**Abstract**

# Approximate Gaussian Elimination

Rasmus Kyng
2017

We show how to perform sparse approximate Gaussian elimination for undirected Laplacian matrices and directed Laplacian matrices. This leads to the simplest known nearly-linear time solvers for linear equations in these matrices. For directed Laplacians, the approach gives the first known nearly-linear time linear equation solver. This in turn gives nearly-linear time algorithms for computing many interesting properties of a random walk matrix, including its stable distribution.

For undirected Laplacians, this is the first solver that is based purely on random sampling, and does not use any graph theoretic constructions such as low-stretch trees, sparsifiers, or expanders. The crux of our analysis is the use of matrix martingales to bound the error accumulation of the algorithm.

To develop and analyze the algorithm for directed Laplacians, we build on and extend recent developments in spectral graph theory for directed graphs, which combined with a matrix martingale analysis leads to a simple and fast solver. The algorithm relies on a surprising routine for performing unbiased, sparse approximation of the Cholesky factorization on a type of matrix known as an Eulerian Laplacian.

We also show how Approximate Gaussian Elimination can be used to compute sparse approximations of Schur complements of Laplacians, and how this in turn leads to the first algorithm for sampling random spanning trees in dense and/or weighted graphs faster than matrix multiplication time. This is the first improvement in running time for this problem in dense graphs in more than 20 years.

# Approximate Gaussian Elimination

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Rasmus Kyng

Dissertation Director: Daniel A. Spielman

July, 2017

# Contents

# Acknowledgments

I owe a lot to a lot of people who helped me get through graduate school. First on that list is Dan, who was an incredible advisor. He taught me to do research, and he has always been extremely generous with his advice and ideas. I continue to aspire to work as he does, and be as helpful and kind as he is.

I would like to thank my mum and my dad, my brother and sisters, and the rest of my incredible family, whom I have spent too much time away from, but could always rely on.

I want to thank Anup and Sushant, who were there with me in the daily grind, and who helped me become a lot less confused about some things. Fortunately that's all it takes. And I want to thank my many and very impressive coauthors, or more importantly, colleagues in research. Working with such good people is a joy, and working without them would hardly be worth it. I especially want to thank Richard, who is unfailingly generous and good to the many people around him. And I want to thank Peng, whose tireless work got us far.

I owe many thanks to my colleagues in the Department of Computer Science, and remember fondly the team behind Operation Crushing Fist. The Yale Institute for Network Science has been my home at Yale for a few years now, and I want to thank the many people who made it a delight to work there, and to thank Emily, Kim, and Nicholas, who took such good care of me.

I want to thank my committee for agreeing to be on it in the first place. I hope they will enjoy reading this thesis.

To Ada, Martin, and Ludwig, I also owe a lot: They were a delight to know and to work with, and they inspired me to head over here.

I have had the great fortune of meeting many wonderful people in New Haven, and made friends there who got me through many good times and bad. I cannot list them all, but give my thanks to the residents of HGS, the boys on Mansfield, and to Gareth and Sam, to Sam, Zlatko, Yehan, Andrew, Luca, Will, and Jess, and many more.

Many generations of Brits in New Haven and visitors from Cambridge have made my stay here much better, but Angus more than anyone—he's been there all the way with me, and always there for me.

Without Yale Swing and Blues (and now Fusion), I would have surely gone mad, and I am lucky to have had so many dances and experienced so much happiness with the people there, especially the generations of the swing crew, Ning, Dan, Kevin, Eric, Renee, and Laurel.

I also want to thank Anders, Christoffer, Oluf, Kaare, and my other good friends back in Denmark. It is always a joy to come back to them, and I am grateful to still be a small part of their lives.

Finally, I have to thank Linda and Iver, my high school teachers who first got me seriously interested in mathematics in general, and linear algebra in particular. That has turned out ok so far.

For Solvej, Morten, and my wonderful siblings.

# Chapter 1

# Introduction

**Solving Linear Equations.** Solving systems of linear equations is a fundamental algorithmic problem. Gaussian Elimination, an algorithm for solving linear equations, has been studied since 179 CE, and possibly much earlier [Cal99].

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we can solve the linear system $\mathbf{A}\boldsymbol{x} = \boldsymbol{c}$ in $O(n^\omega)$ time, where $\omega$ is the matrix multiplication constant, for which the best currently known bound is $\omega < 2.3727$ [Str69, Wil12]. Such a running time bound is cost prohibitive for the large sparse matrices often encountered in practice. However, for certain classes of matrices, we can develop much faster solvers for systems of linear equations. Two of the most important such classes are Symmetric Diagonally Dominant (SDD) and Laplacian matrices:

- A symmetric matrix $\mathbf{M}$ is called Symmetric and Diagonally Dominant (SDD) if for all $i$, $\mathbf{M}(i,i) \geq \sum_{j \neq i} |\mathbf{M}(i,j)|$.
- An SDD matrix $\mathbf{U}$ is a Laplacian if $\mathbf{U}(i,j) \leq 0$ for $i \neq j$, and for all $i$, $\sum_j \mathbf{U}(i,j) = 0$. A Laplacian matrix is naturally associated with a graph on its vertices, where $i$ and $j$ are adjacent if $\mathbf{U}(i,j) \neq 0$.

When $\mathbf{M}$ is an $n \times n$ SDD matrix with $m$ non-zeros, and polynomially bounded entries, the linear system $\mathbf{M}\boldsymbol{x} = \boldsymbol{c}$ can be solved approximately to $\epsilon$-accuracy in $O((m+n)\log^{1/2+o(1)} n \log(1/\epsilon))$ time [ST14a, CKM+14a]. The solvers work with Laplacian matrices, but a reduction due to Gremban [Gre96], shows that an approximate solver for Laplacians can be used to solve SDD linear systems with only constant factor overhead in the running time. The result of Spielman and Teng spurred a series of major developments in fast graph algorithms, sometimes referred to as "the Laplacian Paradigm" of designing graph algorithms [Ten10].

The asymptotically fastest known algorithms for Maximum Flow in directed unweighted graphs [Mad13, Mad16], Negative Weight Shortest Paths and Maximum weight matchings [CMSV17], Minimum cost flows and Lossy generalized flows [LS14, DS08a], sampling spanning trees in unweighted graphs [KM09b, MST15a] all rely on fast Laplacian linear system solvers. Other applications include solutions of partial differential equations via the finite element method [Str86, BHV08], semi-supervised learning on graphs [ZGL03, ZS04, ZBL+04], graph partitioning [OSV12], and more [CKM+11, LS13, MST15b, KMP12, SS11a, DS08b, She09, KM11]). Without directly using linear system solvers, applications of ideas and techniques from the solver of Spielman and Teng have been useful a range of problems in other domains [She13, KLOS14, DKP+16, Mad10, ADK+16])

Only very recently were close-to-linear time linear system solvers obtained for matrix families beyond SDD matrices. Nearly-linear time solvers were extended to symmetric M-matrices

[DS08a]. Beyond these, nearly linear-time solvers have been obtained for Block Diagonally Dominant matrices and Connection Laplacians [KLP+16]. Progress on spectral theory of directed graphs [CKP+16a,CKP+16b,CKK+17] made it possible to develop the first nearly-linear time solver for Row (respectively Column) Diagonally Dominant matrices, and Directed Laplacians, which we present in this thesis.

- A matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is Row Diagonally Dominant if for all $i$, $\mathbf{M}(i,i) \geq \sum_{j \neq i} |\mathbf{M}(i,j)|$. Similarly, the matrix is Column Diagonally Dominant if for all $i$, $\mathbf{M}(i,i) \geq \sum_{j \neq i} |\mathbf{M}(i,j)|$.
- A matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$ is called a *directed Laplacian* if (1) its off diagonal entries are non-positive, i.e. $\mathbf{L}(i,j) \leq 0$ for all $i \neq j$, and (2) it satisfies $\vec{1}^\top \mathbf{L} = \vec{0}$, i.e. $\mathbf{L}(i,i) = -\sum_{j \neq i} \mathbf{L}(j,i)$ for all $i$.

Our linear equation solver for Directed Laplacians gives the first nearly-linear time algorithms for computing $\epsilon$-approximations to various properties of random walks on directed graphs, including stationary distributions, personalized PageRank vectors, hitting times, and escape probabilities [CKP+16a].

There are essentially two main approaches to solving systems of linear equations, from which most other algorithms are derived (e.g. see [TBI97]). The first approach is "direct solvers", of which the most basic is Gaussian Elimination Applied to a non-singular matrix $\mathbf{M}$, Gaussian Elimination produces a factorization (LU-decomposition) $\mathbf{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$, where $\boldsymbol{\mathcal{L}}$ is a lower-triangular matrix and $\boldsymbol{\mathcal{U}}$ is an upper-triangular matrix. Such a factorization allows us to solve a system $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$ by computing $\boldsymbol{x} = \mathbf{M}^{-1}\boldsymbol{b} = (\boldsymbol{\mathcal{U}}^{-1})^\top \boldsymbol{\mathcal{L}}^{-1}\boldsymbol{b}$. The inverse of $\boldsymbol{\mathcal{L}}$, and $\boldsymbol{\mathcal{U}}$ can be applied quickly since linear systems in lower- and upper-triangular matrices can be easily solved in time proportional to the number of non-zero entries in the matrix by back- and forward-substitution. When the input matrix is positive semi-definite, Gaussian Elimination can be used to factor it as $\mathbf{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$, where $\boldsymbol{\mathcal{L}}$ is lower triangular. This is referred to as a Cholesky factorization.

The second main approach to solving systems of linear equations is iterative methods. Roughly speaking, iterative methods start from a trivial initial guess for the solution, and repeatedly improve the approximate solution by correcting for errors in the current estimate. Iterative methods converge quickly if the estimates of the error in the current approximate solution are are sufficiently accurate.

The most basic iterative method is Iterative Refinement. To solve the system $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$, where $\mathbf{M}$ is a square matrix, Iterative Refinement starts with a trivial initial guess for the solution, usually $\boldsymbol{x}^{(0)} = \mathbf{0}$, and then updates the solution by subtracting the residual error in $\boldsymbol{x}^{(i)}$ given by $\mathbf{M}\boldsymbol{x}^{(i)} - \boldsymbol{b}$, i.e. $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - (\mathbf{M}\boldsymbol{x}^{(i)} - \boldsymbol{b})$. Iterative Refinement is only guaranteed to convergence when $\mathbf{M}$ is in some sense very similar to the identity matrix.

Both direct solvers and iterative methods have their drawbacks: Direct methods usually produce dense matrices, which inherently makes them slow, especially when the input matrix is sparse. Iterative methods require good ways to produce estimates of the error in the current solution, which is usually difficult when the input matrix has a large condition number.

The fundamental obstacle to using Gaussian Elimination to quickly solve systems of linear equations is that $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ can be dense matrices even if the original matrix $\mathbf{M}$ is sparse. The reason is that the key step in Gaussian Elimination, eliminating a variable, say $\boldsymbol{x}_i$, from a system of equations, creates a new coefficient $\mathbf{M}'(j,k)$ for every pair $j,k$ such that $\mathbf{M}(j,i)$ and $\mathbf{M}(i,k)$ are non-zero. This phenomenon is called *fill-in*. For Laplacian systems, eliminating the first variable corresponds to eliminating the first vertex in the graph, and the fill-in corresponds to adding a clique on all the neighbors of the first vertex. Sequentially eliminating variables often produces a sequence of increasingly-dense systems, resulting in an $O(n^3)$ worst-case time even for sparse $\mathbf{M}$.

The fundamental obstacle to using iterative methods to quickly solve linear equations is the need to produce sufficiently good estimates for the error in the current solution. For symmetric matrices with positive eigenvalues, the convergence rate of iterative methods depends on the condition number of the matrix, which generally an be extremely large. In the case of asymmetric matrices, for iterative methods to guarantee convergence at all, the matrix must usually be close to the identity matrix in a strong sense.

Iterative methods can sometimes be accelerated using a tool known as *preconditioning*. This technique involves constructing a linear operator $\mathbf{Z}$ that is an approximation of $\mathbf{M}$ and has the property that $\mathbf{Z}^{-1}$ can be applied cheaply. Then for any approximate solution $\boldsymbol{x}^{(i)}$, the error in the solution, $\mathbf{M}^{-1}\boldsymbol{b} - \boldsymbol{x}^{(i)}$, can be approximated by $\mathbf{Z}^{-1}(\mathbf{M}\boldsymbol{x}^{(i)} - \boldsymbol{b})$.

The central contribution of this thesis is to show that for several families of matrices, we can construct sparse and high-quality preconditioners using an approximate version of Gaussian elimination, essentially by randomly discarding most of the entries created by elimination. The preconditioner we produce is a sparse factorization $\mathbf{Z} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$ that gives an excellent approximation of $\mathbf{M}$, and ensures that $\mathbf{Z}^{-1}$ can be applied quickly.

Using these preconditioners in Iterative Refinement gives an extremely simple algorithm for solving Laplacian systems of linear equations in nearly linear time, and it lets us develop the first nearly-linear time algorithm for solving systems of linear equations in Directed Laplacians.

While this dissertation is focused on theoretical analysis of algorithms, the extremely simple algorithms we develop here for solving systems of linear equations suggest that the Laplacian paradigm may finally be reaching the stage where implementations of theoretically sound and asymptotically fast algorithms will change the way we solve a large range of problems in practice. One example is the Laplacians.jl GitHub repository, where Laplacian linear equation solvers inspired by Approximate Gaussian Elimination are actively being developed.

**Approximation of Laplacian Schur Complements.** As a byproduct of our algorithm for approximate Gaussian Elimination, we also get a technique for efficiently computing sparse approximations of Schur complements of Laplacian matrices. The Schur complement of a square matrix onto a set of variables is the remaining matrix after all other variables have been eliminated using Gaussian Elimination. There are several connections between Schur complements and random spanning trees, which we leverage to give a faster algorithm for sampling random spanning trees.

**Sampling Random Spanning Trees.** Random spanning trees are one of the most well-studied probabilistic structures in graphs. Their history goes back to the classic matrix-tree theorem due to Kirchoff in 1840s that connects the spanning tree distribution to matrix determinants [Kir47]. The task of algorithmically sampling random spanning trees has been studied extensively [Gue83, Bro89, Ald90, Kul90, Wil96, CMN96, KM09a, MST15a, HX16].

Over the past decade, sampling random spanning trees have found a few surprising applications in theoretical computer science – they were at the core of the breakthroughs in approximating the traveling salesman problem in both the symmetric [GSS11] and the asymmetric case [AGM+10]. Goyal et al. [GRV09] showed that one could construct a cut sparsifier by sampling random spanning trees.

We leverage connections between Laplacian matrix Schur complements and our new algorithm for approximating these to give the first algorithm for sampling random spanning trees in dense and/or weighted graphs faster than matrix multiplication time. This is the first improvement in running time for this problem in dense graphs in more than 20 years.

## 1.1 Prior Work

**Linear Equation Solvers for Laplacians.** Though the current best algorithm for solving a general $n \times n$ positive semidefinite linear system with $m$ non-zero entries takes time $O(\min\{mn, n^{2.2373}\})$ [Wil12], a breakthrough result by Spielman and Teng [ST04a, ST14b] showed that linear systems in graph Laplacians could be solved in time $O(m \cdot \text{poly}(\log n) \log \frac{1}{\epsilon})$. There has been a lot of progress over the past decade improving their breakthrough result, making it faster, simpler, and more parallelizable [KMP10, KMP11, KOSZ13, CKM+14b, PS14, KLP+16], and the current best running time is $O(m \log^{\frac{1}{2}} n \log \frac{1}{\epsilon})$ (up to polylog $n$ factors) [CKM+14b]. All of these algorithms have relied on graph-theoretic constructions – low-stretch trees [ST04a, KMP10, KMP11, KOSZ13, CKM+14b], graph sparsification [ST04a, KMP10, KMP11, CKM+14b, PS14], and explicit expander graphs [KLP+16].

In contrast, our algorithm requires no graph-theoretic construction, and is based purely on random sampling. Our result only uses two algebraic facts about Laplacian matrices: 1. They are closed under taking Schur complements, and 2. They satisfy the effective resistance triangle inequality (Lemma 3.1.5).

**Comparing Approximate Gaussian Elimination to Incomplete Cholesky Factorization.** A popular approach to tackling fill-in is *Incomplete Cholesky factorization*, where we throw away most of the new entries generated when eliminating variables. The hope is that the resulting factorization is still an approximation to the original matrix $L$, in which case such an approximate factorization can be used to quickly solve systems in $L$. Though variants of this approach are used often in practice, and we have approximation guarantees for some families of Laplacians [Gus78, Gua97, BGH+06], there are no known guarantees for general Laplacians to the best of our knowledge. Most variants of incomplete Cholesky in the literature are deterministic algorithms. A notable exception is a randomized rounding scheme proposed by Clarkson [Cla03] that he experimentally showed performs well on some matrices.

**Directed Laplacians.** For *symmetric* diagonally dominant matrices, which include the Laplacians of *undirected* graphs, Spielman and Teng gave an algorithm in 2004 [ST04b] to solve the corresponding linear systems in nearly-linear time, spurring the development of the "Laplacian Paradigm".

However, while this approach has been incredibly successful for symmetric linear systems and undirected graph optimization problems, comparable results for their asymmetric or directed counterparts have proven quite elusive. In particular the techniques for solving Laplacian systems seemed to rely intrinsically on multiple properties of undirected graphs, and, until recently, the best algorithms in the directed case simply treated the Laplacians as unstructured matrices and applied general linear algebraic routines, leading to super-quadratic running times.

