, CS CMU-CS-94-205, CMU, October 1994. 1.2

[GO09] Tom Goldstein and Stanley Osher, *The split bregman method for l1-regularized problems*, SIAM J. Img. Sci. **2** (2009), 323–343. (document), 5.5.1, 5.1

[Gol65] Sidney Golden, *Lower bounds for the helmholtz function*, Phys. Rev. **137** (1965), B1127–B1128. 1

[GR98] Andrew V. Goldberg and Satish Rao, *Beyond the flow decomposition barrier*, J. ACM **45** (1998), 783–797. 5.3.2

[Gre96] Keith Gremban, *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, October 1996, CMU CS Tech Report CMU-CS-96-123. 1.1

[GT13] A. Gupta and K. Talwar, *Random Rates for 0-Extension and Low-Diameter Decompositions*, CoRR **abs/1307.5582** (2013). 4.2

[GY04] Donald Goldfarb and Wotao Yin, *Second-order cone programming methods for total variation-based image restoration*, SIAM J. Sci. Comput **27** (2004), 622–645. 5.3.1

[Har11] Nicholas Harvey, *C&O 750: Randomized algorithms, winter 2011, lecture 11 notes*, http://www.math.uwaterloo.ca/~harvey/W11/Lecture11Notes.pdf, 2011. 1.6.4, B.1

[Hig02] Nicholas J. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. 2.6, 2.6, 2.6, C, C.2

[HS52] Magnus R. Hestenes and Eduard Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards **49** (1952), no. 6, 409–436. 1.2

[HVBJ11] Toby Hocking, Jean-Philippe Vert, Francis Bach, and Armand Joulin, *Clusterpath: an algorithm for clustering using convex fusion penalties*, Proceedings of the 28th International Conference on Machine Learning (ICML-11) (New York, NY, USA) (Lise Getoor and Tobias Scheffer, eds.), ICML '11, ACM, June 2011, pp. 745–752. 5.2.4

[JáJ92] Joseph JáJá, *An introduction to parallel algorithms*, Addison-Wesley, 1992. 1.2, 4.3.1

[JM93] David S. Johnson and Catherine C. McGeoch (eds.), *Network flows and matching: First dimacs implementation challenge*, American Mathematical Society, Boston, MA, USA, 1993. 5

[Jos97] Anil Joshi, *Topics in optimization and sparse linear systems*, Ph.D. thesis, University of Illinois at Urbana Champaing, 1997. 1.2

[KFS13] Dilip Krishnan, Raanan Fattal, and Richard Szeliski, *Efficient preconditioning of laplacian matrices for computer graphics*, ACM Trans. Graph. **32** (2013), no. 4, 142. 6

[KL51] S. Kullback and R. A. Leibler, *On information and sufficiency*, Ann. Math. Statist. **22** (1951), no. 1, 79–86. 5.4

[KL11] Jonathan A. Kelner and Alex Levin, *Spectral Sparsification in the Semi-Streaming Setting*, 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011) (Dagstuhl, Germany) (Thomas Schwentick and Christoph Dürr, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 9, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, pp. 440–451. 1.6.4, 3.4

[KLOS13] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford, *An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations*, CoRR **abs/1304.2338** (2013). 5.8

[KLP12] Ioannis Koutis, Alex Levin, and Richard Peng, *Improved Spectral Sparsification and Numerical Algorithms for SDD Matrices*, 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012) (Dagstuhl, Germany) (Christoph Dürr and Thomas Wilke, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 266–277. 1.4, 3.1, 3.4

[KM07] Ioannis Koutis and Gary L. Miller, *A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians*, Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007. 1.2, 3, 5.2.1

[KM09a] Jonathan A. Kelner and Aleksander Mądry, *Faster generation of random spanning trees*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '09, IEEE Computer Society, 2009, pp. 13–21. 1.4

[KM09b] Ioannis Koutis and Gary Miller, *The combinatorial multigrid solver*, Conference Talk, March 2009. 5.5

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng, *Approaching optimality for solving sdd linear systems*, Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '10, IEEE Computer Society, 2010, pp. 235–244. 1.3.1, 1.6.4, 2, 2.3, B

[KMP11] ______, *A nearly-m log n time solver for sdd linear systems*, Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '11, IEEE Computer Society, 2011, pp. 590–598. 1.3.1, 2, 2.2, 2.4, 4, 4.1, 5.1.1

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng, *Faster approximate multicommodity flow using quadratically coupled flows*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 1–18. 5.1.1, 5.2.2, 5.8

[KMST09] Ioannis Koutis, Gary L. Miller, Ali Sinop, and David Tolliver, *Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing*, Tech. report, CMU, 2009. 1.4, 6

[KMST10] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng, *Subgraph sparsification and nearly optimal ultrasparsifiers*, Proceedings of the 42nd ACM symposium on Theory of computing (New York, NY, USA), STOC '10, ACM, 2010, pp. 57–66. 2, 6.2

[KMT09] Ioannis Koutis, Gary L. Miller, and David Tolliver, *Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing*, Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I (Berlin, Heidelberg), ISVC '09, Springer-Verlag, 2009, pp. 1067–1078. 5.5

[KOSZ13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu, *A simple, combinatorial algorithm for solving sdd systems in nearly-linear time*, Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (New York, NY, USA), STOC '13, ACM, 2013, pp. 911–920. 1.1, 1.2, 1.3.2, 3, 4

[KP12] Michael Kapralov and Rina Panigrahy, *Spectral sparsification via random spanners*, Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (New York, NY, USA), ITCS '12, ACM, 2012, pp. 393–398. 3.3

[KS97] Philip N. Klein and Sairam Subramanian, *A randomized parallel algorithm for single-source shortest paths*, Journal of Algorithms **25** (1997), 205–220. 4.2.3

[KZ04] Vladimir Kolmogorov and Ramin Zabih, *What energy functions can be minimized via graph cuts*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004), 65–81. 5.3.2

[Lan52] Cornelius Lanczos, *Solution of systems of linear equations by minimized iterations*, Journal of Research of the National Bureau of Standards **49** (1952), 33–53. 1.2

[Lei92] F. Thomson Leighton, *Introduction to parallel algorithms and architectures: Array, trees, hypercubes*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992. 4.3.1

[LNK07] David Liben-Nowell and Jon M. Kleinberg, *The link-prediction problem for social networks*, JASIST **58** (2007), no. 7, 1019–1031. 1.1

[LR99] Tom Leighton and Satish Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM **46** (1999), no. 6, 787–832. 4.2

[LRS13] Yin Tat Lee, Satish Rao, and Nikhil Srivastava, *A new approach to computing maximum flows using electrical flows*, Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (New York, NY, USA), STOC '13, ACM, 2013, pp. 755–764. 1.4

[LRT79] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. on Numerical Analysis **16** (1979), 346–358. 3

[LS93] Nathan Linial and Michael Saks, *Low diameter graph decompositions*, Combinatorica **13** (1993), no. 4, 441–454. 4.2, 1

[LS10] Charles E. Leiserson and Tao B. Schardl, *A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)*, Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures (New York, NY, USA), SPAA '10, ACM, 2010, pp. 303–314. 4.2.3

[LS13] Yin Tat Lee and Aaron Sidford, *Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems*, CoRR **abs/1305.1922** (2013). 1.2, 1.3.2, 3

[LW94] Nick Littlestone and Manfred K. Warmuth, *The weighted majority algorithm*, Inf. Comput. **108** (1994), no. 2, 212–261. 5.4

[Mąd13] Aleksander Mądry, *Navigating central path with electrical flows: from flows to matchings, and back*, CoRR **abs/1307.2205** (2013). 1.4

[Mah11] Michael W. Mahoney, *Randomized algorithms for matrices and data.*, Foundations and Trends in Machine Learning **3** (2011), no. 2, 123–224. 1.6.4

[MLZ$^+$09] Patrycja Vasilyev Missiuro, Kesheng Liu, Lihua Zou, Brian C. Ross, Guoyan Zhao, Jun S. Liu, and Hui Ge, *Information flow analysis of interactome networks*, PLoS Comput Biol **5** (2009), no. 4, e1000350. 1.1

[MP08] James McCann and Nancy S. Pollard, *Real-time gradient-domain painting*, ACM Trans. Graph. **27** (2008), no. 3, 1–7. 1.4

[MP13] Gary L. Miller and Richard Peng, *Approximate maximum flow on separable undirected graphs*, Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 1151–1170. 5.8

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu, *Parallel graph decompositions using random shifts*, Proceedings of the 25th ACM symposium on Parallelism in algorithms and architectures (New York, NY, USA), SPAA '13, ACM, 2013, pp. 196–203. 1.3.4

[MR89] Gary L. Miller and John H. Reif, *Parallel tree contraction part 1: Fundamentals*, Randomness and Computation (Silvio Micali, ed.), JAI Press, Greenwich, Connecticut, 1989, Vol. 5, pp. 47–72. 4.3

[MS89] D. Mumford and J. Shah, *Optimal approximations by piecewise smooth functions and associated variational problems*, Communications on Pure and Applied Mathematics **42** (1989), 577–685. 5.1, 5.8

[MSX11] Charles A Micchelli, Lixin Shen, and Yuesheng Xu, *Proximity algorithms for image models: denoising*, Inverse Problems **27** (2011), no. 4, 045009. (document), 5.5.1, 5.1

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica **7** (1987), no. 1, 105–113. 5.6

[Nes07] Yu. Nesterov, *Gradient methods for minimizing composite objective function*, CORE Discussion Papers 2007076, UniversitÃl' catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2007. 5, 5.3.4

[OSV12] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi, *Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 1141–1160. 1.4

[PGB03] P. Pérez, M. Gangnet, and A. Blake, *Poisson image editing*, ACM Transactions on Graphics (SIGGRAPH'03) **22** (2003), no. 3, 313–318. 5.2.3, 5.5.2

[RDVC$^+$04] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri, *Are loss functions all the same?*, Neural Computation **16** (2004), no. 5, 1063–1076. 5.8

[ROF92] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithm*, Physica D **1** (1992), no. 60, 259–268. 5, 5, 5.5.1

[RV07] Mark Rudelson and Roman Vershynin, *Sampling from large matrices: An approach through geometric functional analysis*, J. ACM **54** (2007), no. 4, 21. 1.6.4, B

[Saa03] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003. 2.1

[SB13] Julian Shun and Guy E. Blelloch, *Ligra: a lightweight graph processing framework for shared memory*, Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming (New York, NY, USA), PPoPP '13, ACM, 2013, pp. 135–146. 4.2.3

[Sey95] P. D. Seymour, *Packing directed circuits fractionally*, Combinatorica **15** (1995), no. 2, 281–288. 4, 4.4, 6.1

[She09] Jonah Sherman, *Breaking the multicommodity flow barrier for $O(\sqrt{logn})$-approximations to sparsest cut*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '09, IEEE Computer Society, 2009, pp. 363–372. 4.2

[She13] Jonah Sherman, *Nearly maximum flows in nearly linear time*, CoRR **abs/1304.2077** (2013). 5.8

[SM97] Jianbo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (1997), 888–905. 1.4

[SS08] Daniel A. Spielman and Nikhil Srivastava, *Graph sparsification by effective resistances*, Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC), 2008, pp. 563–568. 1.4, 1.6.4, 1.6.4, 1.6.4, 3.4, B

[ST96] D. A. Spielman and Shang-Hua Teng, *Spectral partitioning works: planar graphs and finite element meshes*, Proc. of the 37th Annual Symposium on the Foundations of Computer Science (FOCS), 1996, pp. 96–105. 1.4

[ST03] Daniel A. Spielman and Shang-Hua Teng, *Solving sparse, symmetric, diagonally-dominant linear systems in time 0(m1.31)*, Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '03, IEEE Computer Society, 2003, pp. 416–. 1.2

[ST06] Daniel A. Spielman and Shang-Hua Teng, *Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems*, CoRR **abs/cs/0607105** (2006). 1.1, 1.2, 1.3.1, 1.3.2, 1.4, 1.5.1, 1.6.2, 2, 2.1, 2.1, 2.2, 3, 4, 4.1, 4.4, 5, 5.1.1

[ST08] ———, *Spectral sparsification of graphs*, CoRR **abs/0808.4134** (2008). 1.2, 3.3

[Sto10] A.J. Stothers, *On the complexity of matrix multiplication*, Ph.D. thesis, University of Edinburgh, 2010. 1.2

[Str69] Volker Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik **13** (1969), 354–356. 1.2, 5.1.1

[SW09] Daniel A. Spielman and Jaeoh Woo, *A note on preconditioning by low-stretch spanning trees*, CoRR **abs/0903.2816** (2009). B.1

[Tan11] Kanat Tangwongsan, *Efficient parallel approximation algorithms*, Ph.D. thesis, Carnegie Mellon University, 5000 Forbes Ave. Pittsburgh, PA 15213, August 2011, CMU CS Tech Report CMU-CS-11-128. 4.3.1

[Tar79] Robert Endre Tarjan, *Applications of path compression on balanced trees*, J. ACM **26** (1979), no. 4, 690–715. 2.2

[TB97] L.N. Trefethen and D.I. Bau, *Numerical linear algebra*, Society for Industrial and Applied Mathematics, 1997. 6.2

[Ten10] Shang-Hua Teng, *The Laplacian Paradigm: Emerging Algorithms for Massive Graphs*, Theory and Applications of Models of Computation, 2010, pp. 2–14. 1

[Tho65] C. J. Thompson, *Inequality with applications in statistical mechanics*, Journal of Mathematical Physics **6** (1965), 1812–1813. 1

[Tib96] R. Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society (Series B) **58** (1996), 267–288. 5

[Tri02] Kishor S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*, 2nd edition ed., John Wiley and Sons Ltd., Chichester, UK, 2002. 4.2.1

[Tro12] Joel A. Tropp, *User-friendly tail bounds for sums of random matrices*, Found. Comput. Math. **12** (2012), no. 4, 389–434. 1.6.4

[TSR$^+$05] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight, *Sparsity and smoothness via the fused lasso*, Journal of the Royal Statistical Society Series B **67** (2005), no. 1, 91–108. 5.1

[Tut63] W. T. Tutte, *How to Draw a Graph*, Proceedings of the London Mathematical Society **s3-13** (1963), no. 1, 743–767. 1.4

[Vai91] Pravin M. Vaidya, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners*, A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991. 1.2, 1.5.1

[Ver09] R. Vershynin, *A note on sums of independent random matrices after ahlswede-winter*, http://www-personal.umich.edu/~romanv/teaching/reading-group/ahlswede-winter.pdf, 2009. 1.6.4, B.1

[Wil12] Virginia Vassilevska Williams, *Multiplying matrices faster than coppersmith-winograd*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 887–898. 1.2, 1.5.2, 2.2, 2.2, 5.1.1

[WR07] B. Wohlberg and P. Rodriguez, *An iteratively reweighted norm algorithm for minimization of total variation functionals*, Signal Processing Letters, IEEE **14** (2007), no. 12, 948 –951. 5.3.3

[YL06] Ming Yuan and Yi Lin, *Model selection and estimation in regression with grouped variables*, Journal of the Royal Statistical Society, Series B **68** (2006), 49–67. 5.1

Appendix A

Deferred Proofs

We now formally prove some of the results stated in Section 1.6 about matrix norms, solver operators, and spectral approximation.