Two recent papers [CKP+16a, CKP+16b] suggested that it may be possible to close this gap, potentially laying the foundation for a new class of nearly-linear-time algorithms for directed graphs and asymmetric linear systems. The first paper [CKP+16a] showed that linear systems involving several natural classes of asymmetric matrices, including Laplacians of directed graphs, general square column diagonally dominant matrices, and their transposes (called *row diagonally dominant* matrices), could be reduced with only polylogarithmic overhead to solving linear systems in the Laplacians of Eulerian graphs. It then showed how to use these solvers, again with only polylogarithmic overhead, to compute a wide range of fundamental quantities associated with random walks on directed graphs, including the stationary distribution, personalized PageRank vectors, hit-

ting times, and escape probabilities. The paper combined these reductions with an algorithm to solve Eulerian Laplacian systems in time $\widetilde{O}(m^{3/4}n + mn^{2/3})$ to achieve faster (but still significantly super-linear) algorithms for all of these problems.[1]

The second paper [CKP+16b] provided an improved solver for Eulerian systems that runs in almost-linear time $\widetilde{O}(m + n2^{O(\sqrt{\log n \log\log n})})$, thus providing almost-linear-time algorithms for all of the problems reduced to such a solver in [CKP+16a].

In this dissertation, we close the algorithmic gap between the directed and undirected cases (up to logarithmic factors) by providing an algorithm to solve Eulerian Laplacian systems in time $\tilde{O}(m)$.

**Schur Complements.** Very little prior work exists on Schur complement approximation. One paper showed how to approximate Schur complements of onto special submatrices of Laplacians [KLP+16], but their approach to obtaining an $\epsilon$-approximate Schur complement required $\epsilon^{-4}$ scaling of the edge density in the output, as opposed to our scaling of $\epsilon^{-2}$. These two restrictions on subsets and density mean that this earlier result would not be useful for sampling spanning trees.

**Random Spanning Trees.** One of the first major results in the study of spanning trees was Kirchoff's matrix-tree theorem, which states that the total number of spanning trees for general edge weighted graphs is equal to any cofactor of the associated graph Laplacian [Kir47].

Much of the earlier algorithmic study of random spanning trees heavily utilized these determinant calculations by taking a random integer between 1 and the total number of trees, then efficiently mapping the integer to a unique tree. This general technique was originally used in [Gue83, Kul90] to give an $O(mn^3)$-time algorithm, and ultimately was improved to an $O(n^\omega)$-time algorithm by [CMN96], where $m, n$ are the numbers of edges and vertices in the graph, respectively, and $\omega \approx 2.373$ is the matrix multiplication exponent [Wil12]. These determinant-based algorithms have the advantage that they can handle edge-weighted graphs, where the weight of a tree is defined as the product of its edge weights.[2] Despite further improvements for unweighted graphs, no algorithm prior to our work improved upon this $O(n^\omega)$ runtime in the general weighted case in over 20 years since this work. Even for unweighted graphs, nothing faster than $O(n^\omega)$ was known for dense graphs with $m \geq n^{1.78}$.

We now give a brief overview of the improvements for unweighted graphs along with a recent alternative $O(n^\omega)$ algorithm for weighted graphs.

Around the same time as the $O(n^\omega)$-time algorithm was discovered, Broder and Aldous independently showed that spanning trees could be randomly generated with random walks, where each time a new vertex is visited, the edge used to reach that vertex is added to the tree [Bro89, Ald90]. Accordingly, this results in an algorithm for generating random spanning trees that runs in the amount of time proportional to the time it takes for a random walk to cover the graph. For unweighted graphs this *cover time* is $O(mn)$ which in expectation is better than $O(n^\omega)$ in sufficiently sparse graphs and worse in dense ones. However, in the more general case of edge-weighted graphs,

1. Following the notation and terminology of the previous papers, we use $\widetilde{O}$ notation to suppress terms that are polylogarithmic in $n$, the natural condition number of the problem $\kappa$, and the desired accuracy $\epsilon$. We use the term "nearly linear" for algorithms that run in time $\tilde{O}(m) = O(m)\log^{O(1)}(n\kappa\epsilon)$, and "almost linear" for algorithms that run in time $O(m(n\kappa\epsilon^{-1})^{o(1)})$.

2. To see why this definition is natural, note that this corresponds precisely to thinking of an edge with weight $k$ as representing $k$ parallel edges and then associating all spanning trees that differ only in which parallel edges they use.

the cover time can be exponential in the number of bits used to describe the weights. Thus, this algorithm does not yield any improvement in worst-case runtime for weighted graphs. Wilson [Wil96] gave an algorithm for generating a random spanning tree in expected time proportional to the mean hitting time in the graph. This time is always upper bounded by the cover time, and it can be smaller. As with cover time, in weighted graphs the mean hitting time can be exponential in the number of bits used to describe the weights, and so this algorithm also does not yield an improvement in worst-case runtime for weighted graphs.

Kelner and Madry improved upon this result by showing how to simulate this random walk more efficiently. They observed that one does not need to simulate the portions of the walk that only visit previously visited vertices. Then, they use a low diameter decomposition of the graph to partition the graph into components that are covered quickly by the random walk and do precomputation to avoid explicitly simulating the random walk on each of these components after each is initially covered. This is done by calculating the probability that a random walk entering a component at each particular vertex exits on each particular vertex, which can be determined by solving Laplacian linear systems. This approach yields an expected runtime of $\widetilde{O}(m\sqrt{n})$ for unweighted graphs [KM09a].

This was subsequently improved for sufficiently sparse graphs with an algorithm that also uses shortcutting procedures to obtain an expected runtime of $\widetilde{O}(m^{4/3})$ in unweighted graphs [MST15a]. Their algorithm uses a new partition scheme based on effective resistance and additional shortcutting done by recursively finding trees on smaller graphs that correspond to random forests in the original graph, allowing the contraction and deletion of many edges.

Recently, Harvey and Xu [HX16] gave a simpler deterministic $O(n^\omega)$ time algorithm that uses conditional effective resistances to decide whether each edge is in the tree, contracting the edge in the graph if the edge will be in the tree and deleting the edge from the graph if the edge will not.[3] Updating the effective resistance of each edge is done quickly by using recursive techniques similar to those in [CDN89] via an extension of the Sherman-Morrison formula.

## 1.2 Results

**Approximate Gaussian Elimination for Laplacians.** Informally, the algorithm for generating the LU-decomposition of a Laplacian using Gaussian Elimination can be expressed as follows:

---

**for** $i = 1$ *to* $n - 1$ **do**

Use equation $i$ to express the variable for vertex $i$ in terms of the remaining variables.

Eliminate vertex $i$, adding a clique on the neighbors of $i$.

**end**

---

Our algorithm for generating a sparse LU-decomposition using approximate Gaussian Elimination can informally be expressed as follows (see the routine MasterCholApx, Algorithm 1, in

3. Note that for any edge $e$, there is a bijection between spanning trees of the graph in which $e$ is contracted and spanning trees of the original graph that contain $e$. Similarly, there is a bijection between spanning trees of the graph in which $e$ is deleted and spanning trees of the original graph that do not contain $e$.

Section 3.1 for a precise description):

---

Randomly permute the vertices.
**for** $i = 1$ *to* $n - 1$ **do**
  Use equation $i$ to express the variable for vertex $i$ in terms of the remaining variables.
  Eliminate vertex $i$, adding random samples from the clique on the neighbors of $i$.
**end**

---

We prove the following theorem about our algorithm, where for symmetric matrices $\mathbf{A}, \mathbf{B}$, we write $\mathbf{A} \approx_\epsilon \mathbf{B}$ if $e^\epsilon \mathbf{B} - \mathbf{A}$ and $e^\epsilon \mathbf{A} - \mathbf{B}$ are both positive semidefinite (PSD).

**Theorem 1.2.1** (Approximate Cholesky Factorization for Laplacians)**.** *Suppose $\mathbf{U} \in \mathbb{R}^{n \times n}$ is a Laplacian matrix with $m$ non-zero entries. Given a scalar $\delta < 1/n^{100}$, the algorithm* CholApx$(\mathbf{U}, \delta)$ *returns an approximate Cholesky factorization $\boldsymbol{\mathcal{L}}$ s.t. with probability at least $1 - O(\delta)$,*

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_{1/2} \mathbf{U}.$$

*The maximum number of non-zero entries in $\boldsymbol{\mathcal{L}}$ and the total running time are both bounded by $O(m \log^2(1/\delta) \log n)$.*

As an immediate consequence of using the sparse approximate Cholesky factorization combined with Iterative Refinement we get a nearly linear time Laplacian solver.

**Corollary 1.2.2** (Laplacian Linear Equation Solver)**.** *Suppose $G = (V, E)$ is a connected undirected multi-graph with positive edge weights $w : E \to \mathbb{R}_+$, and associated Laplacian $\mathbf{U}$, and that $G$ has $m$ multi-edges. Given a scalar $\delta < 1/n^{100}$, the algorithm* CholApxSolver$(\mathbf{U}, \epsilon, \delta, \boldsymbol{b})$ *returns $\tilde{\boldsymbol{x}}$ s.t. with probability at least $1 - O(\delta)$,*

$$\left\| \tilde{\boldsymbol{x}} - \mathbf{U}^+ \boldsymbol{b} \right\|_{\mathbf{U}} \leq \epsilon \left\| \mathbf{U}^+ \boldsymbol{b} \right\|_{\mathbf{U}}. \tag{1.1}$$

*The running time is bounded by $O(m \log^2(1/\delta) \log n \log(1/\epsilon))$.*

The proof of Theorem 1.2.1 is short, fitting in less than ten pages. This is in stark contrast to the original result of Spielman and Teng [ST14a], which was eventually split into three paper spanning about a hundred pages. This is in large part because improved random matrix theory lets us do away with most of the original algorithmic machinery of their result. The simplified algorithm in turn helps generalize the result beyond Laplacians.

A crucial element of the proof of Theorem 1.2.1 is the use of matrix martingales. Taking an unusual, additive perspective on the process of Gaussian Elimination allows us to analyze a sequence of interleaved eliminations and sampling, unlike previous work which dealt with these procedures separately. This additive view also lets us show that our algorithm in fact computes an unbiased estimator of the LU-decomposition of the input matrix. The interleaving of eliminations and sampling creates dependencies between random variables being analyzed, and to handle these dependencies, we use matrix martingales, in particular Tropp's Matrix Freedman's theorem [Tro11a]. Other key ingredients of the proof include the use of a randomized order of elimination, which helps us control the variance of the process, and our approach to control of leverage scores of samples. In previous solvers, leverage score estimates are obtained using fairly involved procedures (e.g. low-stretch trees, ultrasparsifiers, or the subsampling procedure due to Cohen et al. [CLM+15]). In contrast, our solver starts with the crudest possible estimates of 1 for every edge, and then uses the triangle inequality for effective resistances (Lemma 3.1.5) to obtain estimates for the new edges generated. We show that these estimates suffice for constructing a nearly linear time Laplacian solver.

**Approximate Gaussian Elimination for Directed Laplacians.** We develop the first nearly-linear time solver for linear systems in Directed Laplacians. The core of our algorithm is the use of approximate Gaussian elimination to compute a sparse approximate LU-decomposition of Eulerian Laplacians – a subclass of Directed Laplacians where the associated graph has weighted in-degree equal to out-degree at every vertex. We combine this with a reduction from the general case of solving linear equations in Directed Laplacians to systems in Eulerian Laplacians, developed in [CKP+16a].

Our algorithm for approximate LU-decomposition of an Eulerian Laplacian is EulerianLU, and its performance is described by the theorem below.

**Theorem 1.2.3** (Approximate LU-decomposition for Eulerian Laplacians)**.** *Given an Eulerian Laplacian* $\mathbf{L} \in \mathbb{R}^{n\times n}$ *with* $m$ *nonzero entries and any* $\epsilon \in (0, 1/2)$*, and* $\delta < 1/n$*, in* $\widetilde{O}(m + n\epsilon^{-8}\log^{O(1)}(1/\delta))$ *time, with probability at least* $1-O(\delta)$ *the algorithm* $\textsc{EulerianLU}(\mathbf{L}, \delta, \epsilon)$*, produces lower and upper triangular matrices* $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{n\times n}$ *and* $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{n\times n}$ *such that for some symmetric PSD matrix* $\mathbf{F} \approx_{poly(n)} (\mathbf{L}+\mathbf{L}^\intercal)/2$*,* $(\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}})^\top \mathbf{F}^\dagger (\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}) \succeq 1/O(\log^2 n)\cdot \mathbf{F}$*,* $\|\mathbf{F}^{\dagger/2}(\mathbf{L}-\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}})\mathbf{F}^{\dagger/2}\|_2 \le \epsilon$ *and* $\max[\mathrm{nnz}(\boldsymbol{\mathcal{L}}), \mathrm{nnz}(\boldsymbol{\mathcal{U}})] \le n\log^{O(1)}(1/\delta)\cdot\epsilon^{-6}$.

Using the approximate LU-decomposition as a preconditioner inside Iterative Refinement (see Lemma 5.0.6 [4]), we then obtain a linear equation solver for Eulerian Laplacians.

**Corollary 1.2.4** (Nearly-Linear Time Solver for Eulerian Laplacians)**.** *Given an Eulerian Laplacian* $\mathbf{L} \in \mathbb{R}^{n\times n}$ *and a vector* $\boldsymbol{b} \in \mathbb{R}^n$ *with* $\boldsymbol{b} \perp \mathbf{1}$*, and* $\epsilon \in (0, 1/2)$*, in* $O(m\log^{O(1)} n \log(1/\epsilon))$ *time we can w.h.p. compute an* $\epsilon$*-approximate solution* $\tilde{\boldsymbol{x}}$ *to* $\mathbf{L}\boldsymbol{x} = \boldsymbol{b}$ *in the sense that* $\left\|\tilde{\boldsymbol{x}} - \mathbf{L}^\dagger \boldsymbol{b}\right\|_{\mathbf{U}_\mathbf{L}} \le \epsilon \left\|\mathbf{L}^\dagger \boldsymbol{b}\right\|_{\mathbf{U}_\mathbf{L}}$ *where* $\mathbf{U}_\mathbf{L} = (\mathbf{L}+\mathbf{L}^\top)/2$.

Combining this result with reductions proved in [CKP+16a], we immediately obtain nearly-linear running times (i.e., linear up to polylogarithmic factors in $n$, $m$, and a natural condition number-like quantity) for all of the following problems:

- solving row diagonally dominant (or column diagonally dominant) linear systems including arbitrary (non-Eulerian) directed Laplacian systems
- computing the stationary distribution of a Markov chain
- personalized PageRank
- obtaining polynomially good estimates of the mixing time of a Markov chain
- computing the hitting time from one vertex to another
- computing escape probabilities for any triple of vertices
- computing all-pairs commute times[5]

The core of the algorithm EulerianLU is a novel routine for unbiased, sparse approximation of Gaussian Elimination on Eulerian Laplacians. The routine preserves the null space of the approximate LU-decomposition, which essentially requires maintaining all vertex degrees exactly. A

4. For historical reasons, this lemma refers to the routine as Preconditioned Richardson, although it is more accurately understood as Iterative Refinement

5. If one wishes to compute commute times for a number of pairs greater than the number of edges in the graph, the runtime will be nearly-linear in the output size instead of the number of edges.

priori the existence of a unbiased, sparse, efficiently computable estimator that always maintains the correct nullspace exactly is rather surprising. The sampling we use to approximate the elimination process requires slightly more samples than in the case of undirected Laplacians. This means the matrix slowly grows dense and occasionally we need to sparsify all of the remaining matrix. Here "sparsify" means producing an approximate version of the matrix that is sparser than the input. We sparsify the matrix by making black-box use of the routine developed for this in [CKP+16a]. An informal sketch of the algorithm is stated below:

---

Randomly permute the vertices.
**for** $i = 1$ *to* $n - 1$ **do**
  Use equation $i$ to express the variable for vertex $i$ in terms of the remaining variables.
  Eliminate vertex $i$, adding random samples from the clique on the neighbors of $i$.
  **if** *The remaining matrix is too dense* **then**
    Sparsify the matrix.
  **end**
**end**

---

To prove Theorem 1.2.3, we also need several other components. Measured using a natural notion of ordering of positive semi-definite matrices, Schur complements of undirected Laplacians are always smaller than the original matrix. This is very important for bounding the variance of the approximate Gaussian Elimination routine in the proof of Theorem 1.2.1. An analogous fact is not true for Eulerian Laplacians: Some Schur complements grow very large compared to the original matrix. To control the variance of the elimination process regardless, we carefully choose the sets we eliminate, and show that for these special sets, the Schur complements are not growing too quickly. A technical contribution, but which may be of independent interest, is that we measure the approximation quality of the LU-decomposition w.r.t. a norm that is generated on the fly and depends on random choices made by the algorithm. We then show that Iterative Refinement using our LU-decomposition as a preconditioner will will converge measured in this execution-dependent norm – and eventually demonstrate that the final output is close in the usually preferred norm.

**Approximation of Laplacian Schur Complements and Sampling.** If we perform a partial LU-decomposition of an $n \times n$ matrix $\mathbf{M}$, eliminating only a subset of the variables $F \subset [n]$, the remaining matrix on the remaining subset of the variables $C = [n] \setminus F$ is referred to as the Schur complement of $\mathbf{M}$ onto $C$, which we denote by $\textsc{Sc}[\mathbf{M}]_C$. The Schur complement is introduced in Section 2.3.3. It can be shown that $\textsc{Sc}[\mathbf{M}]_C$ does not depend on the order in which the variables in $F$ are eliminated.

**Theorem 1.2.5** (Approximation of Laplacian Schur Complements)**.** *Suppose $\mathbf{U} \in \mathbb{R}^{n \times n}$ is a Laplacian matrix with $m$ non-zero entries. Given a set vertices $C \subset V$, and scalars $0 < \epsilon \leq 1/2$, $0 < \delta < 1$, the algorithm $\textsc{SchurApx}(\mathbf{L}, C, \epsilon, \delta)$ returns a Laplacian matrix $\widetilde{\mathbf{S}}$. With probability $\geq 1 - O(\delta)$, the following statements all hold: $\widetilde{\mathbf{S}} \approx_\epsilon \textsc{Sc}[\mathbf{U}]_C$. $\widetilde{\mathbf{S}}$ is a Laplacian matrix whose edges are supported on $C$. Let $k = |C| = n - |F|$. The total number of non-zero entries $\widetilde{\mathbf{S}}$ is $O(k\epsilon^{-2} \log(n/\delta))$. The total running time is bounded by $O((m \log n \log^2(n/\delta) + n\epsilon^{-2} \log n \log^4(n/\delta)) \operatorname{polyloglog}(n))$.*

The proof of Theorem 1.2.5 relies on combining techniques used to prove Theorem 1.2.1 with more stages of sparsification to ensure the output is sufficiently sparse, as well as combining both additive an multiplicative views of partial LU-decompositions to show that in fact spectral approximation between LU-decompositions of PSD matrices implies spectral approximation of their respective Schur complements.

**Sampling Random Spanning Trees.** In an unweighted graph, a uniformly random spanning tree is simply a spanning tree of the graph chosen uniformly among all its spanning trees. In a weighted undirected graph, the natural generalization of this distribution is to sample a tree with a probability that is proportional to the product of the weights of the edges in the tree. Restricted to integer weights, we can think of weights as counting multi-edges and letting each choice of multi-edge give rise to a distinct spanning tree. Given an undirected, weighted graph $G(V, E, w)$, we study the algorithmic task of sampling a random spanning tree from this distribution.

Schur complements of Laplacian matrices are closely related to the problem of sampling random spanning trees from undirected graphs. Schur complements can be used to measure the probability of an edge appearing in a random spanning tree. For example, in an unweighted graph, if an edge exists between vertices $i$ and $j$, then $\textsc{Sc}[\mathbf{U}]_{\{i,j\}}$ is the Laplacian of a single edge with some weight $w$, and $1/w$ is the marginal probability that edge $i, j$ appears in a random spanning tree of the graph.

We leverage this connection and the above result on Schur complement approximation to give the fastest known algorithm for sampling random trees in dense and weighted graphs. We give an algorithm for this problem, that, for a given $\delta > 0$, outputs a random spanning tree from this distribution with probability $1 - \delta$ in expected time $\widetilde{O}(n^{5/3}m^{1/3}\log^4(1/\delta))$.

**Theorem 1.2.6** (Sampling Random Spanning Trees)**.** *For any $0 < \delta < 1$, the routine* GenerateSpanningTree *(Algorithm 5) outputs a random spanning tree from the $\boldsymbol{w}$-uniform distribution with probability at least $1 - \delta$ and takes expected time $\widetilde{O}(\max\left\{n^{4/3}m^{1/2}, n^2\right\}\log^4(1/\delta))$.*

Beyond our tools for approximating Schur complements, the proof of Theorem 1.2.6 also requires a few other interesting ideas. The starting point for our algorithm is the $O(n^\omega)$ algorithm of Harvey and Xu [HX16] which has the same running time as the algorithm of Colburn et al. [CMN96], but is much simpler. The bottle-neck of this algorithm is the computation of Schur complements of Laplacians, and so we are able to accelerate it using Theorem 1.2.5. However, the algorithm requires us to sample a sequence of edges using probabilities calculated based on Schur complements. Introducing small errors in these probabilities could quickly make the sampling process diverge from the desired distribution. Instead, we use a powerful trick to sample from the exact distribution, by adaptively computing higher accuracy estimates of the probabilities when necessary.

## 1.3 Bibliographic Notes

The material presented in Chapter 3 is based on the paper [KS16] co-authored with Sushant Sachdeva, except for the section on Schur complement approximation, which is based on [DKP+16], co-authored with David Durfee, John Peebles, Anup B. Rao, and Sushant Sachdeva. The same paper is the source of the material in Chapter 4. The material in Chapter 5 is based on the still unpublished paper [CKK+17], co-authored with Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup Rao, and Aaron Sidford.

# Chapter 2

# Preliminaries

In this chapter, we review a number of central concepts that will be useful to reader of later on.

## 2.1 Linear Algebra

All of this dissertation concerns questions in theoretical computer science that draw heavily on linear algebra. Below we introduce some basic notation and definitions that will be used in later chapters.

**Upper and Lower Triangular Matrices.** We say a square matrix $\boldsymbol{\mathcal{U}}$ is upper triangular if it has non-zero entries $\boldsymbol{\mathcal{U}}(i,j) \neq 0$ only for $i \leq j$ (i.e. above the diagonal). Similarly, we say a square matrix $\boldsymbol{\mathcal{L}}$ is lower triangular if it has non-zero entries $\boldsymbol{\mathcal{U}}(i,j) \neq 0$ only for $i \geq j$ (i.e. below the diagonal). Often, we will work with matrices that are not upper or lower triangular, but which for we know a permutation matrix $\mathbf{P}$ s.t. $\mathbf{P}\boldsymbol{\mathcal{U}}\mathbf{P}^\top$ is upper (respectively lower) triangular. For computational purposes, this is essentially equivalent to having a upper or lower triangular matrix, and *we will refer to such matrices as upper (or lower) triangular*. The algorithms we develop for factorization will always compute the necessary permutation.

**Moore-Penrose Pseudo-inverse.** We use $\mathbf{B}^\dagger$ to denote the Moore-Penrose pseudo-inverse of a matrix $\mathbf{B}$.

**Positive Definite Matrices and Positive Semi-Definite Matrices.** We say a square matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is positive definite if for all $\boldsymbol{x} \in \mathbb{R}^n$ where $\boldsymbol{x} \neq \mathbf{0}$, we have $\boldsymbol{x}^\top \mathbf{M} \boldsymbol{x} > 0$. If for all $\boldsymbol{x} \in \mathbb{R}^n$ where $\boldsymbol{x} \neq \mathbf{0}$, we have $\boldsymbol{x}^\top \mathbf{M} \boldsymbol{x} \geq 0$, then $\mathbf{M}$ is positive semi-definite (PSD).

**Loewner Order.** We use $\preceq$ to denote a partial order on symmetric matrices, where $\mathbf{A} \preceq \mathbf{B}$ if and only $\mathbf{B} - \mathbf{A}$ is PSD.

**Restriction.** Given an $m \times n$ matrix $\mathbf{B}$, and index sets $F \subseteq [m]$, $C \subseteq [n]$, we use $(\mathbf{B})_{FC}$ to denote the $|F| \times |C|$ matrix obtained by restricting $\mathbf{B}$ to the rows $F$ and the columns $C$. When it does not cause ambiguity, we will sometimes omit the brackets and write $\mathbf{B}_{FC}$ to denote the matrix $(\mathbf{B})_{FC}$.

**Projection Matrix** Given matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$, let $\boldsymbol{\Pi}_{\mathbf{B}} \stackrel{\text{def}}{=} \mathbf{B}(\mathbf{B}\mathbf{B}^\top)^\dagger \mathbf{B}^\top$, i.e. the orthogonal projection onto the image of $\mathbf{B}$.Note that $\boldsymbol{\Pi}_{\mathbf{B}} = \boldsymbol{\Pi}_{\mathbf{B}}^\top$ and $\boldsymbol{\Pi}_{\mathbf{B}} = \boldsymbol{\Pi}_{\mathbf{B}}^2$.

**Norms from Quadratic Forms.** For any PSD matrix $\mathbf{M}$ and any vector $\boldsymbol{v}$, define the semi-norm $\|\boldsymbol{v}\|_{\mathbf{M}} \stackrel{\text{def}}{=} \sqrt{\boldsymbol{v}^\top \mathbf{M} \boldsymbol{v}}$.

**Kernel and Cokernel.** Given a matrix $\mathbf{M}$, we use $\ker(\mathbf{M})$ to denote the kernel of $\mathbf{M}$, i.e. the subspace of vectors $\boldsymbol{x}$ s.t. $\mathbf{M}\boldsymbol{x} = \mathbf{0}$. The term cokernel refers to the kernel of $\mathbf{M}^\top$.

**Undirectification (Arithmetic Symmetrization)** Given square matrix $\mathbf{A}$, define $\mathbf{U_A} \stackrel{\text{def}}{=} \frac{\mathbf{A}+\mathbf{A}^\top}{2}$. We refer to $\mathbf{U_A}$ as the undirectification (or the arithmetic symmetrization) of $\mathbf{A}$.

Let $\mathbf{1} \in \mathbb{R}^n$ denote the all ones vector, with dimension $n$ that will always be made clear in the context of its use. Similarly, we let $\mathbf{0}$ denote the all zero vector or matrix, depending on context.

The following fact is useful, since we often need to apply the pseodo-inverse of a matrix.

**Fact 2.1.1** (Pseudo-inverse of a product)**.** *Suppose* $\mathbf{M} = \mathbf{ABC}$ *is square real matrix, where* $\mathbf{A}$ *and* $\mathbf{C}$ *are non-singular. Then*

$$\mathbf{M}^\dagger = \mathbf{\Pi_M C^{-1} B^\dagger A^{-1} \Pi_{M^\top}}.$$

**Definition 2.1.2** (PSD Spectral Approximation)**.** *Given two PSD matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *and a scalar* $\epsilon \geq 0$*, we say* $\mathbf{A} \approx_\epsilon \mathbf{B}$ *if and only if*

$$\exp(-\epsilon)\mathbf{A} \preceq \mathbf{B} \preceq \exp(\epsilon)\mathbf{A}.$$

*We say* $\mathbf{A}$ *is an* $\epsilon$*-approximation of* $\mathbf{B}$*.*

**Definition 2.1.3** (Asymmetric Spectral Approximation)**.** *Consider two square asymmetric matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, s.t.* $\mathbf{A}$ *has kernel equal to its co-kernel and the undirectification* $\mathbf{U_A}$ *of* $\mathbf{A}$ *is PSD. Given a scalar* $\epsilon \geq 0$*, we say* $\mathbf{B}$ *is an* $\epsilon$*-asymmetric spectral approximation of* $\mathbf{A}$ *if and only if*

$$\left\| (\mathbf{U_A^\dagger})^{1/2} (\mathbf{A} - \mathbf{B}) (\mathbf{U_A^\dagger})^{1/2} \right\| \leq \epsilon.$$

## 2.2 Matrix Classes

In this section, we introduce several families of matrices. Each is significant, because we are able to construct fast linear system solvers for matrices in these classes.

**Symmetric Diagonally Dominant (SDD) Matrices.** A matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is said to be Symmetric Diagonally Dominant (SDD), if it is symmetric and for each row $i$,

$$\mathbf{M}(i,i) \geq \sum_{j \neq i} |\mathbf{M}(i,j)|. \tag{2.1}$$

It can be shown that every SDD matrix is positive semi-definite. .

**Undirected Laplacians, a.k.a. Laplacians.** We consider a connected undirected multi-graph $G = (V, E)$, with positive edges weights $w : E \to \mathbb{R}_+$. Let $n = |V|$ and $m = |E|$. We label vertices 1 through $n$, s.t. $V = \{1, \ldots, n\}$. Let $\boldsymbol{\chi}_i$ denote the $i^{\text{th}}$ standard basis vector. Given an ordered pair of vertices $(u, v)$, we define the pair-vector $b_{u,v} \in \mathbb{R}^n$ as $\boldsymbol{b}_{u,v} = \boldsymbol{\chi}_v - \boldsymbol{\chi}_u$. For a multi-edge $e$, with endpoints $u, v$ (arbitrarily ordered), we define $\boldsymbol{b}_e = \boldsymbol{b}_{u,v}$.

By assigning an arbitrary direction to each multi-edge of $G$ we define the Laplacian of $G$ as $\mathbf{U} = \sum_{e \in E} w(e) \boldsymbol{b}_e \boldsymbol{b}_e^\top$. Note that the Laplacian does not depend on the choice of direction for each edge. Given a single multi-edge $e$, we refer to $w(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$ as the Laplacian of $e$.

A weighted multi-graph $G$ is not uniquely defined by its Laplacian, since the Laplacian only depends on the sum of the weights of the multi-edges on each edge. We want to establish a one-to-one correspondence between a weighted multi-graph $G$ and its Laplacian $\mathbf{U}$, so from now on, we will consider every Laplacian to be maintained explicitly as a sum of Laplacians of multi-edges, and we will maintain this multi-edge decomposition as part of our algorithms.

**Fact 2.2.1.** *If $G$ is connected, then the kernel of the corresponding Laplacian $\mathbf{U}$ is the span of the vector $\mathbf{1}$.*

**Directed Laplacians.** A matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$ is called a *directed Laplacian* if (1) its off diagonal entries are non-positive, i.e. $\mathbf{L}(i,j) \leq 0$ for all $i \neq j$, and (2) it satisfies $\mathbf{1}^\top \mathbf{L} = \mathbf{0}$, i.e. $\mathbf{L}(i,i) = -\sum_{j \neq i} \mathbf{L}(j,i)$ for all $i$.

**Associated Graph.** To every directed Laplacian $\mathbf{L} \in \mathbb{R}^{n \times n}$ we associate a graph $G_{\mathbf{L}} = (V, E, w)$ with vertices $V = [n]$, and edges $(i,j)$ of weight $w(i,j) = -\mathbf{L}(j,i)$, for all $i \neq j \in [n]$ with $\mathbf{L}(j,i) \neq 0$. Occasionally we write $\mathbf{L} = \mathbf{D} - \mathbf{A}^\top$ to denote that we decompose $\mathbf{L}$ into the diagonal matrix $\mathbf{D}$ (where $\mathbf{D}(i,i) = \mathbf{L}(i,i)$ is the out degree of vertex $i$ in $G_{\mathbf{L}}$) and non-negative matrix $\mathbf{A}$ (which is weighted adjacency matrix of $G_{\mathbf{L}}$, with $\mathbf{A}(i,j) = w(i,j)$ if $(i,j) \in E$, and $\mathbf{A}(i,j) = 0$ otherwise).

**Eulerian Laplacian.** A matrix $\mathbf{L}$ is called an *Eulerian Laplacian* if it is a directed Laplacian with $\mathbf{L1} = \mathbf{0}$. Note that $\mathbf{L}$ is an *Eulerian Laplacian* if and only if its associated graph is Eulerian.

Below we introduce a special notion of well-behaved subsets of vertices of a Directed Laplacian: $\alpha$-Row and Column Diagonally Dominant sets (abbreviated $\alpha$-RCDD). By eliminating vertices from these sets, we are able to reduce the .

**Definition 2.2.2** ($\alpha$-RCDD)**.** *Given a subset $F$ of the vertices of (the graph of) a Directed Laplacian $\mathbf{L}$, we say a vertex $i \in F$ is $\alpha$-RCDD if $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \leq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ and $\sum_{j \in F, j \neq i} |\mathbf{L}_{ji}| \leq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$. The set $F$ is $\alpha$-RCDD if every vertex in the set is $\alpha$-RCDD.*

## 2.3 Solutions to Linear Equations

When approximately solving a linear system of equations $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$, say, over the reals, it is important to pick a useful notion of approximation.

For Laplacian solvers, the approximation error of an approximate solution $\boldsymbol{x}$ to a system $\mathbf{U}\boldsymbol{x} = \boldsymbol{b}$ is measured by the $\epsilon$ s.t.

$$\left\| \tilde{\boldsymbol{x}} - \mathbf{U}^+ \boldsymbol{b} \right\|_{\mathbf{U}} \leq \epsilon \left\| \mathbf{U}^+ \boldsymbol{b} \right\|_{\mathbf{U}}.$$

In most applications, this is a directly useful notion of approximation. Fortunately, the $\log(1/\epsilon)$ running time dependence of on the approximation parameter $\epsilon$ also means that approximations in many other norms can be guaranteed by solving to high accuracy in the norm stated above and then proving some weak relationship between different notions of error.

For Eulerian Laplacians, we use a similar notion of error for the system $\mathbf{L}\boldsymbol{x} = \boldsymbol{b}$,

$$\left\| x - \mathbf{L}^\dagger b \right\|_{\mathbf{U_L}} \leq \epsilon \left\| \mathbf{L}^\dagger b \right\|_{\mathbf{U_L}}$$

where $\mathbf{U_L} = (\mathbf{L} + \mathbf{L}^\top)/2$. For an Eulerian Laplacian $\mathbf{L}$, the matrix $\mathbf{U_L}$ is always an undirected Laplacian.

### 2.3.1 Iterative Methods and Preconditioners

In this dissertation, we study algorithms for obtaining approximate, but highly accurate, solutions to systems of linear equations in a given matrix. A central framework when developing such algorithms is iterative methods. The convergence rate of most iterative methods inherently depends on the condition number of the matrix in which a linear system is being solved.