Lemma 1.6.1 *If $p(x)$ is a polynomial, then:*

$$p(\boldsymbol{A}) = \sum_{i=1}^{n} p(\lambda_i) \boldsymbol{u}_i \boldsymbol{u}_i^T$$

Proof Since both sides are linear in the monomials, it suffices to prove the claim for monomials. We show by induction that for any t,

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

The base case of $t = 0$ follows from $\mathbf{U}_i$ being an orthonormal basis. For the inductive case, suppose the result is true for t, then writing $\mathbf{A}^{t+1} = \mathbf{A}^t \mathbf{A}$ and substituting the inductive hypothesis gives:

$$\begin{aligned}
\mathbf{A}^{t+1} &= \left(\sum_{i=1}^{n} \lambda_i^t \mathbf{u}_i \mathbf{u}_i^T \right) \mathbf{A} \\
&= \left(\sum_{i=1}^{n} \lambda_i^t \mathbf{u}_i \mathbf{u}_i^T \right) \left(\sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^T \right) \mathbf{A} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i^t \lambda_j \mathbf{u}_i \mathbf{u}_i^T \mathbf{u}_j \mathbf{u}_j^T
\end{aligned}$$

Since $\mathbf{u}_i$ form an orthonormal basis, the only non-zero terms are the ones with $i = j$. These terms in turn leads to:

$$\mathbf{A}^{t+1} = \sum_{j=1}^{n} \lambda_i^{t+1} \mathbf{u}_i \mathbf{u}_i^T$$

So the inductive hypothesis holds for $t+1$ as well. ■

Lemma 1.6.6 (Composition of Spectral Ordering) *For any matrices $\boldsymbol{V}, \boldsymbol{A}, \boldsymbol{B}$, if $\boldsymbol{A} \preceq \boldsymbol{B}$, then $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} \preceq \boldsymbol{V}^T\boldsymbol{B}\boldsymbol{V}$*

Proof For any vector $\mathbf{x}$, if we let $\mathbf{y} = \mathbf{V}\mathbf{x}$, then we have:

$$\begin{aligned}\mathbf{x}^T\mathbf{V}^T\mathbf{A}\mathbf{V}\mathbf{x} &= \mathbf{y}^T\mathbf{A}\mathbf{y}\\ &\leq \mathbf{y}^T\mathbf{B}\mathbf{y} \qquad \text{By definition of } \mathbf{A} \preceq \mathbf{B}\\ &= \mathbf{x}^T\mathbf{V}^T\mathbf{B}\mathbf{V}\mathbf{x}\end{aligned}$$

■

Lemma 1.6.7 *If $\boldsymbol{A}$ is a symmetric positive semi-definite matrix and $\boldsymbol{B}$ is a matrix such that:*

$$(1-\epsilon)\boldsymbol{A}^\dagger \preceq \boldsymbol{B} \preceq (1+\epsilon)\boldsymbol{A}^\dagger$$

then for any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$, $\boldsymbol{x} = \boldsymbol{B}\boldsymbol{b}$ satisfies:

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \epsilon \|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

Proof Applying Lemma 1.6.6 to both sides of the given condition with $\mathbf{A}^{1/2}$ gives:

$$(1-\epsilon)\,\mathbf{\Pi} \preceq \mathbf{B} \preceq (1+\epsilon)\,\mathbf{\Pi}$$

Where $\mathbf{\Pi}$ is the projection matrix onto the rank space of $\mathbf{A}$. Rearranging the LHS of the error term gives:

$$\begin{aligned}\|\bar{\mathbf{x}} - \mathbf{x}\|_{\mathbf{A}}^2 &= \left\|\mathbf{A}^\dagger\mathbf{b} - \mathbf{B}\mathbf{b}\right\|_{\mathbf{A}}^2\\ &= \mathbf{b}^T\left(\mathbf{A}^\dagger - \mathbf{B}\right)\mathbf{A}\left(\mathbf{A}^\dagger - \mathbf{B}\right)\mathbf{b}\\ &= \mathbf{b}^T\mathbf{A}^{\dagger 1/2}\left(\mathbf{\Pi} - \mathbf{A}^{1/2}\mathbf{B}\mathbf{A}^{1/2}\right)^2\mathbf{A}^{\dagger 1/2}\mathbf{b}^T\end{aligned}$$

Where $\mathbf{A}^{\dagger 1/2}$ is the $1/2$ power of the pseudoinverse of $\mathbf{A}$.

Note that the given condition implies that all eigenvalues of $\mathbf{\Pi} - \mathbf{A}^{1/2}\mathbf{B}\mathbf{A}^{1/2}$ are between $-\epsilon$ and ϵ. Therefore $\left(\mathbf{\Pi} - \mathbf{A}^{1/2}\mathbf{B}\mathbf{A}^{1/2}\right)^2 \preceq \epsilon^2\mathbf{I}$ and we have:

$$\begin{aligned}\|\bar{\mathbf{x}} - \mathbf{x}\|_{\mathbf{A}}^2 &\leq \epsilon^2\mathbf{b}^T\mathbf{A}^{\dagger 1/2}\mathbf{I}\mathbf{A}^{\dagger 1/2}\mathbf{b}^T\\ &= \epsilon^2\mathbf{b}^T\mathbf{A}^\dagger\mathbf{b}\\ &= \epsilon^2\left\|\mathbf{A}^\dagger\mathbf{b}\right\|_{\mathbf{A}}^2 = \epsilon^2\|\bar{\mathbf{x}}\|_{\mathbf{A}}^2\end{aligned}$$

Taking square roots of both sides then gives the error bound. ■

Appendix B

Spectral Sparsification by Sampling

Before describing and proving the sparsification algorithms, we need to describe a way to decompose a graph Laplacian into a sum of outer-products. Our basic building block is the indicator vector $\boldsymbol{\chi}_u$, which is 1 in position u and 0 everywhere else. Then an edge $e = uv$ can also correspond to the vector $\boldsymbol{\chi}_{uv} = \boldsymbol{\chi}_u - \boldsymbol{\chi}_v$, which is 1 in u, -1 in v and zero everywhere else. This is one row of the edge-vertex incidence matrix ∂, and the graph Laplacian for a graph $G = (V, E, \mathbf{w})$ can be written as:

$$\mathbf{L}_G = \sum_{e=uv} \mathbf{w}_e \boldsymbol{\chi}_{uv} \boldsymbol{\chi}_{uv}^T$$

Recall that the effective resistance of an edge, R_e was defined as:

$$R_e = \boldsymbol{\chi}_{u,v}^T \mathbf{L}_G^{\dagger} \boldsymbol{\chi}_{u,v}$$

and its statistical leverage score is the product of this value and the weight of the edge:

$$\begin{aligned}\tau_e &= \mathbf{w}_e R_e \\ &= \boldsymbol{\chi}_{u,v}^T \mathbf{L}_G^{\dagger} \boldsymbol{\chi}_{u,v}\end{aligned}$$

It was shown in [SS08] that if we sample each edge with probabilities $c_s \tau_e \log n \epsilon^{-2}$ for some absolute constant c_s, the resulting graph is a $1 + \epsilon$-approximation. Their proof was based on a matrix-concentration bound by Rudelson and Versynin [RV07], which can be viewed as a stronger variant of an earlier result by Ahlswede and Winter [AW02]. This sampling scheme was used in [KMP10] with one slight modification: instead of computing R_e with respect to the entire graph, an upper bound R'_e was computed using an (artificially) heavy spanning tree. We will show the following concentration bound that allows us to show the convergence of both of these processes.

Lemma B.0.1 *Given $n \times n$ positive semi-definite matrices $\boldsymbol{A} = \sum_{i=1}^{m} \boldsymbol{y}_i^T \boldsymbol{y}_i$ and $\boldsymbol{B}$ such that the image space of $\boldsymbol{A}$ is contained in the image space of $\boldsymbol{B}$, along with estimates $\tilde{\tau}_i$ such that:*

$$\tilde{\tau}_i \geq \boldsymbol{y}_i^T \boldsymbol{B}^{\dagger} \boldsymbol{y}_i \qquad \forall 1 \leq i \leq m$$

Then for any error ϵ and any failure probability $\delta = n^{-d}$, there exists a constant c_s such that if we construct $\tilde{\boldsymbol{A}}$ using by repeating the following process $N = c_s \ln n \epsilon^{-2} \left\|\tilde{\tau}_i\right\|_1$ times ($\left\|\tilde{\tau}_i\right\|_1$ is the total of all the estimates):

- *Pick index i' with probability proportional to $\tilde{\tau}_i$*
- *Add $\frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i} \boldsymbol{y}_i \boldsymbol{y}_i^T$ to $\tilde{\boldsymbol{A}}$*

Then with probability at least $1 - \delta = 1 - n^{-d}$, $\boldsymbol{B}$ satisfies:

$$\boldsymbol{A} - \epsilon\left(\boldsymbol{A} + \boldsymbol{B}\right) \preceq \tilde{\boldsymbol{A}} \preceq \boldsymbol{A} + \epsilon\left(\boldsymbol{A} + \boldsymbol{B}\right)$$

The use of $\ln \cdot$ instead of $\log \cdot$ in the notation is for convenience of proof. For algorithmic purposes they're interchangeable with the use of a different constant c_s. We will first show in Section B.1 that Lemma B.0.1 implies the bounds on sparsification algorithms, and then prove this concentration bound in Section B.2.

B.1 From Matrix Chernoff Bounds to Sparsification Guarantees

Both sparsification by effective resistance in Theorem 1.6.9 and sampling off-tree edges by stretch in Lemma 2.2.2 are direct consequences of this concentration bound. We first give bounds for sampling by effective resistance.

Proof of Theorem 1.6.9:

Consider applying Lemma B.0.1 with $\mathbf{A} = \mathbf{B} = \mathbf{L}_G$ and error $\frac{\epsilon}{8}$. For each edge e, we map it to the vector $\mathbf{y}_i = \sqrt{\mathbf{w}_e} \boldsymbol{\chi}_e$. It can be checked that:

$$\begin{aligned}\sum_{i=1}^{m} \mathbf{y}_i \mathbf{y}_i^T &= \sum_{i=1}^{m} \mathbf{w}_e \boldsymbol{\chi}_e \boldsymbol{\chi}_e^T \\ &= \mathbf{L}_G = \mathbf{A}\end{aligned}$$

Also, the definition of τ_i is identical to $\mathbf{y}_i^T \mathbf{B}^\dagger \mathbf{y}_i$. Therefore, if the resulting matrix given by the sampling procedure is $\mathbf{L}_H = \tilde{\mathbf{A}}$, we have with probability at least $1 - n^{-d}$:

$$\left(1 - \frac{\epsilon}{4}\right) \mathbf{L}_G \preceq \mathbf{L}_H \preceq \left(1 + \frac{\epsilon}{4}\right) \mathbf{L}_H$$

Rescaling $\mathbf{L}_H$ by $(1 - \frac{\epsilon}{4})^{-1}$ and using the fact that $\left(1 + \frac{\epsilon}{4}\right)\left(1 - \frac{\epsilon}{4}\right)^{-1} \leq 1 + \epsilon$ gives the desired bound. ■

For sampling with stretch, it was formally shown in [SW09] that $\mathbf{str}_T(e) = \boldsymbol{\chi}_e^T \mathbf{L}_T^\dagger \boldsymbol{\chi}_e$. So applying the same concentration bound from Lemma B.0.1 except with $\mathbf{A}$ set to the off-tree edges and $\mathbf{B}$ set to the tree gives the bound.

Proof of Lemma 2.2.2:

Let $\mathbf{L}_{G\setminus T}$ denote the graph Laplacian for the off-tree edges. We invoke Lemma B.0.1 with $\mathbf{A} = \mathbf{L}_{G\setminus T}$, $\mathbf{B} = \mathbf{L}_T$, error $\frac{1}{4}$, and the same mapping of $\mathbf{y}_i$s. For any edge e which is mapped to index i, we have:

$$\begin{aligned}\mathbf{y}_i^T \mathbf{B}^\dagger \mathbf{y}_i &= \boldsymbol{\chi}_e^T \mathbf{L}_T^\dagger \\ &= \mathbf{str}_T(e)\end{aligned}$$

Let the resulting graph be $H \setminus T$, we have with probability at least $1 - n^{-d}$:

$$\mathbf{L}_{G\setminus T} - \frac{1}{4}\mathbf{L}_G \preceq \mathbf{L}_{H\setminus T} \preceq \mathbf{L}_{G\setminus T} + \frac{1}{4}\mathbf{L}_G$$

Then if we set $\mathbf{L}_H$ to $\mathbf{L}_{H\setminus T} + \mathbf{L}_T$ we have:

$$\frac{3}{4}\mathbf{L}_G \preceq \mathbf{L}_H \preceq \frac{5}{4}\mathbf{L}_T$$

Therefore returning $\frac{4}{3}\mathbf{L}_H$ satisfies the required bounds. ■

In the rest of this Chapter we will prove Lemma B.0.1. Our presentation closely mirrors the ones in [Ver09, Har11],

B.2 Proof of Concentration Bound

We now set forth proving Lemma B.0.1. We first show that it suffices to prove the claim for the case where $\mathbf{A} + \mathbf{B} = \mathbf{I}$.

Lemma B.2.1 *Lemma B.0.1 restricted to the setting $\mathbf{A}+\mathbf{B} = \mathbf{I}$ implies the general version without this restriction.*

Proof Since the image spaces of $\mathbf{A}$ and $\mathbf{B}$ are contained in the image space of $\mathbf{A} + \mathbf{B}$, it suffices to consider the case where $\mathbf{A} + \mathbf{B}$ is full rank. This restriction also implies that $\mathbf{B}$ is full rank, as we can use inverses in place of pseudoinverses in our proofs.

Consider a modified problem with $\hat{\mathbf{A}} = (\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\mathbf{A}(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}$ and $\hat{\mathbf{B}} = (\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\mathbf{B}(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}$. Then the decomposition of $\hat{\mathbf{A}}$ in turn becomes $\hat{\mathbf{A}} = \sum_{i=1}^m \hat{\mathbf{y}}^T\mathbf{y}$ where $\hat{\mathbf{y}} = (\mathbf{A} + \mathbf{B})^{-\frac{1}{2}}\mathbf{y}$.

Then $\hat{\mathbf{A}} + \hat{\mathbf{B}} = (\mathbf{A} + \mathbf{B})^{-\frac{1}{2}}(\mathbf{A} + \mathbf{B})(\mathbf{A} + \mathbf{B})^{-\frac{1}{2}} = \mathbf{I}$. And it suffices to check that both the sampling process and required spectral bounds are equivalent. Both parts can be proven by direct algebraic manipulations. For the sampling process, it suffices to check that the leverage scores are same as before:

$$\begin{aligned}\hat{\mathbf{y}}^T\hat{\mathbf{B}}^{-1}\hat{\mathbf{y}} &= \mathbf{y}^T(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\left((\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\mathbf{B}(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\right)^{-1}(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\mathbf{y}^T \\ &= \mathbf{y}^T(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}(\mathbf{A}+\mathbf{B})^{\frac{1}{2}}\mathbf{B}^{-1}(\mathbf{A}+\mathbf{B})^{\frac{1}{2}}(\mathbf{A}+\mathbf{B})^{-\frac{1}{2}}\mathbf{y}^T \\ &= \mathbf{y}^T\mathbf{B}^{-1}\mathbf{y}\end{aligned}$$

For the spectral conditions, we will only show the LHS sides are implied by each other, as the RHS follows analogously. The Courant-Fischer theorem as stated in Lemma 3.3.9 gives $\mathbf{A}-\epsilon\left(\mathbf{A}+\mathbf{B}\right) \preceq \sum_{i=1}^{m} s_i \mathbf{y}_i \mathbf{y}_i^T$ if and only if for all vectors $\mathbf{x} \in \mathbb{R}^n$, we have:

$$\mathbf{x}^T \mathbf{A} \mathbf{x} - \epsilon\left(\mathbf{A}+\mathbf{B}\right)\mathbf{x} \leq \mathbf{x}^T \left(\sum_{i=1}^{m} s_i \mathbf{y}_i \mathbf{y}_i^T\right) \mathbf{x}$$

. Consider the reversible mapping $\mathbf{x} \leftrightarrow \hat{\mathbf{x}} = \left(\mathbf{A}+\mathbf{B}\right)^{\frac{1}{2}} \mathbf{x}$. We have:

$$\begin{aligned}
\hat{\mathbf{x}}^T \left(\sum_{i=1}^{m} s_i \hat{\mathbf{y}}_i \hat{\mathbf{y}}_i^T\right) \hat{\mathbf{x}} &= \sum_{i=1}^{m} s_i \hat{\mathbf{x}}^T \hat{\mathbf{y}}_i \hat{\mathbf{y}}_i^T \hat{\mathbf{x}} = \sum_{i=1}^{m} s_i \left(\hat{\mathbf{x}}^T \hat{\mathbf{y}}_i\right)^2 \\
&= \sum_{i=1}^{m} s_i \left(\mathbf{x}^T \left(\mathbf{A}+\mathbf{B}\right)^{\frac{1}{2}} \left(\mathbf{A}+\mathbf{B}\right)^{-\frac{1}{2}} \mathbf{y}_i\right)^2 = \sum_{i=1}^{m} s_i \left(\mathbf{x}^T \mathbf{y}_i\right)^2
\end{aligned}$$

Where the last equality follows from $\mathbf{y}_i$ being in the image space of $\mathbf{A}$ and therefore $\mathbf{B}$. Setting $s_i = 1$ also gives that $\hat{\mathbf{x}}^T \hat{\mathbf{A}} \hat{\mathbf{x}} = \mathbf{x}^T \mathbf{A} \mathbf{x}$. It remains to check that $\hat{\mathbf{x}}^T \hat{\mathbf{B}} \hat{\mathbf{x}} = \mathbf{x}^T \mathbf{B} \mathbf{x}$:

$$\begin{aligned}
\hat{\mathbf{x}}^T \hat{\mathbf{B}}^{-1} \hat{\mathbf{x}} &= \mathbf{x}^T \left(\mathbf{A}+\mathbf{B}\right)^{\frac{1}{2}} \left(\mathbf{A}+\mathbf{B}\right)^{-\frac{1}{2}} \mathbf{B} \left(\mathbf{A}+\mathbf{B}\right)^{-\frac{1}{2}} \mathbf{B}^{\frac{1}{2}} \mathbf{x} \\
&= \mathbf{x}^T \mathbf{B} \mathbf{x}
\end{aligned}$$

■

Therefore from here and on we will assume that $\mathbf{A}+\mathbf{B} = \mathbf{I}$. As $\mathbf{A}, \mathbf{B}$ are positive semi-definite, this also implies that $\mathbf{A}, \mathbf{B} \preceq \mathbf{I}$. We start of with some notations on the sampling procedure Let the iterations be $j = 1 \ldots N$ and i_j be the random variable corresponding to the rows picked in iterations j. The final matrix generated by the procedure, $\tilde{\mathbf{A}}$ is a randomized matrix, and we will denote it using the random variable $\mathcal{Y}$. We will also use $\mathcal{Y}_j$ to denote the random matrix added to $\tilde{\mathbf{A}}$ in iteration j. We first check that the procedure returns $\mathbf{A}$ in expectation:

Lemma B.2.2

$$\mathop{\mathbf{E}}_{i_1 \ldots i_N} \left[\mathcal{Y}\right] = \boldsymbol{A}$$

Proof By linearity of expectation, it suffices to check that each sample's expectation is $\frac{1}{N}\mathbf{A}$.

$$\begin{aligned}
\mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j\right] &= \sum_{i=1}^{m} \mathbf{Pr}\left[i_j = i\right] \frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i} \mathbf{y}_i \mathbf{y}_i^T \\
&= \sum_{i=1}^{m} \frac{\tilde{\tau}_i}{\|\tilde{\tau}\|_1} \frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i} \mathbf{y}_i \mathbf{y}_i^T \\
&= \sum_{i=1}^{m} \frac{\epsilon^2}{c_s \ln n \|\tilde{\tau}\|_1} \mathbf{y}_i \mathbf{y}_i^T \\
&= \frac{1}{N} \sum_{i=1}^{m} \mathbf{y}_i \mathbf{y}_i^T
\end{aligned}$$

■

Therefore, bounding the difference between $\tilde{\mathbf{A}}$ and $\mathbf{A}$ is essentially bounding the deviation of $\mathcal{Y}$ from its expectation. Once again, we can measure this per iteration. At iteration j, let $\mathcal{X}_j$ be random variable corresponding to $\mathcal{Y}_j - \mathbf{E}_{i_j}\left[\mathcal{Y}_j\right]$. Also, let $\mathcal{S}_0 \dots \mathcal{S}_N$ be random variables corresponding to the partial sums of the $\mathcal{X}_j$s:

$$\mathcal{S}_j = \sum_{j'=1}^{j} \mathcal{X}_{j'}$$

Our goal is then to bound the spectral norm of: $\mathcal{S}_N = \sum_{j=1}^{N} \mathcal{X}_i$. For its maximum eigenvalue, our goal is to measure the probability that:

$$\mathbf{Pr}\left[\lambda_{\max}(\mathcal{S}_n) > \epsilon\right]$$

We will bound this probability by taking the exponential of $\mathcal{S}_n$ and bounding its trace. We have the following for any multiplier δ:

$$\begin{aligned}
\mathbf{Pr}\left[\lambda_{\max}(\mathcal{S}_n) > \epsilon\right] &\leq \mathbf{Pr}\left[\mathbf{tr}\left[e^{\delta \mathcal{S}_n}\right] > e^{\delta\epsilon}\right] \\
&\leq e^{-\delta\epsilon} \mathop{\mathbf{E}}_{i_1 \dots i_N}\left[\mathbf{tr}\left[e^{\delta \mathcal{S}_n}\right]\right] \qquad \text{By Markov's inequality} \qquad \text{(B.1)}
\end{aligned}$$

The term inside the expectation is a the trace of a matrix exponential. It can be decomposed using the following two facts:

1. Golden Thompson Inequality [Gol65, Tho65]:

$$\mathbf{tr}\left[e^{A+B}\right] \leq \mathbf{tr}\left[e^{A}e^{B}\right]$$

2. Inequalities involving matrix norm and trace:

$$\mathbf{tr}\left[\mathbf{UV}\right] \leq \left\|\mathbf{U}\right\|_2 \mathbf{tr}\left[\mathbf{V}\right]$$

Lemma B.2.3

$$\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_n} \left[\boldsymbol{tr}\left[e^{\delta \mathcal{S}_n}\right]\right] \leq n \cdot \prod_{j=1}^{N} \left\|\mathbf{E}\left[\mathcal{X}_j\right] e^{\mathcal{X}_j}\right\|_2$$

$$\begin{aligned}
&\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_n} \left[\mathbf{tr}\left[e^{\delta \mathcal{S}_n}\right]\right] \\
\leq &\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_n} \left[\mathbf{tr}\left[e^{\delta \mathcal{X}_n} \cdot e^{\delta \mathcal{S}_{n-1}}\right]\right] && \text{(by the Golden-Thompson inequality)} \\
= &\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_{n-1}} \left[\mathbf{tr}\left[\mathop{\mathbf{E}}_{\mathcal{X}_n}\left[e^{\delta \mathcal{X}_n} \cdot e^{\delta \mathcal{S}_{n-1}}\right]\right]\right] && \text{By linearity of expectation} \\
= &\mathop{\mathbf{E}}_{X_1 \ldots X_{n-1}} \left[\mathbf{tr}\left[\mathop{\mathbf{E}}_{X_n}\left[e^{\delta X_n}\right] \cdot e^{\delta S_{n-1}}\right]\right] && \text{By independence of } \mathcal{X}_n \text{ with } \mathcal{X}_1 \ldots \mathcal{X}_{n-1} \\
\leq &\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_{n-1}} \left[\left\|\mathop{\mathbf{E}}_{\mathcal{X}_n}\left[e^{\delta \mathcal{X}_\backslash}\right]\right\|_2 \cdot e^{\delta S_{n-1}}\right] && \text{since } \mathbf{tr}\left[\mathbf{UV}\right] \leq \left\|\mathbf{U}\right\|_2 \mathbf{tr}\left[\mathbf{V}\right]
\end{aligned}$$

Applying this inductively to $\mathcal{X}_{n-1} \ldots \mathcal{X}_1$ gives:

$$\begin{aligned}
&\mathop{\mathbf{E}}_{\mathcal{X}_1 \ldots \mathcal{X}_n} \left[\mathbf{tr}\left[e^{\delta \mathcal{S}_n}\right]\right] \\
\leq &\mathbf{tr}\left[e^{\delta \mathcal{S}_0}\right] \cdot \prod_{j=1}^{N} \left\|\mathop{\mathbf{E}}_{\mathcal{X}_j}\left[e^{\delta \mathcal{X}_j}\right]\right\|_2 \\
\leq &n \cdot \prod_{j=1}^{N} \left\|\mathop{\mathbf{E}}_{\mathcal{X}_j}\left[e^{\mathcal{X}_j}\right]\right\|_2
\end{aligned}$$

Where the last line follows from $\mathcal{S}_0$ being the zero matrix. ■

Therefore our goal now shifts to bounding $\left\|\mathbf{E}_{\mathcal{X}_j}\left[e^{\mathcal{X}_j}\right]\right\|_2$. Since the sampling process is independent over the iterations, this value is the same for all $1 \leq j \leq N$. The following fact can be derived from the Taylor expansion of e^{λ} when $\lambda \leq 1$:

$$1 + \lambda \leq e^{\lambda} \leq 1 + \lambda + \lambda^2$$

Turning this into matrix form gives that when $-\mathbf{I} \preceq \delta \mathcal{X}_j \preceq \mathbf{I}$:

$$\mathbf{I} + \delta \mathcal{X}_j \preceq e^{\delta \mathcal{X}_j} \preceq \mathbf{I} + \delta \mathcal{X}_j + \delta^2 \mathcal{X}_j^2$$

To apply this bound, we need $\left\|\mathcal{X}_j\right\|_2 \leq 1$. This can be obtained from our setup as follows:

Lemma B.2.4

$$-\frac{1}{N}\boldsymbol{I} \preceq \mathcal{X}_j \preceq \frac{\epsilon^2}{c_s \ln n}\boldsymbol{I}$$

Proof It suffices to show this bound for any case of $i_j = i$. Here we have:

$$\mathcal{X}_j = \frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i}\mathbf{y}_i\mathbf{y}_i^T - \frac{1}{N}\mathbf{A}$$

The LHS follows from $\mathbf{A} \preceq \mathbf{I}$. For the RHS, it suffices to upper bound the Frobenius norm of the first term. We have:

$$\left\|\frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i}\mathbf{y}_i\mathbf{y}_i^T\right\|_F = \frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i}\left\|\mathbf{y}_i\right\|_2^2$$

Note that $\tilde{\tau}_i \geq \mathbf{Y}_i\mathbf{B}^{-1}\mathbf{Y}_i$. As $\mathbf{B} \preceq \mathbf{I}$, $\mathbf{B}^{-1} \succeq \mathbf{I}$ and $\mathbf{Y}_i\mathbf{B}^{-1}\mathbf{Y} \geq \|\mathbf{y}_i\|_2^2$. Incorporating these gives:

$$\begin{aligned}\left\|\frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i}\mathbf{y}_i\mathbf{y}_i^T\right\|_F &\leq \frac{\epsilon^2}{c_s \ln n \left\|\mathbf{y}_i\right\|_2^2}\left\|\mathbf{y}_i\right\|_2^2 \\ &\leq \frac{\epsilon^2}{c_s \ln n}\end{aligned}$$

■

Therefore, as long as δ is reasonably small, we can apply Taylor expansion and in turn bound the spectral norm of $\mathcal{X}_j^2$.

Lemma B.2.5 *If* $\delta \leq c_s \ln n \epsilon^{-2}$*, we have:*

$$\left\|\mathop{\mathbf{E}}_{i_j}\left[e^{\delta \mathcal{X}_j}\right]\right\|_2 \leq e^{\frac{\delta^2 \epsilon^2}{N c_s \ln n}}$$

Proof

Lemma B.2.4 gives that for any value of i_j we have:

$$-\mathbf{I} \preceq \delta \mathcal{X}_j \preceq \mathbf{I}$$

Therefore we can apply Taylor expansion to obtain:

$$e^{\delta \mathcal{X}_j} \preceq \mathbf{I} + \delta \mathcal{X}_j + \delta^2 \mathcal{X}_j^2$$

Adding expectation and combining with $\mathbf{E}_{i_j}\left[X_i\right] = 0$ gives:

$$\mathop{\mathbf{E}}_{i_j}\left[e^{\delta \mathcal{X}_j}\right] \preceq \mathbf{I} + \delta^2 \mathop{\mathbf{E}}_{i_j}\left[\mathcal{X}_j^2\right]$$

So it remains to bound the term corresponding to the second moment:

$$\begin{aligned}
\mathop{\mathbf{E}}_{i_j}\left[\mathcal{X}_j^2\right] &= \mathop{\mathbf{E}}_{i_j}\left[\left(\mathcal{Y}_j - \frac{1}{N}\mathbf{A}\right)^2\right] \\
&= \mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j^2\right] - \frac{1}{N}\mathbf{A}\mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j\right] + \frac{1}{N^2}\mathbf{A}^2 \\
&= \mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j^2\right] - \frac{1}{N^2}\mathbf{A}^2 \qquad \text{Since } \mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j\right] = \frac{1}{N}\mathbf{A} \\
&\preceq \mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j^2\right]
\end{aligned}$$

We then consider the value of $\mathcal{Y}_j$ over choices of $i_j = i$:

$$\begin{aligned}
\mathop{\mathbf{E}}_{i_j}\left[\mathcal{Y}_j^2\right] &= \sum_i \frac{\tilde{\tau}_i}{\|\tilde{\tau}\|_1}\left(\frac{\epsilon^2}{c_s \ln n \tilde{\tau}_i}\right)^2 \mathbf{y}_i\mathbf{y}_i^T\mathbf{y}_i\mathbf{y}_i^T \\
&= \frac{\epsilon^2}{N c_s \ln n}\sum_i \frac{\|\mathbf{y}_i\|_2^2}{\tilde{\tau}_i}\mathbf{y}_i\mathbf{y}_i^T \\
&\preceq \frac{\epsilon^2}{N c_s \ln n}\mathbf{A} \preceq \frac{\epsilon^2}{N c_s \ln n}\mathbf{I}
\end{aligned}$$

Substituting this and applying the LHS of the bound on e^{λ} gives the result:

$$\begin{aligned}
\left\|\mathop{\mathbf{E}}_{i_j}\left[e^{\delta \mathcal{X}_j}\right]\right\|_2 &\leq 1 + \frac{\epsilon^2\delta^2}{N c_s \ln n} \\
&\leq e^{\frac{\epsilon^2\delta^2}{N c_s \ln n}}
\end{aligned}$$

■

We can now combine things to give the overall bound.