A wide range of iterative algorithms exist, from the very simplest, known as method Iterative Refinement and its close cousin, Richardson Iteration, to Chebyshev Iteration and Conjugate Gradient Descent, significantly more complex methods that achieve better dependence on the condition number.

While most iterative methods have been designed to solve systems of linear equations in positive semi-definite matrices, Iterative Refinement can also be shown to converge for other matrices, provided the input matrix is close to the identity matrix in a strong sense.

A key tool in iterative methods is *preconditioners*. Given a linear equation $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ in a square matrix $\mathbf{A}$, a preconditioner for $\mathbf{A}$ is a matrix that in some sense approximates $\mathbf{A}$, but is easier to invert. In iterative methods, if we can compute a preconditioner for the matrix, we usually replace the condition number dependence in the running time of the iterative method with a dependence on the approximation quality between $\mathbf{A}$ and the preconditioner, at the cost of having to apply the inverse of the preconditioner.

Throughout this dissertation, the only iterative method we will use is Iterative Refinement. To use Iterative Refinement to build fast algorithms for solving linear equations, we will need to construct extremely high quality preconditioners. This approach has the advantage of extending beyond the setting of positive semi-definite and even symmetric matrices, which will be crucial in Chapter 5, where we construct nearly-linear time solvers for linear systems in Directed Laplacians, a class of matrices which is neither positive semi-definite nor symmetric.

### 2.3.2 Preconditioned Iterative Refinement for Positive Semi-Definite Matrices

We briefly introduce Preconditioned Iterative Refinement [Hig02, Chapter 12] to solve the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, where $\mathbf{A} \in \mathbb{C}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{C}^n$. Suppose $\mathbf{Z}$ is a preconditioner for $\mathbf{A}$. Preconditioned Iterative Refinement refers to the procedure which computes the iterates below.

$$\boldsymbol{x}^{(0)} = \mathbf{0}, \qquad \boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - \mathbf{Z}^{\dagger}(\mathbf{A}\boldsymbol{x}^{(i)} - \boldsymbol{b}),$$

We use $\textsc{PreconIterRefinement}(\mathbf{U}, \mathbf{Z}, \epsilon, \boldsymbol{b})$ to denote the routine which performs preconditioned iteration refinement as described above, to compute and return $\boldsymbol{x}^{(t)}$, with $t = 3\log\frac{1}{\epsilon}$.

**Theorem 2.3.1.** *Consider a positive semi-definite matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ and a vector $\boldsymbol{b} \in \mathbb{R}^n$. Suppose $\mathbf{Z}$ is a linear operator s.t. $\mathbf{Z} \approx_{1/2} \mathbf{A}$. Then for all $0 < \epsilon \leq 1/2$, $\textsc{PreconIterRefinement}(\mathbf{A}, \mathbf{Z}, \epsilon, \boldsymbol{b})$ returns $\boldsymbol{x}^{(t)}$, where $t = \lceil 3\log\frac{1}{\epsilon} \rceil$, s.t. $\left\|\boldsymbol{x}^{(t)} - \mathbf{A}^{\dagger}\boldsymbol{b}\right\|_{\mathbf{A}} \leq \epsilon \left\|\mathbf{A}^{\dagger}\boldsymbol{b}\right\|_{\mathbf{A}}$.*

### 2.3.3 Schur Complements

Gaussian Elimination is a classical algorithm for solving systems of linear equations. It is closely related to the notion of Schur Complements, which we introduce in this section. Suppose $\mathbf{M} \in \mathbb{C}^{n\times n}$ is a square matrix and $F, C \subset [n]$ is a partition of $[n]$ into two sets. W.l.o.g. taking $F$ to be the first $|F|$ indices of $[n]$, we can write

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{F,F} & \mathbf{M}_{F,C} \\ \mathbf{M}_{C,F} & \mathbf{M}_{C,C} \end{pmatrix}.$$

When $\mathbf{M}_{F,F}$ is invertible, we define the Schur complement of $\mathbf{M}$ onto $C$ as

$$\textsc{Sc}[\mathbf{M}]_C \stackrel{\text{def}}{=} \mathbf{M}_{C,C} - \mathbf{M}_{C,F}(\mathbf{M}_{F,F})^{-1}\mathbf{M}_{F,C}.$$

The Schur complement is related to inverses and the LDU decompositions of a matrix. One way to see this is through the blockwise LDU decomposition of a matrix

$$\mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{M}_{C,F}\mathbf{M}_{F,F}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{M}_{F,F} & \mathbf{0} \\ \mathbf{0} & \textsc{Sc}[\mathbf{M}]_C \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{M}_{F,F}^{-1}\mathbf{M}_{F,C} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

This also implies that when $\mathbf{M}$ is invertible,

$$\begin{aligned} \mathbf{M}^{-1} &= \begin{pmatrix} \mathbf{I} & -\mathbf{M}_{F,F}^{-1}\mathbf{M}_{F,C} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{M}_{F,F}^{-1} & \mathbf{0} \\ \mathbf{0} & \textsc{Sc}[\mathbf{M}]_C^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{M}_{C,F}\mathbf{M}_{F,F}^{-1} & \mathbf{I} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{M}_{F,F}^{-1} + \mathbf{M}_{F,F}^{-1}\mathbf{M}_{F,C}\textsc{Sc}[\mathbf{M}]_C^{-1}\mathbf{M}_{C,F}\mathbf{M}_{F,F}^{-1} & -\mathbf{M}_{F,F}^{-1}\mathbf{M}_{F,C}\textsc{Sc}[\mathbf{M}]_C^{-1} \\ -\textsc{Sc}[\mathbf{M}]_C^{-1}\mathbf{M}_{C,F}\mathbf{M}_{F,F}^{-1} & \textsc{Sc}[\mathbf{M}]_C^{-1} \end{pmatrix}. \end{aligned}$$

This in turn implies immediately that

$$(\mathbf{M}^{-1})_{C,C} = \textsc{Sc}[\mathbf{M}]_C^{-1}.$$

Suppose that $C_2 \subseteq C_1 \subseteq [n]$. Then

$$\textsc{Sc}[\mathbf{M}]_{C_2} = \textsc{Sc}\left[\textsc{Sc}[\mathbf{M}]_{C_1}\right]_{C_2}.$$

This follows from noting that $(\mathbf{M}^{-1})_{C_2,C_2} = ((\mathbf{M}^{-1})_{C_1,C_1})_{C_2,C_2}$ and then inverting both matrices.

This in turn tells us that we can compute Schur Complements in stepwise manner, first Schur complementing onto $n-1$ variables (eliminating only one variable), then Schur complementing onto $n-2$ of the remaining variables etc.

The blockwise LDU decomposition can also be written as a sum of the zero-padded Schur complement and a term that agrees with $\mathbf{M}$ on all except the $\mathbf{M}_{C,C}$ block.

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{F,F} & \mathbf{M}_{F,C} \\ \mathbf{M}_{C,F} & \mathbf{M}_{C,F}\mathbf{M}_{F,F}^{-1}\mathbf{M}_{F,C} \end{pmatrix} + \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \textsc{Sc}[\mathbf{M}]_C \end{pmatrix}$$

**Fact 2.3.2** (Schur Complements and Pseudo-Inverses)**.** *Suppose* $\mathbf{U}$ *is an undirected Laplacian. Let* $\mathbf{S} = \textsc{Sc}[\mathbf{U}]_C^{\dagger}$*. Then* $\mathbf{\Pi_S}(\mathbf{U}^{\dagger})_{C,C}\mathbf{\Pi_S} = \textsc{Sc}[\mathbf{U}]_C^{\dagger}$*.*

We prove this fact in Appendix Section A.1.

**Claim 2.3.3.** *Consider* $n \times n$ *Laplacians* $\mathbf{A}, \mathbf{B} \succeq \mathbf{0}$*, s.t. for some* $0 < \epsilon < 1$ *we have* $\mathbf{A} \approx_\epsilon \mathbf{B}$*. Let* $F \subseteq [n]$*. Then*

$$\textsc{Sc}[\mathbf{A}]_F \approx_\epsilon \textsc{Sc}[\mathbf{B}]_F.$$

*Proof.* This follows from Fact 2.3.2, and the fact that restriction and inversion preserve spectral approximation. . □

### 2.3.4 Gaussian Elimination, Cholesky Factorization, and LU-decomposition

An LU-decomposition of a square matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a factorization $\mathbf{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$, where $\boldsymbol{\mathcal{L}}$ is a lower-triangular matrix and $\boldsymbol{\mathcal{U}}$ is an upper-triangular matrix, both up to a permutation (see Section 5).

When $\mathbf{M}$ is non-singular, linear equations $\boldsymbol{\mathcal{L}}\boldsymbol{y} = \boldsymbol{b}$ and $\boldsymbol{\mathcal{U}}\boldsymbol{x} = \boldsymbol{y}$ and be solved by forward and backward substitution algorithms respectively (e.g. see [TBI97]), which run in time $O(\text{nnz}(\boldsymbol{\mathcal{L}}))$ and $O(\text{nnz}(\boldsymbol{\mathcal{U}}))$. I.e. $\boldsymbol{\mathcal{L}}^{-1}$ and $\boldsymbol{\mathcal{U}}^{-1}$ can be applied in time proportional to the number of non-zeros in $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ respectively. This means that if a decomposition $\boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{U}}$ of $\mathbf{M}$ is known, then linear systems in $\mathbf{M}$ be solved in time $O(\text{nnz}(\boldsymbol{\mathcal{L}}) + \text{nnz}(\boldsymbol{\mathcal{U}}))$, since $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$ implies $\boldsymbol{x} = \boldsymbol{\mathcal{U}}^{-1}\boldsymbol{\mathcal{L}}^{-1}\boldsymbol{b}$.

When $\mathbf{M}$ is singular the same forward and backward substitution algorithms can be used to compute the pseudo-inverse, in some situations. We focus on a special case where we can instead factor $\mathbf{M}$ as $\mathbf{M} = \boldsymbol{\mathcal{L}}'\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{U}}'$, where $\boldsymbol{\mathcal{D}}$ is singular and $\boldsymbol{\mathcal{L}}', \boldsymbol{\mathcal{U}}'$ are non-singular.

For the two cases of interest to us, Undirected Laplacians and Eulerian Laplacians of connected graphs, this slightly modified factorization can be trivially obtained from an LU-decomposition, by $\boldsymbol{\mathcal{L}}'$ equal to $\boldsymbol{\mathcal{L}}$ except $\boldsymbol{\mathcal{L}}(n,n) = 1$ and $\boldsymbol{\mathcal{U}}'$ equal to $\boldsymbol{\mathcal{U}}$ except $\boldsymbol{\mathcal{U}}(n,n) = 1$ and taking $\boldsymbol{\mathcal{D}}$ to be the identity matrix, except $\boldsymbol{\mathcal{D}}(n,n) = 0$.

Now, by Fact 2.1.1,

$$\mathbf{M}^{\dagger} = \boldsymbol{\Pi}_{\mathbf{M}}\boldsymbol{\mathcal{L}}'^{-1}\boldsymbol{\mathcal{D}}^{\dagger}\boldsymbol{\mathcal{U}}'^{-1}\boldsymbol{\Pi}_{\mathbf{M}^{\top}}.$$

For connected Undirected and strongly connected Eulerian Laplacians, the kernel and co-kernel are always the span of the $\mathbf{1}$ vector, so we can apply $\boldsymbol{\Pi}_{\mathbf{M}}$ and $\boldsymbol{\Pi}_{\mathbf{M}^{\top}}$ efficiently.

Gaussian Elimination is an algorithm that can be used to compute an LU-decomposition of some matrices. Gaussian Elimination proceeds by writing $\mathbf{M}$ as the sum of a rank 1 term that agrees with $\mathbf{M}$ on the first row and column and a (zero-padded) Schur complement:

As a convenient notational convention, we define $\mathbf{S}^{(0)} \stackrel{\text{def}}{=} \mathbf{M}$, the "0th" Schur complement. We then define

$$\begin{aligned}
\boldsymbol{l}_i &= (\mathbf{S}^{(i-1)}(i,i)^{1/2})^{-1}\mathbf{S}^{(i-1)}(:,i) \\
\boldsymbol{u}_i^{\top} &= (\mathbf{S}^{(i-1)}(i,i)^{1/2})^{-1}\mathbf{S}^{(i-1)}(i,:) \\
\mathbf{S}^{(i)} &= \mathbf{S}^{(i-1)} - \boldsymbol{l}_i\boldsymbol{u}_i^{\top},
\end{aligned} \tag{2.2}$$

unless $\mathbf{S}^{(i-1)}(i,i) = 0$, in which case we define $\mathbf{S}^{(i)} = \mathbf{S}^{(i-1)}$, if $\mathbf{S}^{(i-1)}(:,i) = \mathbf{0}$ and $\mathbf{S}^{(i-1)}(i,:) = \mathbf{0}$. We do not define the Schur complement when the diagonal is zero but off-diagonals are not. By setting

$$\boldsymbol{\mathcal{L}} = \begin{pmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_n \end{pmatrix}$$

and

$$\boldsymbol{\mathcal{U}}^{\top} = \begin{pmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_n \end{pmatrix}$$

we get an LU-decomposition $\mathbf{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$.

It can be shown that Gaussian Elimination as described above always produces an LU-decomposition when applied an Undirected Laplacian or Eulerian Laplacian. For some other classes of matrices, it does not always succeed (e.g. if the first column has a zero diagonal but other non-zero entries).

When $\mathbf{M}$ is a positive semi-definite matrix, it can be decomposed as $\mathbf{M} = \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^{\top}$, i.e. an LU decomposition where the upper triangular matrix is the transpose of the lower triangular matrix. This special case is known as a *Cholesky factorization*. When the Gaussian Elimination algorithm described above is applied to a Laplacian matrix it computes the Cholesky factorization.

**Definition 2.3.4** (Partial LU-decomposition and Partial Cholesky Factorization)**.** *If Gaussian Elimination is terminated after $k$ steps (see Equation (2.2)), we get a decomposition where*

$$\boldsymbol{\mathcal{L}} = \begin{pmatrix} \boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_k \end{pmatrix}$$

*and*

$$\boldsymbol{\mathcal{U}}^\top = \begin{pmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_k \end{pmatrix}$$

*s.t.* $\mathbf{M} = \mathbf{S}^{(k)} + \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}}$*, where* $\mathbf{S}^{(k)}$ *is zero except on indices in* $C \times C$ *where* $C = \{k+1, \ldots, n\}$ *and* $(\mathbf{S}^{(k)})_{C,C} = \textsc{Sc}[\mathbf{M}]_C$*. Letting* $F = [n] \setminus C$*, this decomposition can also be written as*

$$\mathbf{M} = \begin{pmatrix} \boldsymbol{\mathcal{L}}_{C,F} & \mathbf{0} \\ \boldsymbol{\mathcal{L}}_{C,F} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & (\mathbf{S}^{(k)})_{C,C} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \boldsymbol{\mathcal{U}}_{F,C} \\ \mathbf{0} & \boldsymbol{\mathcal{U}}_{C,C} \end{pmatrix}$$

*We refer to* $\boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{U}}, \mathbf{S}$ *as a* partial LU-decomposition *to set* $C$*, and when* $\mathbf{M}$ *is positive semi-definite, so* $\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{U}}^\top$*, we refer to* $\boldsymbol{\mathcal{L}}, \mathbf{S}$ *as a* partial Cholesky factorization *to set* $C$*.*

**Definition 2.3.5** (Approximate Cholesky Factorization and Partial Cholesky Factorization)**.** *We refer to* $\boldsymbol{\mathcal{L}}$ *as an* $\epsilon$-approximate Cholesky factorization *of a matrix* $\mathbf{M}$ *if* $\boldsymbol{\mathcal{L}}$ *is a Cholesky factorization of a matrix* $\widetilde{\mathbf{M}}$ *s.t.* $\widetilde{\mathbf{M}} \approx_\epsilon \mathbf{M}$*.*

*We refer to* $\boldsymbol{\mathcal{L}}, \mathbf{S}$ *as an* $\epsilon$-approximate partial Cholesky factorization *to a set* $C$ *of a matrix* $\mathbf{M}$ *if* $\boldsymbol{\mathcal{L}}, \mathbf{S}$ *is a partial Cholesky factorization to a set* $C$ *of a matrix* $\widetilde{\mathbf{M}}$ *s.t.* $\widetilde{\mathbf{M}} \approx_\epsilon \mathbf{M}$*.*

### 2.3.5 Schur Complements and Closure

Some classes of matrices have the property that if $\mathbf{M}$ is a matrix in the class, then for any subset $C$ of the indices of $\mathbf{M}$, the Schur complement $\textsc{Sc}[\mathbf{M}]_C$ is also in that class.

This observation, combined with the fact that some classes of matrices can be well-approximated with sparse matrices of the same class, is at the core of all the fast algorithms for solving systems of linear equations that we develop in this dissertation.

Positive definite and positive semi-definite matrices both have the property of being closed under Schur complement, i.e. the Schur complement of a positive definite matrix is a positive definite matrix and the Schur complement of a positive semi-definite matrix is positive semi-definite.

Below, we state the closure property for the matrix classes that we use in this dissertation.

**Fact 2.3.6.** *For each of the following classes, it holds that if* $\mathbf{M}$ *is a matrix of this class with index set* $[n]$*, then for all* $C \subseteq [n]$*,* $\textsc{Sc}[\mathbf{U}]_C$ *is a matrix of the same class.*

- *Positive Definite Matrices.*
- *Positive Semi-Definite Matrices.*
- *Undirected Laplacians.*
- *Eulerian Laplacians.*

### 2.3.6 The Clique Structure of Schur Complements

In the previous section, we introduced Schur complement closure properties for several classes of matrices. In this section, we show how to prove the closure property for Schur complements of Laplacians, while also observing that the Schur complement of a Laplacian onto $n-1$ indices has

additional structure that will help us develop algorithms for approximating these Schur complements.