Proof of Lemma B.0.1:

Lemma B.2.3 along with the Markov inequality shown earlier gives:

$$\begin{aligned}
\mathbf{Pr}\left[\lambda_{\max}(\mathcal{S}_n) > \epsilon\right] \leq & e^{-\delta\epsilon} \mathop{\mathbf{E}}_{i_1 \ldots i_N} \left[\mathbf{tr}\left[e^{\delta \mathcal{S}_n}\right]\right] && \text{From Equation B.1} \\
\leq & e^{-\delta\epsilon} \cdot n \cdot \cdot \prod_{i=1}^{N} e^{\delta^2} \left\| \mathbf{E}\left[\mathcal{X}_j\right] e^{\mathcal{X}_j} \right\|_2 && \text{By Lemma B.2.3} \\
\leq & e^{-\delta\epsilon} \cdot n \cdot \prod_{i=1}^{N} e^{\frac{\delta^2 \epsilon^2}{c_s \ln nN}} && \text{By Lemma B.2.5} \\
= & e^{\ln n - \delta\epsilon + \frac{\delta^2 \epsilon^2}{c_s \ln n}}
\end{aligned}$$

Setting $\delta = \frac{1}{2}\epsilon^{-1} c_s \ln n$ gives:

$$\begin{aligned}
\mathbf{Pr}\left[\lambda_{\max}(\mathcal{S}_n) > \epsilon\right] \leq & e^{\ln n - \frac{1}{2} c_s \ln n + \frac{1}{4} c_s \ln n} \\
= & e^{\ln n - \frac{1}{4} c_s \ln n}
\end{aligned}$$

Hence the choice of $c_s = 4(d+2)$ suffices to give a probability of $e^{\ln n - d \ln n} = n^{-d-1}$. Probabilistic bound on the lower eigenvalue of $\mathcal{Y}$ follows similarly, and combining them via. union bound gives the overall bound. ■

Appendix C

Partial Cholesky Factorization

We now prove the guarantees of the greedy elimination steps used in the sequential algorithm in Chapter 2. These proofs can be found in most texts on numerical analysis (e.g. Chapter 9 of [Hig02]). However, we need to track additional parameters such as edge weights across $\log n$ levels of recursive calls. As a result, most of the bounds below include more edge weight related information in their statements.

Partial Cholesky factorization multiplies both sides of an equation by a lower triangular matrix. To avoid confusion with graph Laplacians, we will instead denote these as transposes of upper-triangular matrices, e.g. $\mathbf{U}^T$. Starting with the problem:

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Left multiplying both sides by $\mathbf{U}^T$ gives:

$$\mathbf{U}^T\mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{b}$$

To preserve symmetry, we insert $\mathbf{U}$ and $\mathbf{U}^{-1}$ between $\mathbf{A}$ and $\mathbf{x}$:

$$\mathbf{U}^T\mathbf{A}\mathbf{U}\mathbf{U}^{-1}\mathbf{x} = \mathbf{U}\mathbf{b}$$
$$\left(\mathbf{U}^T\mathbf{A}\mathbf{U}\right)\mathbf{U}^{-1}\mathbf{x} = \mathbf{U}^T\mathbf{b}$$

Then the solution can be given by solving a linear system in $\mathbf{B} = \mathbf{U}^T\mathbf{A}\mathbf{U}$.

$$\begin{aligned}\mathbf{U}^{-1}\mathbf{x} &= \left(\mathbf{U}^T\mathbf{A}\mathbf{U}\right)^{\dagger}\mathbf{U}^T\mathbf{b}\\ &= \mathbf{B}^{\dagger}\mathbf{U}^T\mathbf{b}\\ \mathbf{x} &= \mathbf{U}\mathbf{B}^{\dagger}\mathbf{U}^T\mathbf{b}\end{aligned}$$

Errors also propagate naturally:

Lemma C.0.1 *If $\boldsymbol{U}$ as given above has m_u non-zero off-diagonal entries and $\mathbf{Z_B}$ is a matrix such that $\boldsymbol{B}^{\dagger} \preceq \mathbf{Z_B} \preceq \kappa\boldsymbol{B}^{\dagger}$. Then $\mathbf{Z_A} = \boldsymbol{U}\mathbf{Z_B}\boldsymbol{U}^T$ satisfies $\boldsymbol{A}^{\dagger} \preceq \mathbf{Z_A} \preceq \kappa\boldsymbol{A}^{\dagger}$. Also, for any vector $\boldsymbol{b}$, $\mathbf{Z_A}\boldsymbol{b}$*

can be evaluated with an overhead of $O(m_u)$ to the cost of evaluating $\boldsymbol{Z_B b'}$.

Proof The approximation bound follows from $\mathbf{A}^\dagger = \mathbf{U}\mathbf{B}^\dagger\mathbf{U}^T$ and Lemma 1.6.6.

For the running time, since the diagonal entries of $\mathbf{U}$ form the identity, they incur no additional cost if we keep the original vector. The running time overhead then corresponds to multiplying by $\mathbf{U}^T$ and then $\mathbf{U}$. ■

C.1 Partial Cholesky Factorization on Graph Laplacians

Both of our removal procedures for low-degree vertices can be viewed as special cases of partial Cholesky factorization. The error and running time propagation of Lemma C.0.1 means it suffices to exhibit lower-triangular matrices with a constant number of off-diagonal entries, and show their effect on the combinatorial graph. We first consider the case of removing a vertex of degree 1.

Lemma 2.1.3 *Let G be a graph where vertex u has only one neighbor v. H is reducible to a graph H that's the same as G with vertex u and edge uv removed.*

Proof

Without loss of generality (by permuting rows/columns), we assume that this vertex corresponds to row/column 1, and its neighbor is in row 2 and column 2. Then the matrix $\mathbf{U}^T$ is:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

It can be checked that since the only non-zero entries in the first row/column are in row/column 2, applying this matrix leads to exact cancellation. ■

The case of pivoting out a degree 2 vertex can be done similarly.

Lemma 2.1.4 *Let G be a graph where vertex u has only two neighbors v_1, v_2 with edge weights $\boldsymbol{w}_{uv_1}$ and $\boldsymbol{w}_{uv_2}$ respectively. G is reducible a graph H that's the same as G except with vertex u and edges uv_1, uv_2 removed and weight*

$$\frac{\boldsymbol{w}_{uv_1}\boldsymbol{w}_{uv_2}}{\boldsymbol{w}_{uv_1}+\boldsymbol{w}_{uv_2}} = \frac{1}{\frac{1}{\boldsymbol{w}_{uv_1}}+\frac{1}{\boldsymbol{w}_{uv_2}}}$$

added to the edge between v_1 and v_2.

Proof Without loss of generality suppose the two neighbors are 2 and 3 and the edge weights are $\mathbf{w}_{12}$, $\mathbf{w}_{13}$, then $\mathbf{U}^T$ is:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ \frac{\mathbf{w}_{12}}{\mathbf{w}_{12}+\mathbf{w}_{13}} & 1 & 0 & \dots & 0 \\ \frac{\mathbf{w}_{13}}{\mathbf{w}_{12}+\mathbf{w}_{13}} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

The only new off-diagonal entry affected by this is in row 2, column 3 and row 3, column 2. By symmetry it suffices to check the change to row 2 column 3. Here it is the value of row 1, column 3 ($\mathbf{w}_1 3$) times the coefficient in row 2, column 1, which gives $-\mathbf{w}_{13}\frac{\mathbf{w}_{12}}{\mathbf{w}_{12}+\mathbf{w}_{13}}$. ■

Pivoting out the first vertex leads to a lower-triangular matrix $\mathbf{U}^T$ where the only non-zero off diagonal entries are in the first column. Furthermore, these entries are edge weights divided by the total weighted degree of the vertex removed, and can be viewed as the normalized edge weights.

The inverse of $\mathbf{U}^T$ can also be described simply:

Lemma C.1.1 *Given a lower triangular matrix of the form:*

$$\boldsymbol{U}^T = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_2 & 1 & 0 & \dots & 0 \\ l_3 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ l_n & 0 & 0 & \dots & 1 \end{bmatrix}$$

Its inverse is:

$$\boldsymbol{U}^{-T} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -l_2 & 1 & 0 & \dots & 0 \\ -l_3 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ -l_n & 0 & 0 & \dots & 1 \end{bmatrix}$$

C.2 Errors Under Partial Cholesky Factorization

Errors in matrix norms propagate naturally under partial Cholesky factorizations.

Lemma C.2.1 *Let $\boldsymbol{B}$ be the matrix obtained by applying partial Cholesky factorization to $\boldsymbol{A}$:*

$$\boldsymbol{B} = \boldsymbol{U}^T \boldsymbol{A} \boldsymbol{U}^T$$

For any vector $\boldsymbol{x}$, we have:

$$\|\boldsymbol{x}\|_{\boldsymbol{B}} = \|\boldsymbol{U}\boldsymbol{x}\|_{\boldsymbol{A}}$$

Proof A direct substitution gives:

$$\begin{aligned}\|\mathbf{x}\|_{\mathbf{B}}^2 &= \mathbf{x}^T\mathbf{B}\mathbf{x}\\ &= \mathbf{x}^T\mathbf{U}^T\mathbf{A}\mathbf{U}\mathbf{x}\\ &= (\mathbf{x}\mathbf{U})^T\mathbf{A}(\mathbf{U}\mathbf{x})\\ &= \|\mathbf{x}\|_{\mathbf{A}}^2\end{aligned}$$

■

This allows us to prove general statements regarding the numerical stability of partial Cholesky factorizations. In numerical analysis, the stability of such instances are well-known [Hig02]. However, several technical issues need to be addressed for a complete proof. The main reason is that floating point errors can occur in all the steps:

1. Finding $\mathbf{L}_H$ from $\mathbf{U}^T\mathbf{L}_G\mathbf{U}$.
2. Solves involving $\mathbf{L}_H$ and the diagonal entry corresponding to u.
3. Multiplying vectors by $\mathbf{U}^T$ and $\mathbf{U}^{-1}$.

Since the **U** involved in our computations only have a constant number of off-diagonal entries, we can view operations involving **U** as a constant number of vector scaling/additions. This then allows us to view **U** as an exact matrix with no round-off errors for our analysis. We first show that the edge weights in the resulting graph Laplacian are close to the ones in the exact one. This step is simpler when we view the partial Cholesky factorization process implicitly.

Lemma C.2.2 *Given a graph G where the resistance of all edge weights are between $r_{\min}$ and $r_{\max}$, along with a vertex u of degree at most 2. Let $\bar{H}$ correspond to the exact partial Cholesky factorization with u removed. We can find in $O(1)$ time $\tilde{H}$ whose edge weights approximate those in $\bar{H}$ within multiplicative error of $\pm\frac{6r_{\max}}{r_{\min}}\epsilon_m$.*

Proof

In the case of u having degree 1, we simply remove the edge between u and its neighbor, so the bound on edge weights and total resistance holds in H. If u is degree 2, let the two edges incident to u have weights $\mathbf{w}_1$ and $\mathbf{w}_2$ respectively. Then the weight of the new edge in the absence of floating point error, $\bar{\mathbf{L}}_H$, is: $\frac{1}{\frac{1}{\mathbf{w}_1}+\frac{1}{\mathbf{w}_2}}$. If an additive error of ϵ_m is incurred at each step, the resulting value can be upper bounded by:

$$\frac{1}{\left(\frac{1}{\mathbf{w}_1}-\epsilon_m\right)+\left(\frac{1}{\mathbf{w}_2}-\epsilon_m\right)-\epsilon_m}+\epsilon_m=\frac{1}{\frac{1}{\mathbf{w}_1}+\frac{1}{\mathbf{w}_2}-3\epsilon_m}+\epsilon_m$$

Since $\frac{1}{\mathbf{w}_i} \geq r_{\min}$, each additive factor of ϵ_m incurs a multiplicative error of $\frac{\epsilon_m}{r_{\min}}$. So the result can be upper bounded by:

$$\left(1 - 3\frac{\epsilon_m}{r_{\min}}\right)^{-1} \frac{1}{\frac{1}{\mathbf{w}_1} + \frac{1}{\mathbf{w}_2}} + \epsilon_m$$

As $\frac{1}{\mathbf{w}_1}, \frac{1}{\mathbf{w}_2} \leq r_{\max}$, the exact result is at least $\frac{1}{2r_{\max}}$. So the additive factor of ϵ_m equals to a multiplicative factor of $2r_{\max}\epsilon_m$. Combining these two terms gives a total multiplicative distortion of $(1 - 3\frac{\epsilon_m}{r_{\min}})^{-1} + 2r_{\max}\epsilon_m \leq \frac{6r_{\max}}{r_{\min}}\epsilon_m$.

A lower bound on the resulting value can be obtained similarly. ■

This allows us to bound the effect of round-off errors across all the elimination steps performed during greedy elimination.

Lemma C.2.3 *Let G be a graph on n vertices with a spanning tree T_G such that the minimum resistance of an edge is at least $r_{\min}$ and the total resistance is at most r_s. Then if $\epsilon_m \leq \frac{r_{\min}}{20nr_{\max}}$, all intermediate graph/tree pairs during greedy elimination satisfies:*

1. *The weights of all off tree edges are unchanged.*
2. *The minimum resistance of an edge in T_H is $r_{\min}$ and the total is at most $2r_s$.*
3. *If a tree edge in T_H has weight $\boldsymbol{w}_e$ and its weight if exact arithmetic is used would be $\bar{\boldsymbol{w}}_e$, then $\frac{1}{2}\bar{\boldsymbol{w}}_e \leq \boldsymbol{w}_e \leq \bar{\boldsymbol{w}}_e$.*

Proof Note that aside from discarding edges, the other operation performed by greedy elimination is to sum the resistance of edges. Therefore, we can ensure that the resistance of the resulting edge does not decrease. Also, Lemma C.2.2 gives that the total resistance can only increase by a factor of $1 + \frac{6r_{\max}}{r_{\min}}\epsilon_m \leq 1 + \frac{1}{3n}$. As there are at most n tree edges, the cumulative effect of this is at most $(1 + \frac{1}{3n})^n \leq 2$.

Also, note that the resistance of a single edge in H in the absence of round-off errors is the sum of resistances of edges contracted to it. Therefore, the same analysis as total resistance is applicable here. Inverting $\bar{\mathbf{r}}_e \leq \tilde{\mathbf{r}}_e \leq 2\bar{\mathbf{r}}_e$ then gives the bound on edge weights. ■

This approximation in edges also means that the resulting graph $\mathbf{L}_H$ is spectrally very close to the exact one, $\bar{\mathbf{L}}_H$. Therefore we can show that a solver for $\mathbf{L}_H$ leads to a solver for $\mathbf{L}_G$ as well.

Lemma C.2.4 *Let G be a graph where all edge weights are in the range $[w_{\min}, w_{\max}]$ with $w_{\min} \leq 1 \leq w_{\max}$. If $\epsilon_m \leq \frac{w_{\min}}{10n^3 w_{\max}}$ and H is obtained by performing partial Cholesky factorization on a single vertex u in G. Given any $(\epsilon, \epsilon_1, \mathcal{T})$-solver for $\boldsymbol{L}_H$, $\text{SOLVE}_{\boldsymbol{L}_H}$ with $\frac{1}{n} \leq \epsilon, \epsilon_1 \leq \frac{1}{2}$, we can obtain a $\big((1 + \frac{1}{3n})\epsilon, (1 + \frac{1}{3n})\epsilon_1, \mathcal{T} + O(1)\big)$-solver for $\boldsymbol{L}_G$ such that for any vector $\boldsymbol{b}$, $\text{SOLVE}_{\boldsymbol{L}_G}(\boldsymbol{b})$ makes one call to $\text{SOLVE}_{\boldsymbol{L}_H}(\boldsymbol{b}')$ with a vector $\boldsymbol{b}'$ such that:*

$$\|\boldsymbol{b}'\|_{\boldsymbol{L}_H^\dagger} \leq \|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger} + \frac{n^2}{w_{\min}}\epsilon_m$$

and the magnitude of entries in all intermediate vectors do not exceed $2n^3(w_{\min}^{-1} + w_{\max}) \|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger}$.

Proof

Let $\mathbf{C}_G$ be the exact partial Cholesky factorization:

$$\begin{aligned}\mathbf{C}_G &= \mathbf{U}^T\mathbf{L}_G\mathbf{U}\\ &= \left[\begin{array}{cc}\mathbf{d}_u & 0\\ 0 & \mathbf{L}_{\bar{H}}\end{array}\right]\end{aligned}$$

Where $\bar{H}$ is the graph that would be produced by partial Cholesky factorization in the absence of round-off errors. Then $\mathbf{L}_G^\dagger = \mathbf{U}\mathbf{C}_G^\dagger\mathbf{U}^T$. We first show that $\textsc{Solve}_{\mathbf{L}_H}$ readily leads to a solver for $\mathbf{C}_G$. The exact inverse of $\mathbf{C}_G$ is:

$$\mathbf{C}_G^\dagger = \left[\begin{array}{cc}\mathbf{d}_u^{-1} & 0\\ 0 & \mathbf{L}_H^\dagger\end{array}\right]$$

Note that $\mathbf{C}_G$-norm is the sum of the entry-wise norm in $\mathbf{d}_u$ of the entry corresponding to u and the $\mathbf{L}_H$-norm of the rest of the entries. Since all weights are at most $w_{\max}$, $\mathbf{d}_u \leq 2w_{\max}$. This means that an absolute error of ϵ_m is at most $2w_{\max}\epsilon_m$ in $\mathbf{C}_G$ norm .

Also, Lemma C.2.2 gives that all edge weights in H are within a multiplicative factor of $\frac{1}{n^3}$ from those in $\bar{H}$. Therefore $(1 - \frac{1}{n^3})\mathbf{L}_{\bar{H}} \preceq \mathbf{L}_H \preceq (1 + \frac{1}{n^3})\mathbf{L}_{\bar{H}}$ If $\mathbf{Z}_{\mathbf{L}_H}$ is the matrix related to $\textsc{Solve}_{\mathbf{L}_H}$, then $(1 - \frac{1}{n^3})(1 - \epsilon)\mathbf{L}_{\bar{H}}^\dagger \preceq \mathbf{Z}_{\mathbf{L}_H} \preceq (1 + \frac{1}{n^3})(1 + \epsilon)\mathbf{L}_{\bar{H}}$. Also, the deviation term may increase by a factor of $1 + \frac{1}{n^3}$ due to the spectral difference between $\mathbf{L}_H$ and $\mathbf{L}_{\bar{H}}$. Therefore $\textsc{Solve}_{\mathbf{L}_H}$ is a $((1 + \frac{1}{n^3})\epsilon, (1 + \frac{1}{n^3})\epsilon_1, \mathcal{T})$ solver for $\mathbf{L}_{\bar{H}}$.

Combining $\textsc{Solve}_{\mathbf{L}_H}$ with the scalar inversion of $\mathbf{d}_u$ then gives a solver for $\mathbf{C}_G$, $\textsc{Solve}_{\mathbf{C}_G}$. Since $\mathbf{C}_G$ is block-diagonal, both the bound of approximate inverse operations and errors in $\mathbf{C}_G$-norm can be measured separately. This gives a total deviation of up to $(1 + \frac{1}{n^3})\epsilon_1 + 2w_{\max}\epsilon_m \leq (1 + \frac{2}{n^3})\epsilon_1$ in $\mathbf{C}_G$-norm, while the approximation factor only comes from $\mathbf{L}_H$. Therefore $\textsc{Solve}_{\mathbf{C}_G}$ is a $((1 + \frac{1}{n^3})\epsilon, (1 + \frac{2}{n^3})\epsilon_1, \mathcal{T} + O(1))$ solver for $\mathbf{C}_G$. Also, the only operation performed by $\textsc{Solve}_{\mathbf{C}_G}$ is to call $\textsc{Solve}_{\mathbf{L}_H}$ on the part of the vector without u. The block-diagonal structure of $\mathbf{d}_u$ and $\mathbf{L}_H$ again gives that the $\mathbf{L}_H$ norm of this vector is at most the $\mathbf{C}_G$-norm of the input vector.

It remains to turn $\textsc{Solve}_{\mathbf{C}_G}$ into a solver for $\mathbf{L}_G$. Since $\mathbf{L}_G^\dagger = \mathbf{U}\mathbf{C}_G^\dagger\mathbf{U}^T$, it suffices to bound the errors of multiplying by $\mathbf{U}^T$ and $\mathbf{U}$.

Let $\mathbf{Z}_{\mathbf{C}_G}$ be the linear operator corresponding to $\textsc{Solve}_{\mathbf{C}_G}$. Then combining the guarantees of the algorithm with Lemma C.0.1 gives:

$$\left(1 - \left(1 + \frac{1}{3n}\right)\epsilon\right)\mathbf{L}_G^\dagger \preceq \mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{U}^T \preceq \left(1 + \left(1 + \frac{1}{3n}\right)\epsilon\right)\mathbf{L}_G^\dagger$$

Therefore $\mathbf{Z}_{\mathbf{L}_G} = \mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{U}^T$ suffices as the matrix related to $\textsc{Solve}_{\mathbf{L}_G}$. We now need to bound the deviations caused by $\textsc{Solve}_{\mathbf{C}_G}$, as well as multiplications by $\mathbf{U}^T$ and $\mathbf{U}$. Let these three error vectors be $\mathbf{err}_1$, $\mathbf{err}_2$, and $\mathbf{err}_3$. Then the vector returned by $\textsc{Solve}_{\mathbf{L}_G}$ is :

$$\begin{aligned}
&\mathbf{U}\left(\mathbf{Z}_{\mathbf{C}_G}\left(\mathbf{U}^T\mathbf{b} + \mathbf{err}_1\right) + \mathbf{err}_2\right) + \mathbf{err}_3 \\
&= \mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{U}^T\mathbf{b} + \mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{err}_1 + \mathbf{U}\mathbf{err}_2 + \mathbf{err}_3 \\
&= \mathbf{Z}_{\mathbf{L}_G}\mathbf{b} + \mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{err}_1 + \mathbf{U}\mathbf{err}_2 + \mathbf{err}_3
\end{aligned}$$

The first term is the error-free result, therefore it suffices to bound the $\mathbf{L}_G$ norm of the three later terms separately. Our assumptions about round-off errors gives $\left\|\mathbf{err}_1\right\|_2, \left\|\mathbf{err}_3\right\|_2 \leq \epsilon_m$, and the guarantees of $\textsc{Solve}_{\mathbf{C}_G}$ shown above gives $\left\|\mathbf{err}_2\right\|_{\mathbf{C}_G} \leq (1 + \frac{2}{n^3})\epsilon_1$. For $\mathbf{err}_2$, we have:

$$\begin{aligned}
\left\|\mathbf{U}\mathbf{err}_2\right\|_{\mathbf{L}_G} &= \sqrt{\mathbf{err}_2^T\mathbf{U}^T\mathbf{L}_G\mathbf{U}\mathbf{err}_2} \\
&= \sqrt{\mathbf{err}_2^T\mathbf{C}_G\mathbf{err}_2} \\
&= \left\|\mathbf{err}_2\right\|_{\mathbf{C}_G} \\
&\leq \left(1 + \frac{2}{n^3}\right)\epsilon_1
\end{aligned}$$

For $\mathbf{err}_3$, since the maximum edge weight is at most $w_{\max}$, the Frobenius norm, and in turn 2-norm of $\mathbf{L}_G$ is at most $w_{\max} \leq n^2$. So we have $\left\|\mathbf{err}_3\right\|_{\mathbf{L}_G} \leq n^2 w_{\max}\epsilon_m \leq \frac{1}{20n}\epsilon_1$.

The only remaining term is $\left\|\mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{err}_1\right\|_{\mathbf{L}_G}$ The term can be rewritten as:

$$\mathbf{U}\mathbf{Z}_{\mathbf{C}_G}\mathbf{U}^T\mathbf{U}^{-T}\mathbf{err}_1 = \mathbf{Z}_{\mathbf{L}_G}\mathbf{U}^{-T}\mathbf{err}_1$$

The $\mathbf{L}_G$ norm of it is then:

$$\left\|\mathbf{Z}_{\mathbf{L}_G}\mathbf{U}^{-T}\mathbf{err}_1\right\|_{\mathbf{L}_G}^2 = (\mathbf{U}^{-T}\mathbf{err})^T\mathbf{Z}_{\mathbf{L}_G}\mathbf{L}_G\mathbf{Z}_{\mathbf{L}_G}(\mathbf{U}^{-T}\mathbf{err})$$

A weaker version of the guarantees of $\mathbf{Z}_{\mathbf{L}_G}$ then gives:

$$\begin{aligned}
\left\|\mathbf{Z}_{\mathbf{L}_G}\mathbf{U}^{-T}\mathbf{err}_1\right\|_{\mathbf{L}_G}^2 &\leq 2\left\|\mathbf{U}^{-T}\mathbf{err}\right\|_{\mathbf{Z}_{\mathbf{L}_G}}^2 \\
&\leq 4\left\|\mathbf{U}^{-T}\mathbf{err}\right\|_{\mathbf{L}_G^\dagger}^2
\end{aligned}$$

Since G is connected by edges weighting at least $w_{\min}$, the minimum non-zero eigenvalue of $\mathbf{L}_G$ is at least $\frac{1}{n^2 U}$. So $\mathbf{L}_G^\dagger \preceq \frac{2n^2}{w_{\min}}\mathbf{I}$. Also, since all entries in $\mathbf{U}$ are at most 1, the Frobenius norm of $\mathbf{U}^{-T}\mathbf{U}^{-1}$ can be bounded by n. Combining these gives an upper bound of $\sqrt{\frac{4n^3}{w_{\min}}}\epsilon_m \leq \frac{1}{20n}\epsilon_1$. Therefore the total deviation can be bounded by $(1 + \frac{1}{3n})\epsilon_1$.

The vector passed to $\text{SOLVE}_{\mathbf{C}_G}$ is vector is $\mathbf{U}^T\mathbf{b} + \mathbf{err}_1$. Decomposing this using the triangle inequality gives:

$$\begin{aligned}
\left\|\mathbf{U}^T\mathbf{b} + \mathbf{err}_1\right\|_{\mathbf{C}_G^\dagger} &\leq \left\|\mathbf{U}^T\mathbf{b}\right\|_{\mathbf{C}_G^\dagger} + \left\|\mathbf{err}_1\right\|_{\mathbf{C}^\dagger}\\
&\leq \left\|\mathbf{U}^T\mathbf{b}\right\|_{\mathbf{C}_G^\dagger} + \frac{n^2}{w_{\min}}\epsilon_m\\
&\leq \left\|\mathbf{b}\right\|_{\mathbf{L}_G^\dagger} + \frac{n^2}{w_{\min}}\epsilon_m
\end{aligned}$$

Where the second last inequality follows bounding the minimum non-zero eigenvalue of $\mathbf{C}_G$ by $\frac{1}{n^2 w_{\min}}$. This also bounds the maximum entry of this vector.

These bounds in $\mathbf{L}_G$ and $\mathbf{L}_G^\dagger$ norms can be converted to bounds on magnitude of entries using the ℓ_2 norm. The bounds on edge weights gives that all non-zero eigenvalues of $\mathbf{L}_G$ are between $\frac{w_{\min}}{n^2}$ and $n^2 w_{\max}$. This immediately allows us to bound $\|\mathbf{b}\|_2$ by $n^2 w_{\max}\|\mathbf{b}\|_{\mathbf{L}_G^\dagger}$. Also, our guarantees implies the output, $\mathbf{x}$ satisfies $\|\mathbf{x}\|_{\mathbf{L}_G} \leq 2\|\mathbf{b}\|_{\mathbf{L}_G^\dagger}$. So the ℓ_2-norm of this vector can be bounded by $\frac{n^2}{w_{\min}}\|\mathbf{b}\|_{\mathbf{L}_G^\dagger}$ as well. This combined with bounds on $\|\mathbf{err}_3\|$ and the Frobenius norm of $\mathbf{U}^{-1}$ then gives bounds on the magnitudes of $\mathbf{U}(\mathbf{Z}_{\mathbf{C}_G}(\mathbf{U}^T\mathbf{b} + \mathbf{err}_1) + \mathbf{err}_2)$ and $\mathbf{Z}_{\mathbf{C}_G}(\mathbf{U}^T\mathbf{b} + \mathbf{err}_1)$ as well. ■

Applying this Lemma repeatedly gives the following guarantees of greedy elimination.

Lemma C.2.5 (Greedy Elimination) *Let G be a graph on n vertices with a spanning tree T_G such that the minimum resistance of a tree edge is at least $r_{\min} \leq 1$ and the total resistance is at most $r_s \geq 1$. If $\epsilon_m \leq \frac{r_{\min}}{3n^3 r_s}$, then running* GREEDYELIMINATION *on G and T_G in $O(m)$ time to produces a graph H with a spanning tree T_H such that:*

1. *The weights of all off tree edges are unchanged.*
2. *The resistance of an edge in T_H is at least $r_{\min}$ and the total is at most $2r_s$.*
3. *If a tree edge in T_H has weight $\tilde{\boldsymbol{w}}_e$ and its weight if exact arithmetic is used would be $\bar{\boldsymbol{w}}_e$, then $\frac{1}{2}\bar{\boldsymbol{w}}_e \leq \tilde{\boldsymbol{w}}_e \leq \bar{\boldsymbol{w}}_e$.*
4. *Given a routine $\text{SOLVE}_{\boldsymbol{L}_H}$ that is a $(\epsilon, \epsilon_1, \mathcal{T})$-solver for $\boldsymbol{L}_H$, we can obtain a routine $\text{SOLVE}_{\boldsymbol{L}_G}$ that is a $(2\epsilon, 2\epsilon_1, \mathcal{T} + O(m))$-solver for $\boldsymbol{L}_G$, Furthermore, for any vector $\boldsymbol{b}$, $\text{SOLVE}_{\boldsymbol{L}_G}$ makes one call to $\text{SOLVE}_{\boldsymbol{L}_H}$ with a vector $\boldsymbol{b}'$ such that $\|\boldsymbol{b}'\|_{\boldsymbol{L}_H^\dagger} \leq \|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger} + 1$ and all intermediate values are bounded by $O(n^3(r_s + r_{\min}^{-1})(\|\boldsymbol{b}\|_{\boldsymbol{L}_G^\dagger} + 1))$.*

Proof Let the sequence of graphs generated by the pivoting be $G_0 = G, G_1 \ldots G_k = H$ for some $k < n$. Lemma C.2.3 gives that for all G_i the minimum resistance is at least $r_{\min}$ and the total is at most $2r_s$. Therefore the minimum weight in any G_i is at least $\frac{1}{2r_s}$ and the maximum is at most $\frac{1}{r_{\min}}$. Therefore the choice of ϵ_m meets the requirement for Lemma C.2.4. Since $(1 + \frac{1}{3n})^n \leq 2$, a simple induction gives that $\textsc{Solve}_{\mathbf{L}_H}$ leads to $\textsc{Solve}_{\mathbf{L}_G}$ that is a $(2\epsilon, 2\epsilon_1, \mathcal{T} + O(m))$ solver for $\mathbf{L}_G$.

Another induction down the chain gives that $\|\mathbf{b}'\|_{\mathbf{L}^{\dagger}_{G_i}} \leq \|\mathbf{b}\|_{\mathbf{L}^{\dagger}_{G}} + \frac{n^3}{w_{\min}}\epsilon_m$. Our assumption on ϵ_m gives that this is bounded by $\|\mathbf{b}\|_{\mathbf{L}^{\dagger}_{G}} + 1$. Combining this with the bound of edge weights gives the bound on the magnitudes of all intermediate values. ■

Appendix D

Iterative Methods

In this chapter we prove guarantees of iterative methods for completeness. Our precision analysis is in the fixed-point model, and the necessary assumptions are stated at the start of Section 2.6.