Similar structure can be found in the Schur complements of Eulerian Laplacians, but we defer the proof of this to Chapter 5.

Given a Laplacian $\mathbf{U}$, let $\textsc{St}[\mathbf{U}]_v \in \mathbb{R}^{n\times n}$ denote the Laplacian corresponding to the edges incident on vertex $v$ (the *star* on $v$), i.e.

$$\textsc{St}[\mathbf{U}]_v \stackrel{\text{def}}{=} \sum_{e\in E: e\ni v} w(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top. \tag{2.3}$$

For example, we denote the first column of $\mathbf{U}$ by $\begin{pmatrix} d \\ -\boldsymbol{a}\end{pmatrix}$, then $\textsc{St}[\mathbf{U}]_1 = \begin{bmatrix} d & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a})\end{bmatrix}$. We can write the Schur complement $\mathbf{S} = \textsc{Sc}[\mathbf{U}]_{[n]\setminus\{v_1\}}$.

$$\mathbf{S} = \mathbf{U} - \textsc{St}[\mathbf{U}]_{v_1} + \textsc{St}[\mathbf{U}]_{v_1} - \frac{1}{\mathbf{U}(v_1,v_1)}\mathbf{U}(:,v_1)\mathbf{U}(v_1,:).$$

It is immediate that $\mathbf{U} - \textsc{St}[\mathbf{U}]_{v_1}$ is a Laplacian matrix, since $\mathbf{U} - \textsc{St}[\mathbf{U}]_{v_1} = \sum_{e\in E: e\not\ni v_1} w(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$. A more surprising (but well-known) fact is that

$$\textsc{Cl}[\mathbf{U}]_{v_1} \stackrel{\text{def}}{=} \textsc{St}[\mathbf{U}]_{v_1} - \frac{1}{\mathbf{U}(v_1,v_1)}\mathbf{U}(:,v_1)\mathbf{U}(v_1,:) \tag{2.4}$$

is also a Laplacian, and its edges form a clique on the neighbors of $v_1$. It suffices to show it for $v_1 = 1$. We write $i \sim j$ to denote $(i,j) \in E$. Then

$$\textsc{Cl}[\mathbf{U}]_1 = \mathbf{U}_1 - \frac{1}{\mathbf{U}(1,1)}\mathbf{U}(:,1)\mathbf{U}(:,1)^\top = \begin{bmatrix}\mathbf{0} & \mathbf{0}^\top \\ \mathbf{0} & \operatorname{diag}(\boldsymbol{a}) - \frac{\boldsymbol{a}\boldsymbol{a}^\top}{d}\end{bmatrix} = \sum_{i\sim 1}\sum_{j\sim 1}\frac{w(1,i)w(1,j)}{d}\boldsymbol{b}_{(i,j)}\boldsymbol{b}_{(i,j)}^\top.$$

Thus $\mathbf{S}$ is a Laplacian since it is a sum of two Laplacians. By induction, for all $C \subseteq [n]$, $\textsc{Sc}[\mathbf{U}]_C$ is a Laplacian.

## 2.4 Spanning Trees of Graphs

We assume we are given a weighted undirected graph $G = (V, E, \boldsymbol{w})$, with the vertices labeled $V = \{1, 2, ..., n\}$. Let $\mathbf{U}$ be the associated Laplacian.

**Definition 2.4.1** (Induced Graph)**.** *Given a graph $G = (V, E)$ and a set of vertices $V_1 \subseteq V$, we use the notation $G(V_1)$ to mean the induced graph on $V_1$.*

**Definition 2.4.2.** *Given a set of edges $E$ on vertices $V$, and $V_1, V_2 \subseteq V$, we use the notation $E \cap (V_1, V_2)$ to mean the set of all edges in $E$ with one end point in $V_1$ and the other in $V_2$.*

**Definition 2.4.3** (Contraction and Deletion)**.** *Given a graph $G = (V, E)$ and a set of edges $E_1 \subset E$, we use the notation $G\setminus E_1$ to denote the graph obtained by deleting the edges in $E_1$ from $G$ and $G/E_1$ to denote the graph obtained by contracting the edges in $E_1$ within $G$ and deleting all the self loops.*

### 2.4.1 Spanning Trees

Let $\mathcal{T}_G$ denote the set of all spanning subtrees of $G$. We now define a probability distribution on these trees.

**Definition 2.4.4** ($\boldsymbol{w}$-uniform distribution on trees)**.** *Let $\mathcal{D}_G$ be the probability distribution on $\mathcal{T}_G$ such that*

$$\Pr(X = T) \propto \Pi_{e \in T} \boldsymbol{w}_e.$$

*We refer to $\mathcal{D}_G$ as the $\boldsymbol{w}$-uniform distribution on $\mathcal{T}_G$. When the graph $G$ is unweighted, this corresponds to the uniform distribution on $\mathcal{T}_G$.*

**Definition 2.4.5** (Effective Resistance)**.** *The effective resistance of a pair of vertices $u, v \in V_G$ is defined as*

$$R_{eff}(u, v) = \boldsymbol{b}_{u,v}^T \mathbf{L}^\dagger \boldsymbol{b}_{u,v}.$$

*where $\boldsymbol{b}_{u,v}$ is an all zero vector corresponding to $V_G$, except for entries of $1$ and $-1$ at $u$ and $v$*

**Definition 2.4.6** (Leverage Score)**.** *The statistical leverage score, which we will abbreviate to leverage score, of an edge $e = (u, v) \in E_G$ is defined as*

$$l_e = \boldsymbol{w}_e R_{eff}(u, v).$$

**Fact 2.4.7** (Spanning Tree Marginals)**.** *The probability $\Pr(e)$ that an edge $e \in E_G$ appears in a tree sampled $\boldsymbol{w}$-uniformly randomly from $\mathcal{T}_G$ is given by*

$$\Pr(e) = l_e,$$

*where $l_e$ is the leverage score of the edge $e$.*

### 2.4.2 Schur Complements and Spanning Trees

When the matrix $\mathbf{M} = \mathbf{U}$ is a Laplacian of a graph $G = (V, E)$ and $V_1 \subseteq V$ is a set of vertices, we abuse the notation and use both $\textsc{Sc}[\mathbf{U}]_{V_1}$ or $\textsc{Sc}[G]_{V_1}$ to denote the Schur complement of $\mathbf{U}$ onto the submatrix of $\mathbf{U}$ corresponding to $V_1$; i.e., onto the submatrix of $\mathbf{U}$ consisting of all entries whose coordinates $(i, j)$ satisfy $i, j \in V_1$.

Recall that by Fact 2.3.6, the Schur complement of a Laplacian is a Laplacian. This means that the Schur complement in a graph $G = (V, E)$ onto a set of vertices $V_1$ can be viewed as a graph on $V_1$. Furthermore, we can view this as a multigraph obtained by adding (potentially parallel) edges to $G(V_1)$, the induced graph on $V_1$. We take this view when working with spanning trees: whenever we talk about Schur complements, we separate out the edges of the original graph from the ones created during Schur complement operation.

We now provide some basic facts about how Schur complements relate to spanning trees. This first lemma says that edge deletions and contractions commute with taking Schur complements.

**Fact 2.4.8.** *(Lemma 4.1 of [CDN89]) Given $G$ with any vertex partition $V_1, V_2$, for any edge $e \in E \cap (V_1, V_1)$.*

$$\textsc{Sc}[G \setminus e]_{V_1} = \textsc{Sc}[G]_{V_1} \setminus e \qquad \text{and} \qquad \textsc{Sc}[G/e]_{V_1} = \textsc{Sc}[G]_{V_1/e}$$

**Fact 2.4.9.** *Given $G$ with any vertex partition $V_1, V_2$, for any edge $e \in E \cap (V_1, V_1)$, the leverage score of $e$ in $G$ is same as that in $\textsc{Sc}[G]_{V_1}$.*

*Proof.* This follows immediately from Definition 2.4.6 and Fact 2.3.2.

□

# Chapter 3

# Approximate Gaussian Elimination

Algorithm 1 gives the pseudo-code for our algorithm MasterCholApx our algorithm for computing approximate Cholesky factorizations using approximate Gaussian Elimination. In Section 3.1, we state the MasterCholApx algorithm and prove its correctness. This algorithm is has a large number of input parameters, and can be used to compute both complete and partial approximate Cholesky factorizations, of high or low approximation quality. We later show how choosing these parameters appropriately can give algorithms for

- solving Laplacian linear systems (see proofs Theorem 1.2.1 and Corollary 1.2.2 in Section 3.2),
- solving Laplacian linear systems faster in dense graphs (see proof of Theorem 3.3.1 in Section 3.3),
- and for approximating Laplacian Schur complements (proof of Theorem 1.2.5 in Section 3.4).

### Normalized Matrices and $c$-Boundedness

We will very frequently need to refer to matrices that are normalized by some positive semi-definite matrix $\mathbf{U}$. We adopt the following notation: Given a symmetric matrix $\mathbf{S}$ s.t. $\ker(\mathbf{U}) \subseteq \ker(\mathbf{S})$,

$$\overline{\mathbf{S}} \stackrel{\text{def}}{=} (\mathbf{U}^+)^{1/2}\mathbf{S}(\mathbf{U}^+)^{1/2}.$$

We will only use this notation for matrices $\mathbf{S}$ that satisfy the condition $\ker(\mathbf{U}) \subseteq \ker(\mathbf{S})$. Note that $\overline{\mathbf{U}} = \mathbf{\Pi}_{\mathbf{U}}$ and $\mathbf{A} \preceq \mathbf{B}$ iff $\overline{\mathbf{A}} \preceq \overline{\mathbf{B}}$.

**Definition 3.0.1.** *We say a multi-edge $e$ is $c$-bounded if*

$$\left\| w(e)\overline{\boldsymbol{b}_e\boldsymbol{b}_e^\top} \right\| \leq c.$$

Given a Laplacian $\mathbf{S}$ that corresponds to a multi-graph $G_{\mathbf{S}}$, and a scalar $\rho > 0$, we say that $\mathbf{S}$ is $c$-bounded if every multi-edge of $\mathbf{S}$ is $c$-bounded. Since every multi-edge of $\mathbf{U}$ is trivially 1-bounded, we can obtain a $c$-bounded Laplacian that corresponds to the same matrix, by splitting each multi-edge into $k = \lceil \frac{1}{c} \rceil$ identical copies, with a fraction $1/k$ of the initial weight. The resulting Laplacian has at most $km$ multi-edges.

**Algorithm 1:** MasterCholApx$(\mathbf{U}, k, \epsilon, \delta, c)$

1 Split edges into $k = \lceil c \log^2(1/\delta)\epsilon^{-2} \rceil$ copies with $1/k$ fraction of the original weight each.
2 Define the set of vertices $F_0 = \{1, \ldots, k\}$
3 **for** $i = 1$ *to* $k$ **do**
4 Pick some index $\pi(i)$ from $F_{i-1}$ as the $i^{th}$ vertex
5 $F_i = F_{i-1} \setminus \{\pi(i)\}$
6 $\boldsymbol{c}_i \leftarrow \begin{cases} \frac{1}{\left(\widehat{\mathbf{S}}^{(i-1)}(\pi(i),\pi(i))\right)^{1/2}} \widehat{\mathbf{S}}^{(i-1)}(\pi(i),:) & \text{if } \widehat{\mathbf{S}}^{(i-1)}(\pi(i),\pi(i)) \neq 0 \\ \mathbf{0} & \text{otherwise} \end{cases}$
7 $\widehat{\mathbf{C}}_i \leftarrow$ CliqueSample$(\widehat{\mathbf{S}}^{(i-1)}, \pi(i))$
8 $\widehat{\mathbf{S}}^{(i)} \leftarrow \widehat{\mathbf{S}}^{(i-1)} - \mathrm{ST}\left[\widehat{\mathbf{S}}^{(i-1)}\right]_{\pi(i)} + \widehat{\mathbf{C}}_i$
9 $\widehat{\mathbf{S}} \leftarrow \widehat{\mathbf{S}}^{(k)}$
10 $\widehat{\boldsymbol{\mathcal{L}}} \leftarrow \begin{pmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \ldots & \boldsymbol{c}_k \end{pmatrix}$
11 **return** $(\widehat{\boldsymbol{\mathcal{L}}}, \widehat{\mathbf{S}})$

## 3.1 The Master Cholesky Approximation Algorithm

**Theorem 3.1.1.** *Suppose $G = (V, E)$ is a connected undirected multi-graph with positive edge weights $w : E \to \mathbb{R}_+$, and associated Laplacian $\mathbf{U}$, and that $G$ has $m$ $c$-bounded multi-edges. Given scalars $\delta < 1/n^{100}$, $0 < \epsilon \leq 1/2$, the algorithm* MasterCholApx$(\mathbf{U}, k, \epsilon, \delta, c)$ *returns an an approximate partial Cholesky decomposition $\widehat{\boldsymbol{\mathcal{L}}}, \widehat{\mathbf{S}}$ onto the set $\{k+1, \ldots, n\}$ s.t. with probability at least $1 - O(\delta)$,*

$$\widehat{\mathbf{S}} + \widehat{\boldsymbol{\mathcal{L}}}\widehat{\boldsymbol{\mathcal{L}}}^\top \approx_\epsilon \mathbf{U}, \tag{3.1}$$

*and $\widehat{\mathbf{S}}$ is always a Laplacian supported on $\{k+1, \ldots, n\}$. Depending on what rule is used in Line 4 of the algorithm, the following are true:*

*(i) If each vertex is picked uniformly at random from the remaining vertex set $F_{i-1}$, then for all $t > 1$, the maximum number of non-zero entries in $\widehat{\boldsymbol{\mathcal{L}}}$ and $\widehat{\mathbf{S}}$ and the total running time are each bounded by $O(tm \log n \lceil c \log^2(1/\delta)\epsilon^{-2} \rceil)$ with probability $1 - 1/n^t$.*

*(ii) If each vertex is picked uniformly at random from the vertices in $F_{i-1}$ with at most twice the average degree of those in this set then the maximum number of non-zero entries in $\widehat{\boldsymbol{\mathcal{L}}}$ and $\widehat{\mathbf{S}}$ and the total running time are each bounded by*

$$O(m \log(n/(n-k)) \lceil c \log^2(1/\delta)\epsilon^{-2} \rceil).$$

The following lemma captures some important facts about, Algorithm 2, the CliqueSample routine, which is used inside MasterCholApx. We prove this lemma in Section 3.1.1.

**Lemma 3.1.2.** *Given a Laplacian matrix $\mathbf{S}$ that is $1/\rho$-bounded w.r.t. $\mathbf{U}$ and a vertex $v$,* CliqueSample$(\mathbf{S}, v)$ *returns a sum $\sum_e \mathbf{Y}_e$ of $\deg_{\mathbf{S}}(v)$ IID samples $\mathbf{Y}_e \in \mathbb{R}^{n \times n}$. The following conditions hold*

1. *$\mathbf{Y}_e$ is 0 or the Laplacian of a multi-edge with endpoints $u_1, u_2$, where $u_1, u_2$ are neighbors of $v$ in $\mathbf{S}$.*

**Algorithm 2:** $\sum_i \mathbf{Y}_i = \textsc{CliqueSample}(\mathbf{S}, v)$ : Returns several i.i.d samples of edges from the clique generated after eliminating vertex $v$ from the multi-graph represented by $\mathbf{S}$

```
for i ← 1 to deg_S(v) do
    Sample e_1 from list of multi-edges incident on v with probability w(e)/w_S(v)
    Sample e_2 uniformly from list of multi-edges incident on v
    if e_1 has endpoints v, u_1 and e_2 has endpoints v, u_2 and u_1 ≠ u_2 then
        Y_i ← (w(e_1)w(e_2))/(w(e_1)+w(e_2)) b_{u_1,u_2} b_{u_1,u_2}^⊤
    else
        Y_i ← 0
return ∑_i Y_i
```

*2.* $\mathbb{E}\sum_e \mathbf{Y}_e = \textsc{Cl}[\mathbf{S}]_v$.
*3.* $\left\|\overline{\mathbf{Y}}_e\right\| \leq 1/\rho$, *i.e.* $\mathbf{Y}_e$ *is* $1/\rho$*-bounded w.r.t.* $\mathbf{U}$.
*The algorithm runs in time* $O(\deg_{\mathbf{S}}(v))$.

Now, let us state the Freedman Inequality from [Tro11a]. Let $(\Omega, \mathscr{F}, \Pr)$ be a probability space, and let $\mathscr{F}_0 \subseteq \mathscr{F}_1 \subseteq \mathscr{F}_2 \subseteq \ldots \subseteq \mathscr{F}$ be a filtration of the master sigma algebra. A zero-mean matrix martingale is a sequence $\{\mathbf{Y}_j : j = 0, 1, 2, \ldots\}$ that is a symmetric-matrix-valued random process which is adapted to the filtration and satisfies the following properties: $\mathbb{E}\|\mathbf{Y}_j\| \leq \infty$ for all $j$, and $\mathbf{Y}_0 \preceq 0$, and $\mathbb{E}[\mathbf{Y}_{j+1}|\mathscr{F}_j] = \mathbf{Y}_j$. For a more compact notation, we write $\mathbb{E}_{<j}\mathbf{Z} \stackrel{\text{def}}{=} \mathbb{E}[\mathbf{Z}|\mathscr{F}_{j-1}]$.

**Theorem 3.1.3.** *Consider a zero-mean matrix martingale whose values are symmetric matrices* $\{\mathbf{Y}_j : j = 0, 1, 2, \ldots\}$ *with dimension* $d$*, and let* $\{\mathbf{X}_j : j = 1, 2, \ldots\}$ *be the difference sequence* $\mathbf{X}_j = \mathbf{Y}_j - \mathbf{Y}_{j-1}$*. Assume the difference sequence is bounded in the sense that*

$$\lambda_{\max}(\mathbf{X}_j) \leq R \text{ almost surely for } j = 1, 2, 3, \ldots.$$

*Define the* predictable quadratic variation process *of the martingale:*

$$W_j = \sum_{i=1}^{j} \mathbb{E}\left[\mathbf{X}_i^2 \middle| \mathscr{F}_{i-1}\right] \text{ for } j = 1, 2, 3, \ldots.$$

*Then, for all* $t \geq 0$ *and* $\sigma^2 > 0$,

$$\Pr[\exists j \geq 0 : \lambda_{\max}(\mathbf{Y}_j) \geq t \text{ and } \lambda_{\max}(\mathbf{W}_j) \leq \sigma^2] \leq d \exp\left(-\frac{t^2/2}{\sigma^2 + Rt/3}\right).$$

*Proof of Theorem 3.1.1.* We will set up a martingale and apply Theorem 3.1.3. In order to apply Theorem 3.1.3, we need a total ordering on the random variables, and a sequence of filtrations. Consider the $i$th elimination, and the $e$th edge sample of this round of elimination: This gives a pair $(i, e)$, and we overload the $\leq$ and $<$ to compare pair of variables in the lexicographical sense. We use $\mathscr{F}_{(i,e)}$ to denote the filtration corresponding to conditioning on all the random choices of the algorithm up until and including the sampling of edge, and when $(i, e)$ contains the last edge sample $e$ of elimination $i$, the filtration $\mathscr{F}_{(i,e)}$ also includes conditioning on the vertex choice for round $i + 1$. This ensures that conditioning on some $\mathscr{F}_{(i,e)}$ will determine the vertex at which the next edge is sampled.

We define the $j^{\text{th}}$ approximate partial Cholesky decomposition

$$\mathbf{U}^{(j)} = \widehat{\mathbf{S}}^{(j)} + \sum_{i=1}^{j} \boldsymbol{c}_i \boldsymbol{c}_i^\top. \tag{3.2}$$

Thus our final output equals $\mathbf{U}^{(n)}$.

Observe that

$$\begin{aligned}
\mathbf{U}^{(j)} - \mathbf{U}^{(j-1)} &= \boldsymbol{c}_j \boldsymbol{c}_j^\top + \widehat{\mathbf{S}}^{(j)} - \widehat{\mathbf{S}}^{(j-1)} \\
&= \boldsymbol{c}_j \boldsymbol{c}_j^\top + \widehat{\mathbf{C}}_j - \textsc{St}\left[\widehat{\mathbf{S}}^{(j-1)}\right]_{\pi(j)} \\
&= \widehat{\mathbf{C}}_j - \textsc{Cl}\left[\widehat{\mathbf{S}}^{(j-1)}\right]_{\pi(j)}.
\end{aligned}$$

Each call to CliqueSample returns a sum of sample edges. Letting $\mathbf{Y}_e^{(j)}$ denote the $e^{\text{th}}$ sample in the $j^{\text{th}}$ call to CliqueSample, we can write this sum as $\sum_e \mathbf{Y}_e^{(j)}$. We can apply Lemma 3.1.2 to find that the expected value of $\widehat{\mathbf{C}}_j$ is $\sum_e \mathbb{E}_{<(j,e)} \mathbf{Y}_e^{(j)} = \textsc{Cl}\left[\widehat{\mathbf{S}}^{(j-1)}\right]_{\pi(j)}$. Hence the expected value of $\mathbf{U}^{(j)}$ is exactly $\mathbf{U}^{(j-1)}$, and we can write

$$\mathbf{U}^{(j)} - \mathbf{U}^{(j-1)} = \sum_e \mathbf{Y}_e^{(j)} - \mathop{\mathbb{E}}_{<(j,e)} \mathbf{Y}_e^{(j)}.$$

By defining $\mathbf{X}_e^{(j)} \stackrel{\text{def}}{=} \mathbf{Y}_e^{(j)} - \mathbb{E}_{<(j,e)} \mathbf{Y}_e^{(j)}$, this becomes $\mathbf{U}^{(j)} - \mathbf{U}^{(j-1)} = \sum_e \mathbf{X}_e^{(j)}$. So, without conditioning on the choices of the CholApx algorithm, we can write

$$\mathbf{U}^{(j)} - \mathbf{U}^{(0)} = \sum_{i=1}^{j} \sum_e \mathbf{X}_e^{(i)}.$$

For the sequence of $j$, this is a zero-mean martingale.

Ultimately, our goal is to show that

$$-\epsilon \mathbf{U} \preceq \mathbf{U}^{(n)} - \mathbf{U} \preceq \epsilon \mathbf{U} \tag{3.3}$$

with high probability. It turns out to be easier to consider a related martingale, which includes a simple stopping condition. We define

$$\mathbf{Z}_e^{(i)} = \begin{cases} \overline{\mathbf{X}}_e^{(i)} & \text{if } \sum_{j<i} \sum_f \mathbf{Z}_f^{(j)} \preceq \epsilon \boldsymbol{\Pi} \\ \mathbf{0} & \text{o.w.} \end{cases} \tag{3.4}$$

We then consider the zero-mean martingale

$$\mathbf{T}_{(i,e)} = \sum_{(j,f) \le (i,e)} \mathbf{Z}_f^{(j)}$$

Now, if

$$-\epsilon \boldsymbol{\Pi} \preceq \mathbf{T}_{(i,e)} \preceq \epsilon \boldsymbol{\Pi}$$

then Equation (3.3) is satisfied. Next we apply Theorem 3.1.3 to prove concentration of $\mathbf{T}_{(i,e)}$.

Let $\mathbf{W}_{(i,e)} = \sum_{(j,f)\le(i,e)} \mathbb{E}_{<(j,f)}(\overline{\mathbf{Z}}_f^{(j)})^2$ Then

$$\begin{aligned}\Pr[\mathbf{U}^{(n)} \preceq (1+\epsilon)\mathbf{U}] &\le \Pr[\exists(i,e) : \lambda_{\max}(\mathbf{T}_{(i,e)}) \ge \epsilon] \\ &\le \Pr[\exists(i,e) : \lambda_{\max}(\mathbf{T}_{(i,e)}) \ge \epsilon \text{ and } \lambda_{\max}(\mathbf{W}_{(i,e)}) \le \sigma^2] \\ &\quad + \Pr[\exists(i,e) : \lambda_{\max}(\mathbf{W}_{(i,e)}) > \sigma^2]\end{aligned}$$

We start by bounding the first probability among these two terms. Initially, note that either $\overline{\mathbf{Z}}_e^{(i)} = \mathbf{0}$ or $\overline{\mathbf{Z}}_e^{(i)} = \overline{\mathbf{Y}}_e^{(i)} - \mathbb{E}\,\overline{\mathbf{Y}}_e^{(i)}$, so

$$\begin{aligned}\left\|\overline{\mathbf{Z}}_e^{(i)}\right\| &\le \left\|\overline{\mathbf{Y}}_e^{(i)} - \mathbb{E}\,\overline{\mathbf{Y}}_e^{(i)}\right\| \\ &\le \max\left\{\left\|\overline{\mathbf{Y}}_e^{(i)}\right\|, \left\|\mathbb{E}\,\overline{\mathbf{Y}}_e^{(i)}\right\|\right\} \qquad \text{(since both terms are PSD)} \\ &\le 1/\rho, \end{aligned} \tag{3.5}$$

by the $1/\rho$-boundedness guarantee of Lemma 3.1.2. Thus, a valid norm bound for the application of Theorem 3.1.3 is $R \le 1/\rho = \frac{\epsilon^2}{3\log^2(1/\delta)}$. Finally, we choose $\sigma^2 = \frac{\log(1/\delta)}{\rho}$. By Theorem 3.1.3, this gives

$$\begin{aligned}\Pr[\exists(i,e) : \lambda_{\max}(\mathbf{T}_{(i,e)}) \ge \epsilon \text{ and } \lambda_{\max}(\mathbf{W}_{(i,e)}) \le \sigma^2] &\le n\exp\left(-\frac{\epsilon^2/2}{\sigma^2 + R\epsilon/3}\right) \\ &\le n\exp\left(-\frac{\rho\epsilon^2/2}{\log(1/\delta) + \epsilon^3/3}\right) \\ &\le \delta.\end{aligned}$$

Secondly, we bound the term $\Pr[\exists(i,e) : \lambda_{\max}(\mathbf{W}_{(i,e)}) > \sigma^2]$. We define a notation of $\mathbf{W}_{(i,e)}$ at the last edge sample $\mathbf{W}_i \stackrel{\text{def}}{=} \mathbf{W}_{(i,e_{last})}$. Note that $\mathbf{W}_{(i,e)} \preceq \mathbf{W}_{(i',e')}$, whenever $(i,e) \le (i',e')$. Thus $\Pr[\exists(i,e) : \lambda_{\max}(\mathbf{W}_{(i,e)}) > \sigma^2] = \Pr[\exists i : \lambda_{\max}(\mathbf{W}_i) > \sigma^2]$.

We now construct a zero mean martingale which we will use to bound this probability, by *another application* of Theorem 3.1.3. In this application, we need a different sequence of filtrations than before. We define $\mathscr{F}_i$ to denote the filtration corresponding to conditioning on all the random choices of the algorithm in round $i$, *but not including the random choice of vertex for round* $i+1$.

We define $\mathbf{W}_0 = \mathbf{0}$. Let $\mathbf{V}_i \stackrel{\text{def}}{=} \mathbf{W}_i - \mathbf{W}_{i-1} - \mathbb{E}_{<i}[\mathbf{W}_i - \mathbf{W}_{i-1}] = \mathbf{W}_i - \mathbb{E}_{<i}[\mathbf{W}_i]$. $\mathbf{V}_i$ is zero-mean conditional on $\mathscr{F}_{i-1}$, and so $\mathbf{R}_i = \sum_{j=1}^{i} \mathbf{V}_j$ is a zero-mean martingale.

$$\begin{aligned}\mathbf{R}_i &= \sum_{j=1}^{i} \mathbf{W}_j - \mathbf{W}_{j-1} - \mathop{\mathbb{E}}_{<j}[\mathbf{W}_j - \mathbf{W}_{j-1}] \\ &= \mathbf{W}_i - \sum_{j=1}^{i} \mathop{\mathbb{E}}_{<j}[\mathbf{W}_j - \mathbf{W}_{j-1}]\end{aligned}$$

Let $\mathbf{M}_i = \sum_{j\le i} \mathbb{E}_{<j}\,\mathbf{V}_j^2$. In the terminology of Theorem 3.1.3, $\mathbf{W}_{(i,e)}$ is the *predictable quadratic variation process* of $\mathbf{T}_{(i,e)}$. $\mathbf{R}_i$ is a zero-mean martingale corresponding to $\mathbf{W}_i$, $\mathbf{V}_i$ is the associated difference sequence, and $\mathbf{M}_i$ is the *predictable quadratic variation process* of $\mathbf{R}_i$.

Having established the necessary notation, we state a few key facts about these random matrices in Claim 3.1.4 below. We will prove Claim 3.1.4 later, but first we will see how to use the claim to complete the proof of Theorem 3.1.1.

**Claim 3.1.4.**

1. $\mathbf{0} \preceq \mathbf{W}_j - \mathbf{W}_{j-1} \preceq \frac{3}{2\rho}\mathbf{\Pi}$.
2. $\sum_j \mathbb{E}_{<j}\left[\mathbf{W}_j - \mathbf{W}_{j-1}\right] \preceq \frac{6\ln n}{\rho}\mathbf{\Pi}$.
3. $\mathbf{M}_i \preceq \frac{9\ln n}{\rho^2}\mathbf{\Pi}$ .

Let $\omega^2 = \frac{9}{\rho^2}\ln n$. Then by Claim 3.1.4, Part 3, we have $\Pr[\exists i : \lambda_{\max}(\mathbf{M}_i) > \omega^2] = 0$. Now, using Claim 3.1.4, Part 2, we get

$$\begin{aligned}
\Pr[\exists i : \lambda_{\max}(\mathbf{W}_i) > \sigma^2] &= \Pr\left[\exists i : \lambda_{\max}\left(\mathbf{R}_i + \sum_{j=1}^{i} \mathop{\mathbb{E}}_{<j}\left[\mathbf{W}_j - \mathbf{W}_{j-1}\right]\right) > \sigma^2\right] \\
&\leq \Pr\left[\exists i : \lambda_{\max}\left(\mathbf{R}_i\right) > \sigma^2 - \frac{6\ln n}{\rho}\right] \\
&= \Pr\left[\exists i : \lambda_{\max}(\mathbf{R}_i) \geq \sigma^2 - \frac{6\ln n}{\rho} \text{ and } \lambda_{\max}(\mathbf{M}_i) \leq \omega^2\right].
\end{aligned}$$

We now want to apply Theorem 3.1.3, to bound the probability above, with $\mathbf{R}_i$ as the zero-mean martingale, $\mathbf{V}_i$ is the associated difference sequence, and $\mathbf{M}_i$ as the predictable quadratic variation process. By Claim 3.1.4, Part 1, we have

$$\begin{aligned}
\|\mathbf{V}_i\| &= \left\|\mathbf{W}_i - \mathbf{W}_{i-1} - \mathop{\mathbb{E}}_{<i}\left[\mathbf{W}_i - \mathbf{W}_{i-1}\right]\right\| \\
&\leq \max\left\{\|\mathbf{W}_i - \mathbf{W}_{i-1}\|, \left\|\mathop{\mathbb{E}}_{<i}\left[\mathbf{W}_i - \mathbf{W}_{i-1}\right]\right\|\right\} \qquad \text{(since both terms are PSD)} \\
&\leq 1/\rho,
\end{aligned}$$

which gives us a value for the norm control parameter $R$. Thus by Theorem 3.1.3, and using $\log(1/\delta) \geq 100\log n$, we get

$$\begin{aligned}
\Pr\left[\exists i : \lambda_{\max}(\mathbf{R}_i) \geq \sigma^2 - \frac{6}{\rho}\ln n \text{ and } \lambda_{\max}(\mathbf{M}_i) \leq \omega^2\right] &\leq n\exp\left(-\frac{\left(\frac{\log(1/\delta)}{\rho} - \frac{6}{\rho}\ln n\right)^2/2}{\left(\frac{6}{\rho^2}\ln n\right) + \left(\frac{\log(1/\delta)}{\rho^2}\right)/3}\right) \\
&\leq \delta.
\end{aligned}$$

An essentially identical proof can be used to control $\Pr[\exists (i,e) : \lambda_{\max}(-\mathbf{T}_{(i,e)}) \geq \epsilon]$, since all norm calculations remain the same and $-\mathbf{T}_{(i,e)}$ and $\mathbf{T}_{(i,e)}$ have the same predictable quadratic variation process. All together this completes the proof of the claim that

$$\widehat{\mathbf{S}}^{(j)} + \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_\epsilon \mathbf{U}. \tag{3.6}$$

holds with probability at least probability at least $1 - 4\delta$.

Finally, we need to bound the expected running time of the algorithm. We start by observing that the algorithm maintains the two following invariants:

1. Every multi-edge in $\widehat{\mathbf{S}}^{(j-1)}$ is $1/\rho$-bounded.
2. The total number of multi-edges is at most $\rho m$.

We establish the first invariant inductively. The invariant holds for $\widehat{\mathbf{S}}^{(0)}$, because of the splitting of original edges into $\rho$ copies with weight $1/\rho$. The invariant thus also holds for $\widehat{\mathbf{S}}^{(0)} - \textsc{St}\left[\widehat{\mathbf{S}}^{(0)}\right]_{\pi(1)}$, since the multi-edges of this Laplacian are a subset of the previous ones. By Lemma 3.1.2, every multi-edge $Y_e$ output by CliqueSample is $1/\rho$-bounded, so $\widehat{\mathbf{S}}^{(1)} = \widehat{\mathbf{S}}^{(0)} - \textsc{St}\left[\widehat{\mathbf{S}}^{(0)}\right]_{\pi(1)} + \widehat{C}_1$ is $1/\rho$-bounded. If we apply this argument repeatedly for $j = 1, \ldots, n-1$ we get invariant (1).

Invariant (2) is also very simple to establish: It holds for $\widehat{\mathbf{S}}^{(0)}$, because splitting of original edges into $\rho$ copies does not produce more than $\rho m$ multi-edges in total. When computing $\widehat{\mathbf{S}}^{(j)}$, we subtract $\textsc{St}\left[\widehat{\mathbf{S}}^{(j-1)}\right]_{\pi(j)}$, which removes exactly $\deg_{\widehat{\mathbf{S}}^{(j-1)}}(\pi(j))$ multi-edges, while we add the multi-edges produced by the call to CliqueSample$(\widehat{\mathbf{S}}^{(j-1)}, \pi(j))$, which is at most $\deg_{\widehat{\mathbf{S}}^{(j-1)}}(\pi(j))$. So the number of multi-edges is not increasing.