D.1 Richardson Iteration

We now show Richardson iteration, which reduces the error from a constant to ϵ.

Lemma 1.6.8 *If $\boldsymbol{A}$ is a positive semi-definite matrix and* $\textsc{Solve}_{\boldsymbol{A}}$ *is a routine such that for any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$,* $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ *returns a vector $\boldsymbol{x}$ such that:*

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \frac{1}{2}\|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

Then for any ϵ, there is an algorithm $\textsc{Solve}'_{\boldsymbol{A}}$ *such that any vector $\boldsymbol{b} = \boldsymbol{A}\bar{\boldsymbol{x}}$,* $\textsc{Solve}'_{\boldsymbol{A}}(\boldsymbol{b})$ *returns a vector $\boldsymbol{x}$ such that:*

$$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_{\boldsymbol{A}} \leq \epsilon\|\bar{\boldsymbol{x}}\|_{\boldsymbol{A}}$$

by making $O(\log(1/\epsilon))$ calls to $\textsc{Solve}'_{\boldsymbol{A}}$ *and matrix-vector multiplications to $\boldsymbol{A}$. Furthermore,* $\textsc{Solve}'_{\boldsymbol{A}}$ *is stable under round-off errors.*

The assumptions that we make about round-off errors are presented in detail in Section 2.6. Specifically, any matrix/vector computations leads to an answer that differs from the true answer by a vector whose magnitude is at most ϵ_m. Furthermore, we assume that $\mathbf{b}$ is normalized so that $\|\mathbf{b}\|_2 = 1$, and at each step we project our solution vector onto the column space of $\mathbf{A}$.

Let λ be a bound such that all non-zero eigenvalues of $\mathbf{A}$ are between $\frac{1}{\lambda}$ and λ, We will show convergence when $\epsilon_m \leq \frac{\epsilon}{60\lambda^4}$.

Proof Consider the following recurrence:

$$\mathbf{x}_0 = \mathbf{0}$$
$$\mathbf{x}_{i+1} = \mathbf{x}_i - \frac{1}{4}\textsc{Solve}_{\mathbf{A}}\left(\mathbf{A}\mathbf{x}_i - \mathbf{b}\right)$$

Implementing this recurrence as given leads to two sources of additional round-off errors: computing $\mathbf{A}\mathbf{x}_i - \mathbf{b}$, and subtracting the a scaled version of the output of $\textsc{Solve}_{\mathbf{A}}$ from $\mathbf{x}_i$. If we denote them as $\mathbf{err}_i^{(1)}$ and $\mathbf{err}_i^{(2)}$, the result of our algorithm is then produced by the following recurrence:

$$\tilde{\mathbf{x}}_0 = \mathbf{0}$$
$$\tilde{\mathbf{x}}_{i+1} = \tilde{\mathbf{x}}_i - \frac{1}{4}\textsc{Solve}_{\mathbf{A}}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right) + \mathbf{err}_i^{(2)}$$

Also, at each iteration, we will check the value of $\left\|\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\right\|_2 \leq \frac{\epsilon}{\lambda^2}$ and terminate if it is too small. This is because the spectrum bounds on $\mathbf{A}$ gives that if $\left\|\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\right\|_2 \leq \frac{\epsilon}{\lambda^2}$, $\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \leq \frac{\epsilon}{\lambda}$. Since $\left\|\mathbf{b}\right\|_2 = 1$, $\left\|\bar{\mathbf{x}}\right\|_{\mathbf{A}} = \left\|\mathbf{b}\right\|_{\mathbf{A}^{\dagger}} \geq \frac{1}{\lambda}$ and the answer meets the requirement. Therefore we may work under the assumption that $\left\|\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\right\|_2 \geq \frac{\epsilon}{\lambda^2}$, which once again using spectral bounds on $\mathbf{A}$ translates to $\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \geq \frac{\epsilon}{\lambda^3}$.

We will show that the error reduces geometrically over the iterations.

For the starting point of $\tilde{\mathbf{x}}_0 = \mathbf{0}$, we have:

$$\begin{aligned}\left\|\tilde{\mathbf{x}}_0 - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 &= \left\|\mathbf{0} - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \\ &= \left\|\bar{\mathbf{x}}\right\|_{\mathbf{A}}^2\end{aligned}$$

We will then bound the change of the error from $\tilde{\mathbf{x}}_i$ to $\tilde{\mathbf{x}}_{i+1}$. To simplify notation, we use $\mathbf{r}_i$ to denote $\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}$, and Δ_i to denote the difference, $\tilde{\mathbf{x}}_{i+1} - \tilde{\mathbf{x}}_i$.

$$\begin{aligned}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 - \left\|\tilde{\mathbf{x}}_{i+1} - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 &= \left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 - \left\|\Delta_i + \tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 && \text{(D.1)} \\ &= -2\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)^T \mathbf{A}\Delta_i - \Delta_i^T\mathbf{A}\Delta_i && \text{(D.2)} \\ &= -2(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b})^T\Delta_i - \Delta_i^T\mathbf{A}\Delta_i && \text{(D.3)} \\ &= -2\mathbf{r}_i^T\Delta_i - \Delta_i^T\mathbf{A}\Delta_i && \text{(D.4)}\end{aligned}$$

We will upper bound this quantity by showing that $-4\Delta_i$ is close to $\bar{\mathbf{x}} - \tilde{\mathbf{x}}_i$. The guarantees of

$\textsc{Solve}_{\mathbf{A}}$ can be expressed as:

$$\left\|\textsc{Solve}_{\mathbf{A}}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right) - \mathbf{A}^{\dagger}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right)\right\|_{\mathbf{A}} \leq \frac{1}{2}\left\|\mathbf{A}^{\dagger}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right)\right\|_{\mathbf{A}}$$
$$\left\|\textsc{Solve}_{\mathbf{A}}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right) - (\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}) - \mathbf{A}^{\dagger}\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}} \leq \frac{1}{2}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}} + \mathbf{A}^{\dagger}\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}}$$

Moving the $\mathbf{A}^{\dagger}\mathbf{err}_i^{(1)}$ term out of both sides using the triangle inequality, and incorporating in $\mathbf{err}_i^{(2)}$ then gives:

$$\left\|\textsc{Solve}_{\mathbf{A}}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right) - (\tilde{\mathbf{x}}_i - \bar{\mathbf{x}})\right\|_{\mathbf{A}} \leq \frac{1}{2}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} + \frac{3}{2}\left\|\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}^{\dagger}}$$
$$\left\|-4\Delta_i - (\tilde{\mathbf{x}}_i - \bar{\mathbf{x}})\right\|_{\mathbf{A}} \leq \frac{1}{2}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} + \frac{3}{2}\left\|\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}^{\dagger}} + 4\left\|\mathbf{err}_i^{(2)}\right\|_{\mathbf{A}}$$

Our bounds on round-off errors gives $\left\|\mathbf{err}_i^{(1)}\right\|_2, \left\|\mathbf{err}_i^{(2)}\right\|_2 \leq \epsilon_m$, Furthermore, the assumption that all non-zero eigenavalues of $\mathbf{A}$ are between $\frac{1}{\lambda}$ and λ gives $\mathbf{A}, \mathbf{A}^{\dagger} \preceq \lambda \cdot \mathbf{I}$. Therefore the norms of the two (additive) error terms can be bounded by $6\lambda\epsilon_m$. By the assumption that $\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \geq \frac{\epsilon}{\lambda^3}$ and $\epsilon_m \leq \frac{\epsilon}{60\lambda^4}$, this translates to a multiplicative error of $\frac{1}{10}$, giving:

$$\left\|-4\Delta_i - (\tilde{\mathbf{x}}_i - \bar{\mathbf{x}})\right\|_{\mathbf{A}} \leq \frac{3}{5}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}$$

Squaring both sides of it and and substituting in $\mathbf{r}_i = \mathbf{A}\mathbf{x}_i - \mathbf{b}$ gives:

$$\left\|-4\Delta_i - (\tilde{\mathbf{x}}_i - \bar{\mathbf{x}})\right\|_{\mathbf{A}} \leq \frac{3}{5}\left\|\mathbf{x}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \tag{D.5}$$
$$16\Delta_i^T\mathbf{A}\Delta_i + 8\mathbf{r}_i^T\Delta_i + \left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \leq \frac{9}{25}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \tag{D.6}$$
$$2\mathbf{r}_i^T\Delta_i + 4\Delta_i^T\mathbf{A}\Delta_i \leq -\frac{4}{25}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \tag{D.7}$$

Since $\mathbf{A}$ is positive semi-definite, $\Delta_i^T\mathbf{A}\Delta_i \geq 4\Delta_i^T\mathbf{A}\Delta_i$. Thus we have:

$$\left\|\mathbf{x}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 - \left\|\mathbf{x}_{i+1} - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \geq \frac{4}{25}\left\|\mathbf{x}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}^2 \tag{D.8}$$

Rearranging and taking square roots of both sides gives:

$$\left\|\mathbf{x}_{i+1} - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \leq \sqrt{1 - \frac{4}{25}}\left\|\mathbf{x}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \tag{D.9}$$

Which means that unless the algorithm is terminated early, the error is decreasing geometrically at a constant rate. Therefore in $O\left(\log\left(1/\epsilon\right)\right)$ iterations it becomes less than ϵ.

It remains to bound the magnitude of intermediate vectors. Note that the assumption of $\|\mathbf{b}\|_2 = 1$ and assumption of on the eigenvalues of $\mathbf{A}$ gives $\|\bar{\mathbf{x}}\|_{\mathbf{A}} \leq \sqrt{\lambda}$. The convergence condition shown above gives $\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\|_{\mathbf{A}} \leq \|\bar{\mathbf{x}}\|_{\mathbf{A}}$. The triangle inequality then gives $\|\tilde{\mathbf{x}}_i\|_{\mathbf{A}} \leq 2\,\|\bar{\mathbf{x}}\|_{\mathbf{A}} \leq 2\sqrt{\lambda}$. By the assumption that $\tilde{\mathbf{x}}_i$ is projected into the column space of $\mathbf{A}$ at each step and the minimum non-zero eigenvalue of $\mathbf{A}$ being is least $\frac{1}{\lambda}$, $\|\tilde{\mathbf{x}}_i\|_2 \preceq \sqrt{\lambda}\,\|\tilde{\mathbf{x}}_i\|_{\mathbf{A}}$. Therefore the maximum magnitude of the entries in $\tilde{\mathbf{x}}_i$ can be bounded by $2\sqrt{\lambda}$.

For the intermediate steps, we have:

$$\begin{aligned}
\|\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\|_\infty &\leq \|\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\|_2 \\
&= \|\mathbf{A}(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}})\|_2 \\
&\leq \sqrt{\lambda}\,\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\|_{\mathbf{A}} \\
&\leq \lambda
\end{aligned}$$

Since the maximum magnitude of an entry in $\mathbf{b}$ is at most 1, the magnitude of entries in $\mathbf{A}\tilde{\mathbf{x}}_i$ can be bounded by $\lambda + 1 \leq 2\lambda$. Also, the magnitude of the entries in intermediate vectors passed to $\textsc{Solve}_{\mathbf{A}}$ can be bounded by $\left\|\mathbf{A}(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}) + \mathbf{err}_2^{(1)}\right\|_2 \leq \lambda + \epsilon_m \leq 2\lambda$.

■

D.2 Chebyshev Iteration

As the name suggests, Chebyshev iteration is closely related with Chebyshev Polynomials. There are two kinds of Chebyshev Polynomials, both defined by recurrences. Chebyshev polynomials of the first kind, $T_n(x)$ can be defined as:

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_{i+1}(x) &= 2xT_i(x) - T_{i-1}(x)
\end{aligned}$$

Alternatively, $T_i(x)$ can be characterized using the following closed from:

$$T_i(x) = \frac{\left(x - \sqrt{x^2-1}\right)^i + \left(x + \sqrt{x^2-1}\right)^i}{2}$$

The following facts about Chebyshev polynomials of the first kind will be used to bound convergence.

Fact D.2.1 *If $x = \cos(\theta)$, then*

$$T_n(x) = \cos(n\theta)$$

This allows us to show that if $|x| \leq 1$, $|T_n(x)| \leq 1$. For proving convergence, we also need the opposite statement for lower bounding $T_n(x)$ when x is large.

Fact D.2.2 *If $x = 1 + \frac{1}{\kappa}$, then:*

$$\begin{aligned} T_i(x) &\geq \frac{1}{2}\left(x + \sqrt{x^2 - 1}\right)^i \\ &\geq \frac{1}{2}\left(1 + \sqrt{1 + \frac{2}{\kappa} - 1}\right)^i \\ &\geq \frac{1}{2}\left(1 + \frac{1}{\sqrt{\kappa}}\right)^i \end{aligned}$$

We can also show that these terms are steadily inceasing:

Fact D.2.3 *If $i \leq j$ and $x \geq 1$, then $T_i(x) \geq 2T_j(x)$.*

Proof $x \geq 1$ implies $0 \leq x - \sqrt{x^2 - 1} \leq 1$ and $1 \leq x + \sqrt{x^2 - 1}$. Therefore:

$$\begin{aligned} T_{i+1}(x) &\geq \frac{\left(x + \sqrt{x^2 - 1}\right)^{i+1}}{2} \\ &\geq \frac{\left(x + \sqrt{x^2 - 1}\right)^{i}}{2} \\ &\geq T_i(x) - \frac{1}{2} \end{aligned}$$

Fact D.2.2 also gives $T_{i+1}(x) \geq \frac{1}{2}$. Combining these gives $T_i(\delta) \geq 2T_j(\delta)$. ■

For bounding the extraneous error, we also need Chebyshev polynomials of the second kind. These polynomials, given as $U_n(x)$, follow the same recurrence, but has a different base case:

$$\begin{aligned} U_{-1}(x) &= 0 \\ U_0(x) &= 1 \\ U_{i+1}(x) &= 2xT_i(x) - T_{i-1}(x) \end{aligned}$$

These polynomials are related to Chebyshev polynomials of the first kind by the following identity:

Fact D.2.4

$$U_i(x) = \begin{cases} 2\sum_{j \text{ odd}} T_j(x) & \text{If } i \text{ is odd} \\ 2\sum_{j \text{ even}} T_j(x) & \text{If } i \text{ is even} \end{cases}$$

Throughout our presentation, we will apply both of these polynomials to matrices. Recall from Lemma 1.6.1 the resulting linear operator equals to the one obtained by applying the polynomial on each of the eigenvalues.

We first show the convergence of Chebyshev iteration under exact arithmetic.

Lemma 2.1.1 (Preconditioned Chebyshev Iteration) *There exists an algorithm* PRECONCHEBY *such that for any symmetric positive semi-definite matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and κ where*

$$\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa \boldsymbol{A}$$

any error tolerance $0 < \epsilon \leq 1/2$, PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *in the exact arithmetic model is a symmetric linear operator on $\boldsymbol{b}$ such that:*

1. *If $\boldsymbol{Z}$ is the matrix realizing this operator, then $(1-\epsilon)\boldsymbol{A}^{\dagger} \preceq \boldsymbol{Z} \preceq (1+\epsilon)\boldsymbol{A}^{\dagger}$*
2. *For any vector $\boldsymbol{b}$,* PRECONCHEBY$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{b}, \epsilon)$ *takes $N = O\left(\sqrt{\kappa}\log\left(1/\epsilon\right)\right)$ iterations, each consisting of a matrix-vector multiplication by $\boldsymbol{A}$, a solve involving $\boldsymbol{B}$, and a constant number of vector operations.*

Preconditioned Chebyshev iteration is given by the following recurrence with δ set to $1 + \frac{1}{\kappa}$:

Base case:

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{0} \\ \mathbf{x}_1 &= \mathbf{B}^{\dagger}\mathbf{b} \end{aligned}$$

Iteration:

$$\begin{aligned} \mathbf{y}_{i+1} &= \mathbf{B}^{\dagger}\left(\mathbf{A}\mathbf{x}_i - \mathbf{b}\right) \\ \mathbf{x}_{i+1} &= \frac{2\delta T_i\left(\delta\right)}{T_{i+1}\left(\delta\right)}\left(\mathbf{x}_i - \mathbf{y}_{i+1}\right) - \frac{T_{i-1}\left(\delta\right)}{T_{i+1}\left(\delta\right)}\mathbf{x}_{i-1} \end{aligned}$$

We first show that the residue, $\bar{\mathbf{x}} - \mathbf{x}_i$ can be characterized in terms of Chebyshev polynomials.

Lemma D.2.5 *Let $\bar{x} = \boldsymbol{A}^{\dagger}\boldsymbol{b}$. The result produced after i iterations of Chebyshev iteration under exact arithmetic, $\boldsymbol{x}_i$ satisfies:*

$$T_i\left(\delta\right)\left(\bar{\boldsymbol{x}} - \boldsymbol{x}_i\right) = \boldsymbol{A}^{\dagger 1/2} T_i\left(\delta\left(\boldsymbol{I} - \boldsymbol{A}^{1/2}\boldsymbol{B}^{\dagger}\boldsymbol{A}^{1/2}\right)\right)\boldsymbol{A}^{1/2}\bar{\boldsymbol{x}}$$

Proof The proof is by induction. The base case can be checked as follows:

$$\begin{aligned} \bar{\mathbf{x}} - \mathbf{x}_0 &= \bar{\mathbf{x}} \\ &= \mathbf{A}^{\dagger 1/2}\mathbf{A}^{1/2}\bar{\mathbf{x}} \\ \bar{\mathbf{x}} - \mathbf{x}_1 &= \bar{\mathbf{x}} - \mathbf{B}^{\dagger}\mathbf{b} \\ &= \bar{\mathbf{x}} - \mathbf{B}^{\dagger}\mathbf{A}\bar{\mathbf{x}} \\ &= \mathbf{A}^{\dagger 1/2}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} \end{aligned}$$

For the inductive case, the recurrence can be rearranged to give:

$$T_{i+1}(\delta)\,\mathbf{x}_{i+1} = 2\delta T_i(\delta)\left(\mathbf{x}_i - \mathbf{y}_{i+1}\right) - T_{i-1}(\delta)(\delta)\,\mathbf{x}_{i-1}$$

Recall from the definition of Chebyshev polynomials of the first kind that:

$$T_{i+1}(\delta) = 2(\delta)\,T_i(\delta) - T_{i-1}(\delta)$$

So we can subtract both sides from $T_{i+1}(\delta)\,\bar{\mathbf{x}}$ to get:

$$T_{i+1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i+1}\right) = 2\delta T_i(\delta)\left(\bar{\mathbf{x}}_i - \mathbf{x}_i + \mathbf{y}_i\right) - T_{i-1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i-1}\right)$$

The change, $\mathbf{y}_i$, can be viewed as computed by multiplying the difference at iteration i by $\mathbf{B}^{-1}\mathbf{A}$:

$$\begin{aligned} y_{i+1} &= \mathbf{B}^{\dagger}\left(\mathbf{A}\mathbf{x}_i - \mathbf{b}\right) \\ &= \mathbf{B}^{\dagger}\left(\mathbf{A}\mathbf{x}_i - \mathbf{A}\bar{\mathbf{x}}\right) \\ &= \mathbf{B}^{\dagger}\mathbf{A}\left(\mathbf{x}_i - \bar{\mathbf{x}}\right) \end{aligned}$$

Substituting this in gives:

$$\begin{aligned} T_{i+1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i+1}\right) &= 2\delta T_i(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_i - \mathbf{B}^{\dagger}\mathbf{A}\left(\bar{\mathbf{x}} - \mathbf{x}_i\right)\right) - T_{i-1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i-1}\right) \\ &= 2\delta T_i(\delta)\left(\mathbf{I} - \mathbf{B}^{\dagger}\mathbf{A}\right)\left(\bar{\mathbf{x}} - \mathbf{x}_i\right) - T_{i-1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i-1}\right) \\ &= 2\delta\left(\mathbf{I} - \mathbf{B}^{\dagger}\mathbf{A}\right)T_i(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_i\right) - T_{i-1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i-1}\right) \end{aligned}$$

Substituting in the inductive hypothesis then gives:

$$\begin{aligned} &T_{i+1}(\delta)\left(\bar{\mathbf{x}} - \mathbf{x}_{i+1}\right) \\ &= 2\delta\left(\mathbf{I} - \mathbf{B}^{\dagger}\mathbf{A}\right)\mathbf{A}^{\dagger 1/2}T_i\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} - \mathbf{A}^{\dagger 1/2}T_i\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} \\ &= 2\delta\mathbf{A}^{\dagger 1/2}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)T_i\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} - \mathbf{A}^{\dagger 1/2}T_i\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} \\ &= \mathbf{A}^{\dagger 1/2}T_{i+1}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\bar{\mathbf{x}} \end{aligned}$$

■

Therefore to bound the error, it suffices to lower bound $T_i(\delta)$ as well as upper bounding the effect of $T_i\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{\dagger}\mathbf{A}^{1/2}\right)\right)$. The lower bound follows from Fact D.2.2, and we need to show the upper bound:

Lemma D.2.6 *If $\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa\boldsymbol{A}$, we have for any i:*

$$\left\|T_i\left(\delta\left(\boldsymbol{I} - \boldsymbol{A}^{\frac{1}{2}}\boldsymbol{B}^{\dagger}\boldsymbol{A}^{\frac{1}{2}}\right)\right)\right\|_2 \leq 1$$

Proof The given condition is equivalent to:

$$\begin{aligned}\frac{1}{\kappa}\mathbf{A}^{\dagger} \preceq \mathbf{B}^{\dagger} \preceq \mathbf{A}^{\dagger}\\ \frac{1}{\kappa}\mathbf{I} \preceq \frac{1}{\kappa}\mathbf{A}^{\frac{1}{2}}\mathbf{B}^{\dagger}\mathbf{A}^{\frac{1}{2}} \preceq \mathbf{I}\\ 0 \preceq \mathbf{I} - \frac{1}{\kappa}\mathbf{A}^{\frac{1}{2}}\mathbf{B}^{\dagger}\mathbf{A}^{\frac{1}{2}} \preceq \left(1-\frac{1}{\kappa}\right)\mathbf{I}\end{aligned}$$

Recall that $\delta = 1 + \frac{1}{\kappa}$, so $\left(1 - \frac{1}{\kappa}\right)\delta \leq 1$ and:

$$0 \preceq \frac{1}{\delta}\left(\mathbf{I} - \frac{1}{\kappa}\mathbf{A}^{\frac{1}{2}}\mathbf{B}^{\dagger}\mathbf{A}^{\frac{1}{2}}\right) \preceq \mathbf{I}$$

Therefore, all eigenvalues of $\delta\left(\mathbf{I} - \mathbf{A}^{\frac{1}{2}}\mathbf{B}^{\dagger}\mathbf{A}^{\frac{1}{2}}\right)$ are in the range $[0, 1]$. Applying the upper bound for Chebyshev polynomials for values in this range then gives the result.

■

This allows us to prove the overall error bound

Proof of Lemma 2.1.1:

Let $\mathbf{Z}_i$ be the linear operator corresponding to running preconditioned Chebyshev iteration for i steps. In other words, $\mathbf{x}_i = \mathbf{Z}_i\mathbf{b}$. Since all terms the Chebyshev polynomial are of the form $\mathbf{B}^{-1}\mathbf{A}\mathbf{B}^{-1}\ldots\mathbf{A}\mathbf{B}^{-1}$, the operator is symmetric. Therefore it suffices to show its spectral guarantees. The goal condition of $(1-\epsilon)\mathbf{A}^{\dagger} \preceq \mathbf{Z} \preceq (1+\epsilon)\mathbf{A}^{\dagger}$ is equivalent to:

$$\begin{aligned}(1-\epsilon)\,\mathbf{I} \preceq \mathbf{A}^{1/2}\mathbf{Z}\mathbf{A}^{1/2} \preceq (1+\epsilon)\,\mathbf{I}\\ -\epsilon\mathbf{I} \preceq \mathbf{A}^{1/2}\left(\mathbf{Z} - \mathbf{A}^{\dagger}\right)\mathbf{A}^{1/2} \preceq \epsilon\mathbf{I}\\ \left\|\mathbf{A}^{1/2}\left(\mathbf{Z} - \mathbf{A}^{\dagger}\right)\mathbf{A}^{1/2}\right\|_2 \leq \epsilon\end{aligned}$$

Since Lemma D.2.5 holds for any vector $\mathbf{x}$, it can also be viewed as a statement about the corresponding operators. As the input is $\mathbf{b} = \mathbf{A}\bar{\mathbf{x}}$, the difference between $\mathbf{x}_i$ and $\bar{\mathbf{x}}$ equals to $\mathbf{Z}_i\mathbf{A}\bar{\mathbf{x}} - \bar{\mathbf{x}}$.

$$\begin{aligned}\mathbf{Z}_i\mathbf{A} - \mathbf{I} = \frac{1}{T_i(\delta)}\mathbf{A}^{\dagger 1/2}T_i\left(\delta\left(\mathbf{I} - \mathbf{B}^{\dagger}\mathbf{A}\right)\right)\mathbf{A}^{1/2}\\ \mathbf{A}^{1/2}\left(\mathbf{Z}_i - \mathbf{A}^{\dagger}\right)\mathbf{A}^{1/2} = \frac{1}{T_i(\delta)}T_i\left(\delta\left(\mathbf{I} - \mathbf{B}^{\dagger}\mathbf{A}\right)\right)\end{aligned}$$

By Lemma D.2.6, the spectral norm of this operator can be bounded by $\frac{1}{T_i\left(\frac{1}{\delta}\right)}$. The lower bound on Chebyshev polynomials from Fact D.2.2 gives that when $i \geq O(\sqrt{\kappa}\log(1/\epsilon))$ for an appropriate constant in the big-O, $T_i(\delta) \geq 10\epsilon^{-1}$. ■

D.3 Chebyshev Iteration with Round-off Errors

We now incorporate round-off errors in our analysis. There are two sources of such error here errors from evaluating $\mathbf{B}^{\dagger}$, and errors from vector additions and matrix-vector multiplications. Our proof is based on showing that errors incurred at each step of Chebyshev iteration compound at a rate polynomial to the iteration count.

In addition to showing that the total deviation is small, we also need to bound the norms of vectors passed as input to $\textsc{Solve}_{\mathbf{B}}$. Our overall bound is as follows:

Lemma 2.6.3 (Preconditioned Chebyshev Iteration) *Given a positive semi-definite matrix* $\boldsymbol{A}$ *with* m *non-zero entries and all non-zero eigenvalues between* $\frac{1}{\lambda}$ *and* λ*, a positive semi-definite matrix* $\boldsymbol{B}$ *such that* $\boldsymbol{A} \preceq \boldsymbol{B} \preceq \kappa\boldsymbol{A}$*, and a* $(0.2, \epsilon_1, \mathcal{T})$*-solver for* $\boldsymbol{B}$*,* $\textsc{Solve}_{\boldsymbol{B}}$*. If* $\epsilon_m < \frac{\epsilon_1}{20\kappa\lambda}$*, preconditioned Chebyshev iteration gives a routine* $\textsc{Solve}_{\boldsymbol{A}}$ *that is a* $(0.1, O(\kappa\epsilon_1), O(\sqrt{\kappa}(m+\mathcal{T})))$*-solver for* $\boldsymbol{A}$*. Furthermore, all intermediate values in a call* $\textsc{Solve}_{\boldsymbol{A}}(\boldsymbol{b})$ *have magnitude at most* $O(\sqrt{\lambda}\left(\|\boldsymbol{b}\|_{\boldsymbol{A}^{\dagger}} + \kappa\epsilon_1\right))$ *and the calls to* $\textsc{Solve}_{\boldsymbol{B}}$ *involve vectors* $\boldsymbol{b}'$ *such that* $\|\boldsymbol{b}'\|_{\boldsymbol{B}^{\dagger}} \leq \|\boldsymbol{b}\|_{\boldsymbol{A}^{\dagger}} + O\left(\kappa\epsilon_1\right)$*.*

Consider the recurrence as Chebyshev iteration with a slightly less aggressive step and $\delta = 1 + \frac{1}{2\kappa}$.

Base case:

$$\mathbf{x}_0 = \mathbf{0}$$
$$\mathbf{x}_1 = \frac{1}{2}\textsc{Solve}_{\mathbf{B}}\left(\mathbf{b}\right)$$

Iteration:

$$\mathbf{x}_{i+1} = \frac{2\delta T_i\left(\delta\right)}{T_{i+1}\left(\delta\right)}\left(\mathbf{x}_i - \frac{1}{2}\textsc{Solve}_{\mathbf{B}}\left(\mathbf{A}\mathbf{x}_i - \mathbf{b}\right)\right) - \frac{T_{i-1}\left(\delta\right)}{T_{i+1}\left(\delta\right)}\mathbf{x}_{i-1}$$

Let $\mathbf{Z}_{\mathbf{B}}$ be the matrix corresponding to $\textsc{Solve}_{\mathbf{B}}$. Then $0.8\mathbf{B}^{\dagger} \preceq \mathbf{Z}_{\mathbf{B}} \preceq 1.2\mathbf{B}^{\dagger}$. Combining this with $\mathbf{A} \preceq \mathbf{B} \preceq \kappa\mathbf{B}$ gives $\mathbf{A} \preceq 2\mathbf{Z}_{\mathbf{B}}^{\dagger} \preceq 4\kappa\mathbf{A}$. Therefore the 'ideal' iteration process similar to Section D.2 would involve multiplying by $\frac{1}{2}\mathbf{Z}_{\mathbf{B}}$ at each iteration. It is more convenient to view this algorithm as deviating from this process with a single error vector.