Finally, the statement of Theorem 3.1.1 considers two variants of Algorithm 1, depending on the rule used for picking a random vertex in Line 4 of the algorithm. First we prove the statement for Rule (i), where a vertex is chosen uniformly at random. By Lemma 3.1.2, the running time for the call to CliqueSample is $O(\deg_{\widehat{\mathbf{S}}^{(j)}}(\pi(j)))$. Given the invariants, we get that the expected time for the $j^{th}$ call to CliqueSample is $O(\mathbb{E}_{\pi(j)} \deg_{\widehat{\mathbf{S}}^{(j)}}(\pi(j))) = O(\rho m/(k+1-j))$. Thus the expected running time of all calls to CliqueSample is $O(m\rho \sum_{j=1}^{k} \frac{1}{k+1-j}) = O(m\epsilon^{-2} \log^2(1/\delta) \log n)$. This also bounds the expected running time of the whole algorithm. The total number of entries in the $\widehat{\boldsymbol{\mathcal{L}}}, \widehat{\mathbf{S}}$ matrices must also be bounded by $O(m\epsilon^{-2} \log^2(1/\delta) \log n)$ in expectation. Finally, we can also show that the running time and number of non-zero entries in the output of the algorithm is concentrated. The conditional on previous rounds, the running time in round $j$ of elimination is a positive random variable that is at most $2\rho m$, and in expectation $2\rho m \cdot \frac{1}{k+1-j}$. By subtracting the average in each round, we can get a scalar martingale. Using a standard application of the scalar Freedman Inequality [Fre75] (see also [Tro11a] for a convenient version), one can show that the running time is upper bounded by $O(c\rho m \log n)$ with probability $1 - 1/n^c$ for all $c \geq 1$. This completes the analysis for the Rule (i) case.

We now consider the Rule (ii) case, where for each elimination a vertex is chosen uniformly at random among the remaining vertices of $F_{i-1}$ with at most twice the average degree among vertices in $F_{i-1}$. It is simple to construct a data structure for picking such a vertex: We maintain an array of degrees of remaining vertices, as well as their sum. We then pick a random vertex from this array, repeating until we get a vertex with at most twice the average degree. Degrees in the array can be updated efficiently as edges are added or removed. To handle vertex deletions from this array, we can mark vertices as deleted when we eliminate them, and resize the array each time half the vertices have been marked as deleted. With probability $> 1 - \delta$, the time spent on vertex sampling using this data structure is dominated by other terms in the running time.

By Lemma 3.1.2, the running time for the call to CliqueSample is $O(\deg_{\widehat{\mathbf{S}}^{(j)}}(\pi(j)))$. Given the invariants, we get that the time for the $j^{th}$ call to CliqueSample is $O(\mathbb{E}_{\pi(j)} \deg_{\widehat{\mathbf{S}}^{(j)}}(\pi(j))) = O(\rho m/(k+1-j))$. Thus the running time of all calls to CliqueSample is $O(m\rho \sum_{j=1}^{k} \frac{1}{k+1-j}) = O(m\epsilon^{-2} \log^2(1/\delta) \log n)$. This also bounds the running time of the whole algorithm. The total number of entries in the $\widehat{\boldsymbol{\mathcal{L}}}, \widehat{\mathbf{S}}$ matrices must also be bounded by $O(m\epsilon^{-2} \log^2(1/\delta) \log n)$. □

### 3.1.1 Clique Sampling Proofs

In this section, we prove Lemmas 3.1.2 and 3.1.4 that characterize the behavior of our algorithm CliqueSample, which is used in CholApx to approximate the clique generated by eliminating a variable.

A important element of the CliqueSample algorithm is our very simple approach to leverage

score estimation. Using the well-known result that effective resistance in Laplacians is a distance (see Lemma 3.1.6), we give a bound on the leverage scores of all edges in a clique that arises from elimination. Given a undirected Laplacian $\mathbf{S}$, we let $w_{\mathbf{S}}(v)$ the sum of the weights of the edges incident on vertex $v$, i.e.

$$w_{\mathbf{S}}(v) = \sum_{\substack{e \in E(\mathbf{S}) \\ e \ni v}} w(e).$$

Then by Equation (2.3.6)

$$\textsc{Cl}[\mathbf{S}]_{v_1} = \frac{1}{2} \sum_{\substack{e \in E(\mathbf{S}) \\ e \text{ has} \\ \text{endpoints} \\ v,u}} \sum_{\substack{e' \in E(\mathbf{S}) \\ e' \text{ has} \\ \text{endpoints} \\ v, z \neq u}} \frac{w(e)w(e')}{w_{\mathbf{S}}(v)} \boldsymbol{b}_{u,z} \boldsymbol{b}_{u,z}^{\top}. \tag{3.7}$$

Note that the factor $\frac{1}{2}$ accounts for the fact that every pair is double counted.

**Lemma 3.1.5.** *Suppose multi-edges $e, e' \ni v$ are $1/\rho$-bounded w.r.t. $\mathbf{U}$, and have endpoints $v, u$ and $v, z$ respectively, and $z \neq u$, then $w(e)w(e')\boldsymbol{b}_{u,z}\boldsymbol{b}_{u,z}^{\top}$ is $\frac{w(e)+w(e')}{\rho}$-bounded.*

To prove Lemma 3.1.5, we need the following result about Laplacians:

**Lemma 3.1.6.** *Given a connected weighted multi-graph $G = (V, E, w)$ with associated Laplacian matrix $L$ in $G$, consider three distinct vertices $u, v, z \in V$, and the pair-vectors $\boldsymbol{b}_{u,v}$, $\boldsymbol{b}_{v,z}$ and $\boldsymbol{b}_{u,z}$.*

$$\left\| \overline{\boldsymbol{b}_{u,z}\boldsymbol{b}_{u,z}^{\top}} \right\| \leq \left\| \overline{\boldsymbol{b}_{u,v}\boldsymbol{b}_{u,v}^{\top}} \right\| + \left\| \overline{\boldsymbol{b}_{v,z}\boldsymbol{b}_{v,z}^{\top}} \right\|.$$

This is known as phenomenon that Effective Resistance is a distance [KR93].

*Proof of Lemma 3.1.5.* Using the previous lemma:

$$w(e)w(e') \left\| \overline{\boldsymbol{b}_{u,z}\boldsymbol{b}_{u,z}^{\top}} \right\| \leq w(e)w(e') \left( \left\| \overline{\boldsymbol{b}_{u,v}\boldsymbol{b}_{u,v}^{\top}} \right\| + \left\| \overline{\boldsymbol{b}_{v,z}\boldsymbol{b}_{v,z}^{\top}} \right\| \right) \leq \frac{1}{\rho} \left( w(e) + w(e') \right).$$

□

To prove Lemma 3.1.2, we need the following result of Walker [Wal77] (see Bringmann and Panagiotou [BP12] for a modern statement of the result).

**Lemma 3.1.7.** *Given a vector $p \in \mathbb{R}^d$ of non-negative values, the procedure* UnsortedProportionalSampling *requires $O(d)$ preprocessing time and after this allows for IID sampling for a random variable $x$ distributed s.t.*

$$\Pr[x = i] = p(i) / \|p\|_1.$$

*The time required for each sample is $O(1)$.*

**Remark 3.1.8.** We note that there are simpler sampling constructions than that of Lemma 3.1.7 that need $O(\log n)$ time per sample, and using such a method would only worsen our running time by a factor $O(\log n)$.

*Proof of Lemma 3.1.2.* From Lines (5) and (7), $\mathbf{Y}_i$ is 0 or the Laplacian of a multi-edge with endpoints $u_1, u_2$. To upper bound the running time, it is important to note that we do *not* need access to the entire matrix $S$. We only need the multi-edges incident on $v$. When calling CliqueSample, we only pass a copy of just these multi-edges.

We observe that the uniform samples in Line (3) can be done in $O(1)$ time each, provided we count the number of multi-edges incident on $v$ to find $\deg_{\mathbf{S}}(v)$. We can compute $\deg_{\mathbf{S}}(v)$ in $O(\deg_{\mathbf{S}}(v))$ time. Using Lemma 3.1.7, if we do $O(\deg_{\mathbf{S}}(v))$ time preprocessing, we can compute each sample in Line (2) in time $O(1)$. Since we do $O(\deg_{\mathbf{S}}(v))$ samples, the total time for sampling is hence $O(\deg_{\mathbf{S}}(v))$.

Now we determine the expected value of the sum of the samples. Note that in the sum below, each pair of multi-edges appears twice, with different weights.

$$\mathbb{E}\sum_i \mathbf{Y}_i = \deg_{\mathbf{S}}(v) \sum_{\substack{e\in E(\mathbf{S}) \\ e \text{ has} \\ \text{endpoints} \\ v,u}} \sum_{\substack{e'\in E(\mathbf{S}) \\ e' \text{ has} \\ \text{endpoints} \\ v,z\neq u}} \frac{w(e)}{w_{\mathbf{S}}(v)} \frac{1}{\deg_{\mathbf{S}}(v)} \frac{w(e)w(e')}{w(e)+w(e')} \boldsymbol{b}_{u,z}\boldsymbol{b}_{u,z}^\top = \textsc{Cl}[\mathbf{S}]_v\,.$$

By Lemma 3.1.5,

$$\left\|\overline{\mathbf{Y}_i}\right\| \leq \max_{\substack{e,e'\in E(\mathbf{S}) \\ e,e' \text{ has} \\ \text{endpoints} \\ v,u \text{ and } v,z\neq u}} \frac{w(e)w(e')}{w(e)+w(e')} \left\|\overline{\boldsymbol{b}_{u,z}\boldsymbol{b}_{u,z}^\top}\right\| \leq 1/\rho.$$

□

*Proof of Claim 3.1.4.* We first establish Part 1.

$$\mathbf{W}_i - \mathbf{W}_{i-1} = \sum_e \mathop{\mathbb{E}}_{<(i,e)} (\mathbf{Z}_e^{(i)})^2$$

We consider two cases, the first case is when the condition

$$\sum_{j<i}\sum_f \mathbf{Z}_f^{(j)} \preceq \epsilon\boldsymbol{\Pi} \tag{3.8}$$

holds. In this case, by (3.4), we have

$$\mathbf{W}_i - \mathbf{W}_{i-1} = \sum_e \mathop{\mathbb{E}}_{<(i,e)} (\overline{\mathbf{X}}_e^{(i)})^2 \succeq \mathbf{0}.$$

and, using the norm bound $\left\|\overline{\mathbf{Y}}_e^{(i)}\right\| \leq 1/\rho$ established Equation (3.5) in the proof of Theorem 3.1.1, we get

$$\begin{aligned}\sum_e \mathop{\mathbb{E}}_{<(i,e)} (\overline{\mathbf{X}}_e^{(i)})^2 &= \sum_e \mathop{\mathbb{E}}_{<(i,e)} (\overline{\mathbf{Y}}_e^{(i)})^2 - \left(\mathop{\mathbb{E}}_{<(i,e)} \overline{\mathbf{Y}}_e^{(i)}\right)^2 \preceq \sum_e \mathop{\mathbb{E}}_{<(i,e)} (\overline{\mathbf{Y}}_e^{(i)})^2 \\ &\preceq \sum_e \mathop{\mathbb{E}}_{<(i,e)} \left\|\overline{\mathbf{Y}}_e\right\| \overline{\mathbf{Y}}_e \preceq \frac{1}{\rho}\overline{\textsc{Cl}\left[\widehat{\mathbf{S}}^{(i-1)}\right]_{\pi(i)}} \preceq \frac{1}{\rho}\overline{\textsc{St}\left[\widehat{\mathbf{S}}^{(i-1)}\right]_{\pi(i)}}.\end{aligned} \tag{3.9}$$

Equation (3.8) implies, using $\epsilon \leq 1/2$,

$$\mathbf{U}^{(i-1)} = \mathbf{U} + \sum_{j=1}^{i-1}\sum_e \mathbf{X}_e^{(j)} \preceq (1+\epsilon)\mathbf{U} \preceq \frac{3}{2}\mathbf{U}. \tag{3.10}$$

This in turn gives

$$\mathbf{0} \preceq \mathbf{W}_i - \mathbf{W}_{i-1} \preceq \frac{1}{\rho}\overline{\mathrm{St}\left[\widehat{\mathbf{S}}^{(i-1)}\right]_{\pi(i)}} \preceq \frac{1}{\rho}\overline{\mathbf{U}^{(i)}} \preceq \frac{3}{2\rho}\mathbf{\Pi}.$$

When the condition in Equation (3.8) does not hold, by (3.4), $\mathbf{W}_i - \mathbf{W}_{i-1} = \mathbf{0}$. All together, this completes the proof of Part 1.

We now turn to Part 2. First we consider the case when the condition in Equation (3.8) holds. Note that $\mathbb{E}_{<i} \mathrm{St}\left[\widehat{\mathbf{S}}^{(i-1)}\right]_{\pi(i)} \preceq \frac{4}{n-i-1}\widehat{\mathbf{S}}^{(i-1)}$, since we choose from $\frac{n-i-1}{2}$ vertices and every edge is included for at most 2 of those choices.

Combining this with Equations (3.9) and (3.10) gives

$$\mathbf{0} \preceq \sum_i \mathop{\mathbb{E}}_{<i}\left[\mathbf{W}_i - \mathbf{W}_{i-1}\right] \preceq \mathbf{\Pi}\sum_i \frac{6}{\rho(n-i-1)} \preceq \frac{6\ln n}{\rho}\mathbf{\Pi}.$$

This establishes Part 2. Finally, we show Part 3.

$$\begin{aligned}\mathbf{M}_i &= \sum_{j\le i} \mathop{\mathbb{E}}_{<j} \mathbf{V}_j^2 = \sum_{j\le i} \mathop{\mathbb{E}}_{<j}\left(\mathbf{W}_j - \mathbf{W}_{j-1} - \mathop{\mathbb{E}}_{j}\left[\mathbf{W}_j - \mathbf{W}_{j-1}\right]\right)^2 \preceq \sum_{j\le i} \mathop{\mathbb{E}}_{<j}\left(\mathbf{W}_j - \mathbf{W}_{j-1}\right)^2 \\ &\preceq \sum_{j\le i} \mathop{\mathbb{E}}_{<j}\left(\mathbf{W}_j - \mathbf{W}_{j-1}\right)\left\|\mathbf{W}_j - \mathbf{W}_{j-1}\right\| \preceq \frac{9\ln n}{\rho^2}\mathbf{\Pi}.\end{aligned}$$

□

## 3.2 Solving Laplacian Linear Systems using Approximate Gaussian Elimination

We use $\textsc{CholApx}(\mathbf{U}, \delta) \stackrel{\text{def}}{=} \textsc{MasterCholApx}(\mathbf{U}, k \leftarrow n, \epsilon \leftarrow 1/2, \delta, c \leftarrow 1)$ to denote the algorithm we get from calling MasterCholApx with $k = n$, $\epsilon = 1/2$, and $c = 1$, and using Rule (ii) for eliminations. As an immediate corollary of Theorem 3.1.1 with these parameter settings, we get the following.

**Theorem 1.2.1** (Approximate Cholesky Factorization for Laplacians)**.** *Suppose $\mathbf{U} \in \mathbb{R}^{n\times n}$ is a Laplacian matrix with $m$ non-zero entries. Given a scalar $\delta < 1/n^{100}$, the algorithm $\textsc{CholApx}(\mathbf{U}, \delta)$ returns an approximate Cholesky factorization $\boldsymbol{\mathcal{L}}$ s.t. with probability at least $1 - O(\delta)$,*

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_{1/2} \mathbf{U}.$$

*The maximum number of non-zero entries in $\boldsymbol{\mathcal{L}}$ and the total running time are both bounded by $O(m\log^2(1/\delta)\log n)$.*

The sparse approximate Cholesky factorization for $\mathbf{U}$ given by Theorem 1.2.1 immediately implies fast solvers for Laplacian systems.

We use $\textsc{CholApxSolver}(\mathbf{U}, \epsilon, \delta, \boldsymbol{b})$ to denote a routine which first calls $\textsc{CholApx}(\mathbf{U}, \delta)$ to compute an approximate Cholesky factorization and then uses it in the preconditioned iterative refinement, as described in Theorem 2.3.1 (Section 2.3.2) to compute and return an approximate solution $\boldsymbol{x}^{(t)}$ of the linear system $\mathbf{U}\boldsymbol{x} = \boldsymbol{b}$.

**Corollary 1.2.2** (Laplacian Linear Equation Solver)**.** *Suppose $G = (V, E)$ is a connected undirected multi-graph with positive edge weights $w : E \to \mathbb{R}_+$, and associated Laplacian $\mathbf{U}$, and that $G$ has $m$ multi-edges. Given a scalar $\delta < 1/n^{100}$, the algorithm* CholApxSolver$(\mathbf{U}, \epsilon, \delta, \boldsymbol{b})$ *returns $\tilde{\boldsymbol{x}}$ s.t. with probability at least $1 - O(\delta)$,*

$$\left\|\tilde{\boldsymbol{x}} - \mathbf{U}^+\boldsymbol{b}\right\|_{\mathbf{U}} \leq \epsilon \left\|\mathbf{U}^+\boldsymbol{b}\right\|_{\mathbf{U}}. \tag{1.1}$$

*The running time is bounded by $O(m \log^2(1/\delta) \log n \log(1/\epsilon))$.*

## 3.3 Going Faster with Sparsification

The running time and sparsity of the CholApx algorithm for solving Laplacian linear systems can be improved by combining it with a sparsification routine.