Lemma D.3.1 *If* $\epsilon_m \leq \frac{\epsilon_1}{20\lambda\kappa}$*, then preconditioned Chebyshev iteration under round-off errors can be described by the following recurrence where* $\boldsymbol{err}_i$ *satisfies* $\|\boldsymbol{err}_i\|_{\boldsymbol{A}} \leq 9\epsilon_1$*.*

$$\boldsymbol{x}_0 = \boldsymbol{0}$$
$$\boldsymbol{x}_1 = \frac{1}{2}\textsc{Solve}_{\boldsymbol{B}}\left(\boldsymbol{b}\right) + \boldsymbol{err}_1$$
$$\boldsymbol{x}_{i+1} = \frac{2\delta T_i\left(\delta\right)}{T_{i+1}\left(\delta\right)}\left(\boldsymbol{x}_i - \frac{1}{2}\boldsymbol{Z}_{\boldsymbol{B}}\left(\boldsymbol{A}\boldsymbol{x}_i - \boldsymbol{b}\right)\right) - \frac{T_{i-1}\left(\delta\right)}{T_{i+1}\left(\delta\right)}\boldsymbol{x}_{i-1} + \boldsymbol{err}_i$$

Proof

The presence of round-off errors causes us to deviate from this in three places:

1. Errors from computing $\mathbf{Ax} - \mathbf{b}$.

2. Deviation between output of $\textsc{Solve}_{\mathbf{B}}(\mathbf{b}')$ and $\mathbf{Z_B b}'$.

3. Errors from adding (rescaled) vectors.

We will denote these error vectors $\mathbf{err}_i^{(1)}$, $\mathbf{err}_i^{(2)}$, $\mathbf{err}_i^{(3)}$ respectively. The assumptions about round-off error and guarantee of $\textsc{Solve}_{\mathbf{B}}$ gives: $\left\|\mathbf{err}_i^{(1)}\right\|_2, \left\|\mathbf{err}_i^{(3)}\right\|_2 \leq \epsilon_m$ and $\left\|\mathbf{err}_i^{(2)}\right\|_{\mathbf{B}} \leq \epsilon_1$. The recurrence under round-off error then becomes:

$$\begin{aligned}
\mathbf{x}_0 &= \mathbf{0}\\
\tilde{\mathbf{x}}_1 &= \textsc{Solve}_{\mathbf{B}}\mathbf{b} + \mathbf{err}_1^{(2)}\\
\tilde{\mathbf{x}}_{i+1} &= \frac{2\delta T_i(\delta)}{T_{i+1}(\delta)}\left(\tilde{\mathbf{x}}_i - \left(\mathbf{Z_B}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\right) + \mathbf{err}_i^{(2)}\right)\right) - \frac{T_{i-1}(\delta)}{T_{i+1}(\delta)}\tilde{\mathbf{x}}_{i-1} + \mathbf{err}_i^{(3)}
\end{aligned}$$

Aggregating the errors in the iterative step gives:

$$\tilde{\mathbf{x}}_{i+1} = \frac{2\delta T_i(\delta)}{T_{i+1}(\delta)}\left(\tilde{\mathbf{x}}_i - \mathbf{Z_B}\left(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b}\right)\right) - \frac{T_{i-1}(\delta)}{T_{i+1}(\delta)}\tilde{\mathbf{x}}_{i-1} - \frac{2\delta T_i(\delta)}{T_{i+1}(\delta)}\left(\mathbf{Z}\mathbf{err}_i^{(1)} + \mathbf{err}_i^{(2)}\right) + \mathbf{err}_i^{(3)}$$

So we can set $\mathbf{err}_i$ to the trailing terms, and it suffices to bound its norm. We first bound the coefficient in front of the first two terms. Since $1 \leq \delta \leq 2$, Fact D.2.3 gives that the coefficient $\frac{2\delta T_i(\delta)}{T_{i+1}(\delta)}$ is at most $4 \cdot 2 \leq 8$. We can now bound the $\mathbf{A}$-norm of the terms individually. Since the maximum eigenvalue of $\mathbf{A}$ is λ, $\left\|\mathbf{err}_i^{(3)}\right\|_{\mathbf{A}} \leq \lambda\epsilon_m$. Also, since $\mathbf{A} \preceq \mathbf{B}$, $\left\|\mathbf{err}_i^{(2)}\right\|_{\mathbf{A}} \leq \left\|\mathbf{err}_i^{(2)}\right\|_{\mathbf{B}} \leq \epsilon_1$. $\left\|\mathbf{Z_B}\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}}$ can be rewritten as $\left\|\mathbf{err}_i^{(1)}\right\|_{\mathbf{Z_B A Z_B}}$. Since $\mathbf{A} \preceq 2\mathbf{Z}_{\mathbf{B}}^{\dagger} \preceq 4\kappa\mathbf{A}$, applying Lemma 1.6.6 gives $\mathbf{Z_B A Z_B} \preceq 2\mathbf{Z_B}$. The spectrum of $\mathbf{Z_B}$ can also be bounded using the given condition that $\mathbf{A}^{\dagger} \preceq \lambda\mathbf{I}$, giving $\mathbf{Z_B} \preceq 2\kappa\lambda$ and in turn $\left\|\mathbf{Z_B}\mathbf{err}_i^{(1)}\right\|_{\mathbf{A}} \leq 2\kappa\lambda\epsilon_m$. Combining these gives that the total deviation is at most $16\kappa\lambda\epsilon_m + 8\epsilon_1 + \lambda\epsilon_m \leq 8\epsilon_1 + 17\kappa\lambda\epsilon_m$. This also serves as a bound for the case of $i = 1$ where only the second error term is present. The assumption of $\epsilon_m \leq \frac{\epsilon_1}{20\kappa\lambda}$ then gives the overall bound. ■

We will show that the deviations caused by $\mathbf{err}_i$ accumulate in a controllable way. Specifically we show that this accumulation is described precisely by Chebyshev polynomials of the second kind. We will prove the following statement by induction:

Lemma D.3.2 *The discrepancy between the approximate solution $\tilde{\mathbf{x}}_i$ and the solution that would* *be obtained in the absence of round-off errors, $\boldsymbol{x}_i$, is:*

$$T_i(\delta)(\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i) = \sum_{j=1}^{i} T_j(\delta)\boldsymbol{A}^{\dagger 1/2} U_{i-j}\left(\delta\left(\boldsymbol{I} - \boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}\right)\right)\boldsymbol{A}^{1/2}\boldsymbol{err}_j$$

Where $U_i(x)$ is a Chebyshev polynomial of the second kind.

Proof The proof is by induction. The base case of $i = 0$ and $i = 1$ can be checked using the fact that $U_0(x) = 1$. The inductive case can be proven by isolating each of the $\mathbf{err}_j$ terms and checking that the coefficients satisfy the recurrence for Chebyshev polynomials of the second kind, which is the same as the recurrence for Chebyshev polynomials of the first kind.

$$\begin{aligned}
&T_{i+1}(\delta)(\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1})\\
&= T_{i+1}(\delta)\tilde{\mathbf{x}}_{i+1} - T_{i+1}(\delta)\mathbf{x}_{i+1}\\
&= 2\delta T_i(\delta)(\tilde{\mathbf{x}}_i - \mathbf{Z_B}(\mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b})) - T_{i-1}(\delta)\tilde{\mathbf{x}}_{i-1} + T_{i+1}(\delta)\mathbf{err}_{i+1}\\
&\quad - (2\delta T_i(\delta)(\mathbf{x}_i - \mathbf{Z_B}(\mathbf{A}\mathbf{x}_i - \mathbf{b})) - T_{i-1}(\delta)\mathbf{x}_{i-1})\\
&= 2\delta T_i(\delta)(\tilde{\mathbf{x}}_{i-1} - \mathbf{x}_{i-1} - \mathbf{Z_B}(\tilde{\mathbf{x}}_i - \mathbf{x}_i)) - T_{i-1}(\delta)\mathbf{A}(\tilde{\mathbf{x}}_{i-1} - \mathbf{x}_{i-1}) + T_{i+1}(\delta)\mathbf{err}_{i+1}\\
&= 2\delta T_i(\delta)\mathbf{A}^{\dagger 1/2}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{Z_B}\mathbf{A}^{1/2}\right)\mathbf{A}^{1/2}(\tilde{\mathbf{x}}_{i-1} - \mathbf{x}_{i-1}) - T_{i-1}(\delta)(\tilde{\mathbf{x}}_{i-2} - \mathbf{x}_{i-2}) + T_{i+1}(\delta)\mathbf{err}_{i+1}
\end{aligned}$$

Substituting in the inductive hypothesis gives:

$$\begin{aligned}
&T_{i+1}(\delta)(\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1})\\
&= 2\delta\mathbf{A}^{\dagger 1/2}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\mathbf{A}^{1/2}\mathbf{A}^{\dagger 1/2}\sum_{j=1}^{i} T_j(\delta)U_{i-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\\
&\quad - \sum_{j=1}^{i-1} T_j(\delta)\mathbf{A}^{\dagger 1/2}U_{i-1-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j + T_{i+1}(\delta)\mathbf{err}_{i+1}
\end{aligned}$$

As $U_{-1}(x) = 0$, we can also include in the $j = i$ term in the second summation:

$$\begin{aligned}
&T_{i+1}(\delta)(\tilde{\mathbf{x}}_{i+1} - \mathbf{x}_{i+1})\\
&= \sum_{j=1}^{i}\mathbf{A}^{\dagger 1/2}\left(2\delta T_j(\delta)\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\mathbf{A}\right)U_{i-j}\left(\frac{1}{\delta}\left(\mathbf{I} - \mathbf{B}^{-1}\mathbf{A}\right)\right)\right.\\
&\qquad \left. - U_{i-j-1}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j + T_{i+1}(\delta)\mathbf{err}_{i+1}\\
&= \sum_{j=1}^{i} T_j(\delta)\mathbf{A}^{\dagger 1/2}U_{i+1-j}\left(\delta\left(\mathbf{I} - \mathbf{B}^{-1}\mathbf{A}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j + T_i(\delta)\mathbf{A}^{\dagger 1/2}\mathbf{A}^{1/2}\mathbf{err}_i
\end{aligned}$$

Since $U_0(x) = 1$, $U_0\left(\frac{1}{\delta}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right) = \mathbf{I}$, and we may multiply the last term by it. Hence the inductive hypothesis holds for $i + 1$ as well. ■

Using this Lemma as well as the relations between the two kinds of Chebyshev polynomials given in Fact D.2.4, we can prove that the error in the **A**-norm is amplified by at most **poly** (i).

Lemma D.3.3 *The accumulation of errors after i iterations can be bounded by:*

$$\left\|\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i\right\|_{\boldsymbol{A}} \leq \sum_j 8i \left\|\boldsymbol{err}_j\right\|_{\boldsymbol{A}}$$

Proof By the bound on total error given in Lemma D.3.2 above, and the property of norms, we have:

$$\begin{aligned}
\left\|\tilde{\mathbf{x}}_i - \mathbf{x}_i\right\|_{\mathbf{A}} &= \frac{1}{T_i(\delta)}\left\|\sum_{j=1}^{i} T_j(\delta)\,\mathbf{A}^{\dagger}1/2U_{i-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{err}_j\right\|_{\mathbf{A}} \\
&\leq \sum_j \frac{T_j(\delta)}{T_i(\delta)}\left\|\mathbf{A}^{\dagger 1/2}U_{i-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\right\|_{\mathbf{A}} \\
&\leq \sum_j 8\left\|\mathbf{A}^{\dagger 1/2}U_{i-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\right\|_{\mathbf{A}} \qquad \text{By Fact D.2.3}
\end{aligned}$$

Applying the decomposition of $U_i(x)$ into a sum of $T_j(x)$ from Fact D.2.4 gives:

$$\left\|\mathbf{A}^{\dagger 1/2}U_{i-j}\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\right\|_{\mathbf{A}} \leq \sum_{k=0}^{i-j}\left\|\mathbf{A}^{\dagger 1/2}T_k\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\right\|_{\mathbf{A}}$$

Once again we apply the shrinkage properties given in Lemma D.2.6 to each of the terms:

$$\begin{aligned}
&\left\|\mathbf{A}^{\dagger 1/2}T_k\left(\delta\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)\right)\mathbf{A}^{1/2}\mathbf{err}_j\right\|_{\mathbf{A}}^2 \\
&= \mathbf{err}_j^T\mathbf{A}^{1/2}\left(\mathbf{I} - \mathbf{A}^{1/2}\mathbf{B}^{-1}\mathbf{A}^{1/2}\right)^2\mathbf{A}^{1/2}\mathbf{err}_j \\
&\leq \mathbf{err}_j^T\mathbf{A}\mathbf{err}_j = \left\|\mathbf{err}_j\right\|_{\mathbf{A}}^2
\end{aligned}$$

Taking square roots of both sides and summing gives the overall bound. ■

Proof of Lemma 2.6.3: The properties of the exact linear operator follows from Lemma 2.1.1 and the condition that $\mathbf{A}^{\dagger} \preceq 2\mathbf{Z}_{\mathbf{B}}^{1/2} \preceq 4\kappa\mathbf{A}^{\dagger}$. The total deviation is given by Lemmas D.3.1 and D.3.3:

$$\begin{aligned}
\left\|\tilde{\mathbf{x}}_i - \mathbf{x}_i\right\|_{\mathbf{A}} &\leq \sum_j 8i\left\|\mathbf{err}_j\right\|_{\mathbf{A}} \\
&\leq i \cdot 8i \cdot 9\epsilon_1 \leq O(\kappa)\epsilon_1
\end{aligned}$$

The vector passed to $\textsc{Solve}_{\mathbf{B}}$, at iteration i is:

$$\begin{aligned}\mathbf{b}' &= \mathbf{A}\tilde{\mathbf{x}}_i - \mathbf{b} + \mathbf{err}_i^{(1)}\\ &= \mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right) + \mathbf{err}_i^{(1)}\end{aligned}$$

Since $\left\|\mathbf{err}_i^{(1)}\right\|_2 \leq \epsilon_m$ and $\mathbf{B} \preceq \kappa\mathbf{A} \preceq \kappa\lambda \cdot \mathbf{I}$, $\left\|\mathbf{err}_i^{(1)}\right\|_{\mathbf{B}} \leq \sqrt{\kappa\lambda}\epsilon_m \leq \epsilon_1$. It remains to bound the $\mathbf{B}^{\dagger}$-norm of $\mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)$ The condition $\mathbf{A} \preceq \mathbf{B}$ gives $\mathbf{B}^{\dagger} \preceq \mathbf{A}^{\dagger}$ and:

$$\begin{aligned}\left\|\mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)\right\|_{\mathbf{B}^{\dagger}} &= \sqrt{\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)\mathbf{A}\mathbf{B}^{\dagger}\mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)}\\ &\leq \sqrt{\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)\mathbf{A}\mathbf{A}^{\dagger}\mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)}\\ &= \left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}\end{aligned}$$

This can in turn be decomposed using the triangle inequality:

$$\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} \leq \left\|\tilde{\mathbf{x}}_i - \mathbf{x}_i\right\|_{\mathbf{A}} + \left\|\mathbf{x}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}}$$

The first term was bounded above by $O(\kappa)\epsilon_1$, while the second term can be bounded by $\|\mathbf{b}\|_{\mathbf{A}^{\dagger}}$ by the convergence of exact Chebyshev iteration in Lemma 2.1.1. Combining these gives:

$$\left\|\mathbf{b}'\right\|_{\mathbf{B}^{\dagger}} \leq \|\mathbf{b}\|_{\mathbf{A}^{\dagger}} + O\left(\kappa\epsilon_1\right)$$

These bounds also allows us to bound the ℓ_2-norm, and in turn magnitudes entries of all intermediate vectors. Then given assumption that all non-zero eigenvalues of $\mathbf{A}$ are between $\frac{1}{\lambda}$ and λ implies:

$$\begin{aligned}\left\|\tilde{\mathbf{x}}_i\right\|_2, \left\|\mathbf{A}\tilde{\mathbf{x}}_i\right\|_2 &\leq \sqrt{\lambda}\left\|\tilde{\mathbf{x}}_i\right\|_{\mathbf{A}}\\ &\leq \sqrt{\lambda}\left(\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} - \bar{\mathbf{x}}\right)\\ &\leq \sqrt{\lambda}O\left(\|\mathbf{b}\|_{\mathbf{A}^{\dagger}} + \kappa\epsilon_1\right)\\ \left\|\mathbf{b}'\right\|_2 &\leq \left\|\mathbf{A}\left(\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right)\right\|_2 + \left\|\mathbf{err}_i^{(1)}\right\|_2\\ &\leq \sqrt{\lambda}\left\|\tilde{\mathbf{x}}_i - \bar{\mathbf{x}}\right\|_{\mathbf{A}} + \epsilon_m\\ &\leq O\left(\sqrt{\lambda}\left(\|\mathbf{b}\|_{\mathbf{A}^{\dagger}} + \kappa\epsilon_1\right)\right)\end{aligned}$$

■