We will build our sparsification routine using CholApx and the uniform sampling technique of [CLM+15]. One consequence of their result is that one can compute fairly good upper bounds for the leverage scores of Laplacian edges by computing leverage scores with w.r.t. a crudely subsampled version of the matrix. The leverage scores w.r.t. the crudely subsampled matrix can in turn be computed using the Johnson-Lindenstrauss approach of [SS11b].

Specialized to Laplacians and combined with the leverage score estimation procedure of Spielman and Srivastava, the uniform sampling approach gives the algorithm SubsamplingGraphSparsify, whose guarantees are described in the following theorem.

**Theorem 3.3.1.** *Let* LapSolver *be a Laplacian linear system solver s.t.* LapSolver$(\mathbf{U}, \boldsymbol{b}, \epsilon, \delta)$ *returns $\boldsymbol{x}$ s.t. with probability $1 - O(\delta)$, $\|\boldsymbol{x} - \mathbf{U}^+\boldsymbol{b}\|_L \leq \epsilon \|\mathbf{U}^+\boldsymbol{b}\|_{\mathbf{U}}$ in time $T(m, \epsilon, \delta)$.*

*Given scalars $0 < \delta < 1/n$, $0 < \epsilon < 1/2$, and $K > 1$, let $\rho = \epsilon^{-2} \log(1/\delta)$. The algorithm* SubsamplingGraphSparsify(LapSolver$, \mathbf{U}, \epsilon, \delta, K)$ *returns $\widetilde{\mathbf{U}}$ with $O(K\rho n)$ edges s.t. with probability $1 - O(\delta)$,*

$$\widetilde{\mathbf{U}} \approx_\epsilon \mathbf{U}$$

*and every multi-edge in $\widetilde{\mathbf{U}}$ is $\frac{1}{\rho}$-bounded w.r.t. $\widetilde{\mathbf{U}}$.*

*The algorithm runs in time $O(\rho K n + m \log n + T(m/K, 1/2) \log n)$.*

The theorem above is proven in [CLM+15], except they do not note that the output edges are $1/\rho$-bounded, however, this is immediate from the concentration results, provided one uses Spielman-Srivastava style replacement sampling [SS11c].

**Algorithm 3:** SparserCholApx$(\mathbf{U}, \delta)$

$K_1 \leftarrow \log^2 n$

$\widetilde{\mathbf{U}} \leftarrow$ SubsamplingGraphSparsify(CholApxSolver$, \mathbf{U}, 1/8, \delta, K_1)$

$K \leftarrow \log^{100} n$

$\widetilde{\mathbf{S}}, \widetilde{\boldsymbol{\mathcal{L}}} \leftarrow$ MasterCholApx$(\widetilde{\mathbf{U}}, k \leftarrow n(1 - 1/K^2), \epsilon \leftarrow 1/8, \delta, c \leftarrow 1/\log(1/\delta))$

$\widehat{\boldsymbol{S}} \leftarrow$ SubsamplingGraphSparsify(CholApxSolver$, \widetilde{\mathbf{S}}, 1/8, \delta, K)$

$\widehat{\boldsymbol{\mathcal{L}}} \leftarrow$ MasterCholApx$(\widehat{\boldsymbol{S}}, k \leftarrow n/K^2, \epsilon \leftarrow 1/8, \delta, c \leftarrow 1/\log(1/\delta))$

$\boldsymbol{\mathcal{L}} \leftarrow \begin{pmatrix} \widetilde{\boldsymbol{\mathcal{L}}} & \begin{pmatrix} \mathbf{0} \\ \widehat{\boldsymbol{\mathcal{L}}} \end{pmatrix} \end{pmatrix}$

**return** $\boldsymbol{\mathcal{L}}$

**Theorem 3.3.2.** *Suppose $G = (V, E)$ is a connected undirected multi-graph with positive edge weights $w : E \to \mathbb{R}_+$, and associated Laplacian $\mathbf{U}$, and that $G$ has $m$ multi-edges. Given a scalar*

$\delta < 1/n^{100}$, *the algorithm* SparserCholApx$(\mathbf{U}, \delta)$ *returns an approximate Cholesky decomposition* $\boldsymbol{\mathcal{L}}$ *s.t. with probability at least* $1 - O(\delta)$,

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top \approx_{1/2} \mathbf{U}. \tag{3.11}$$

*The running time of the algorithm is bounded by* $O(m \log^2(1/\delta) + n \log^3 n \log^2(1/\delta) \log\log(1/\delta))$, *and the maximum number of non-zero entries in* $\boldsymbol{\mathcal{L}}$ *is bounded by* $O(n \log^2 n \log^2(1/\delta) \log\log(1/\delta))$.

*Proof.* The first call to SubsamplingGraphSparsify runs in time

$$O\left((m + \frac{1}{K_1} m \log^2(1/\delta) \log n) \log n + K_1 n \log(1/\delta)\right) = O\left(m \log^2(1/\delta) + n \log(1/\delta) \log^2 n\right).$$

Output edge count is $n \log(1/\delta) \log^2 n$, and is $1/\log(1/\delta)$-bounded. So after splitting to $\frac{1}{\log^2(1/\delta)}$-bounded, the output edge count is $n \log^2(1/\delta) \log^2 n$.

Then MasterCholApx in Line 4 runs in time $n \log^2(1/\delta) \log^2 n \log\log(1/\delta)$, and the output edge count is $n \log^2(1/\delta) \log^2 n \log\log(1/\delta)$.

Next, SubsamplingGraphSparsify in Line 5 runs in expected time ($n$ refers to the original size):

$$n \log^2(1/\delta) \log^3 n \log\log(1/\delta) + n \frac{1}{K^2} K \log(1/\delta) + \frac{1}{K} n \log^2(1/\delta) \log^5 n \log\log(1/\delta).$$

And, after splitting edges to ensure $\frac{1}{\log^2(1/\delta)}$-boundedness, the output edge count is $n \log^2(1/\delta) \log^2 n$. Then the last call to MasterCholApx runs in time $n \frac{1}{K} \log^2(1/\delta) \log n$.

□

## 3.4 Approximating Schur Complements using Approximate Gaussian Elimination

Given a Laplacian $\mathbf{U}$ and a subset $C$ of its vertices, we use WeakSchurApx$(\mathbf{U}, C, \epsilon, \delta)$ to denote algorithm we get from calling MasterCholApx with $k = n - |C|$ using Rule (ii) for eliminations, and w.l.o.g. we assume that the indices of $\mathbf{U}$ are sorted so that the first $k$ indices correspond to the vertices of $V \setminus C$. Finally, we define WeakSchurApx to only output $\widetilde{\mathbf{S}}^{(k)}$. As an immediate corollary of Theorem 3.1.1 with these parameter settings, we get the following.

**Corollary 3.4.1.** *Suppose* $G = (V, E)$ *is a connected undirected multi-graph with positive edge weights* $w : E \to \mathbb{R}_+$, *and associated Laplacian* $\mathbf{U}$, *and that* $G$ *has* $m$ *multi-edges. Given a subset* $C \subseteq V$ *of the vertices of* $\mathbf{U}$, *and scalars* $\delta < 1/n^{100}$, $0 < \epsilon \leq 1/2$, *the algorithm* WeakSchurApx$(\mathbf{U}, C, \epsilon, \delta)$ *returns an approximate partial Cholesky decomposition* $\widehat{\boldsymbol{\mathcal{L}}}, \widehat{\mathbf{S}}$ *onto set* $C$ *s.t. with probability at least* $1 - O(\delta)$,

$$\widehat{\mathbf{S}} + \widehat{\boldsymbol{\mathcal{L}}}\widehat{\boldsymbol{\mathcal{L}}}^\top \approx_\epsilon \mathbf{U}. \tag{3.12}$$

*The maximum number of non-zero entries in* $\widehat{\boldsymbol{\mathcal{L}}}$ *and* $\widehat{\mathbf{S}}$ *and the total running time are each bounded by* $O(\epsilon^{-2} m \log^2(1/\delta) \log n)$.

We also need the following observation about the output $\widetilde{\mathbf{S}}^{(k)}$

**Claim 3.4.2.** *Writing* $\mathbf{U}^{(k)} = \widetilde{\mathbf{S}}^{(k)} + \widehat{\boldsymbol{\mathcal{L}}}\widehat{\boldsymbol{\mathcal{L}}}^{\top}$, *we have*

$$\mathrm{Sc}\left[\mathbf{U}^{(k)}\right]_F = \widetilde{\mathbf{S}}^{(k)}.$$

*Proof.* This claim is very simple to prove: Using the one-by-one elimination procedure for computing a Schur complement as described in Section 2.3.4, we get that the rank 1 subtraction in the $i$th step when computing $\mathrm{Sc}\left[\mathbf{U}^{(k)}\right]_F$ is exactly $\boldsymbol{c}_i\boldsymbol{c}_i^{\top}$. □

By combining Corollary 3.4.3, Claim 3.4.2, and Claim 2.3.3, we immediately get the result below.

**Corollary 3.4.3.** *Suppose* $G = (V, E)$ *is a connected undirected multi-graph with positive edge weights* $w : E \to \mathbb{R}_+$, *and associated Laplacian* $\mathbf{U}$, *and that* $G$ *has* $m$ *multi-edges. Let* $F \subseteq V$ *be a subset of the vertices of* $\mathbf{U}$ *with* $|F| = k$. *Given scalars* $\delta < 1/n^{100}$, $0 < \epsilon \leq 1/2$, *the Laplacian matrix* $\widehat{\mathbf{S}}$ *returned by the algorithm* WeakSchurApx$(\mathbf{U}, \epsilon, \delta)$ *satisfies* $\widehat{\mathbf{S}} \approx_\epsilon \mathrm{Sc}[\mathbf{U}]_F$, *with probability at least* $1 - O(\delta)$. *The maximum number of non-zero entries in* $\widehat{\boldsymbol{\mathcal{L}}}$ *and* $\widehat{\mathbf{S}}$ *and the total running time are each bounded by* $O(\epsilon^{-2} m \log^2(1/\delta) \log n)$.

Finally, we want to develop a version of the Schur approximation algorithm where the $\epsilon^{-2}$ factor in the running time multiplies a factor of $n$ rather than $m$. This will later be important when we develop an algorithm for sampling random spanning trees.

We can achieve this and improved sparsity of the output by combining the routine with a sparsification algorithm. We use a slightly simpler routine than SubsamplingGraphSparsify from Section 4.1, since we will not worry too much about the number of logs in the running time.

We use GraphSparsify to refer to the leverage score estimation procedure of Spielman and Srivastava [SS11c]. Stated for Laplacian solvers with solving time equal to their construction time, the guarantees of the algorithm are described in the following theorem.

**Theorem 3.4.4.** *Let* LapSolver *be a Laplacian linear system solver s.t.* LapSolver$(\mathbf{U}, \boldsymbol{b}, \epsilon, \delta)$ *returns* $\boldsymbol{x}$ *s.t. with probability* $1 - O(\delta)$, $\|\boldsymbol{x} - \mathbf{U}^{+}\boldsymbol{b}\|_L \leq \epsilon \|\mathbf{U}^{+}\boldsymbol{b}\|_{\mathbf{U}}$. *Assume* LapSolver *runs in time* $T(m, \epsilon, \delta)$.

*Then the algorithm* GraphSparsify(LapSolver, $\mathbf{U}, \epsilon, \delta$) *returns* $\widetilde{\mathbf{U}}$ *with* $O(\epsilon^{-2} n \log(1/\delta))$ *edges s.t. with probability* $1 - O(\delta)$

$$\widetilde{\mathbf{U}} \approx_\epsilon \mathbf{U}.$$

*The algorithm runs in time* $O(n\epsilon^{-2}\log(1/\delta) + m\log(1/\delta) + T(m, 1/2)\log(1/\delta))$.

By combining WeakSchurApx with GraphSparsify and CholApx we get a Schur complement approximation, SchurApx, stated as Algorithm 4.

**Algorithm 4:** SchurApx$(\mathbf{U}, F, \epsilon, \delta)$

```
1 Ũ ← GraphSparsify(CholApx, U, ε/3, δ)
2 L̂, Ŝ ← WeakSchurApx(Ũ, F, ε/3, δ)
3 Ŝ ← GraphSparsify(CholApx, U, ε/3, δ)
4 return Ŝ
```

The performance of this algorithm is characterized by the following theorem.

**Theorem 1.2.5** (Approximation of Laplacian Schur Complements)**.** *Suppose* $\mathbf{U} \in \mathbb{R}^{n \times n}$ *is a Laplacian matrix with* $m$ *non-zero entries. Given a set vertices* $C \subset V$, *and scalars* $0 < \epsilon \leq 1/2$, $0 < \delta < 1$, *the algorithm* SchurApx$(\mathbf{L}, C, \epsilon, \delta)$ *returns a Laplacian matrix* $\widetilde{\mathbf{S}}$. *With probability* $\geq 1 - O(\delta)$, *the following statements all hold:* $\widetilde{\mathbf{S}} \approx_\epsilon \mathrm{Sc}[\mathbf{U}]_C$. $\widetilde{\mathbf{S}}$ *is a Laplacian matrix whose edges are supported on* $C$. *Let* $k = |C| = n - |F|$. *The total number of non-zero entries* $\widetilde{\mathbf{S}}$ *is* $O(k\epsilon^{-2}\log(n/\delta))$. *The total running time is bounded by* $O((m \log n \log^2(n/\delta) + n\epsilon^{-2} \log n \log^4(n/\delta))\,\mathrm{polyloglog}(n))$.

*Proof.* The proof is immediate from combining Corollary 3.4.3, Theorem 3.4.4 and Corollary 1.2.2. □

# Chapter 4

# Sampling Random Spanning Trees

We present an algorithm that, with high probability, generates a random spanning tree from an edge-weighted undirected graph in $\widetilde{O}(\max\left\{n^{4/3}m^{1/2}, n^2\right\})$ time. The tree is sampled from a distribution where the probability of each tree is proportional to the product of its edge weights. This improves upon the previous best algorithm due to Colbourn et al. that runs in matrix multiplication time, $O(n^\omega)$. For the special case of unweighted graphs, this improves upon the best previously known running time of $\tilde{O}(\min\{n^\omega, m\sqrt{n}, m^{4/3}\})$ for $m \gg n^{7/4}$ (Colbourn et al. '96, Kelner-Madry '09, Madry et al. '15).

Our algorithm samples edges according to their conditional effective resistance as in [HX16]. We repeatedly use the well known fact that the effective resistance multiplied by the edge weight, which we will refer to as the *leverage score* of the edge, is equal to the probability that the edge belongs to a randomly generated spanning tree. To generate a uniformly random spanning tree, one can sample edges in an iterative fashion. In every iteration, the edge being considered is added to the spanning tree with probability exactly equal to its leverage score. If it is added to the tree, the graph is updated by contracting that edge, otherwise, the edge is removed from the graph. Though using fast Laplacian solvers [ST14a] one can compute the leverage score of a single edge in $\widetilde{O}(m)$ time, since one needs to potentially do this $m$ times (and the graph keeps changing every iteration), this can take $\widetilde{O}(m^2)$ time if done in a naive way. It therefore becomes necessary to compute the leverage scores in a more clever manner.

The algorithms in [CDN89, HX16] get a speed up by a clever recursive structure which enables one to work with much smaller graphs to compute leverage scores at the cost of building such a structure. This kind of recursion will be the starting point of our algorithm which will randomly partition the vertices into two equally sized sets, and compute Schur complements onto each of the set. We crucially use the fact that Schur complement, which can be viewed as block Gaussian elimination, preserves effective resistances of all the edges whose incident vertices are not eliminated. We first recursively sample edges contained in both these sets, contracting or deleting every edge along the way, and then the edges that go across the partition is sampled. Algorithm in [HX16] is essentially this, and they prove that it takes $O(n^\omega)$.

In order to improve the running time, the main workhorse we use is the Schur complement approximation routine from Section 3.4. Since we compute approximate Schur complements, the leverage scores of edges are preserved only approximately. But we set the error parameter such that we can get a better estimate of the leverage score if we move up the recursion tree, at the cost of paying more for the computing leverage score of an edge in a bigger graph. We give a sampling procedure that samples edges into the random spanning tree from the true distribution by showing that approximate leverage score can be used to make the right decisions most of the times.

Subsequently, we are presented with a natural trade-off for our error parameter choice in the SchurApx routine: larger errors speed up the runtime of SchurApx, but smaller errors make moving up the recursion to obtain a more exact effective resistance estimate less likely. Furthermore, the recursive construction will cause the total vertices across each level to double making small error parameters even more costly as we recurse down. Our choice of the error parameter will balance these trade-offs to optimize running time.

The routine SchurApx produced an approximate Schur complement only with high probability. We are not aware of a way to certify that a graph sparsifier is good quickly. Therefore, we condition on the event that the SchurApx produces correct output on all the calls, and show ultimately show that it is true with high probability.

Our algorithm for approximately generating random spanning trees, along with a proof of Theorem 1.2.6 is given in Section 4.1 and 20.

## 4.1 Algorithm for Sampling Spanning Trees

It is well known that for any edge of a graph, the probability of that edge appearing in a random spanning tree is equal to it's leverage score. We can iteratively apply this fact to sample a $\boldsymbol{w}$-uniform random tree. We can consider the edges in an arbitrary sequential order, say $e_1, ..., e_m \in E$, and make decisions on whether they belong to tree. Having decided for edges $e_1, ..., e_i$, one computes the probability $p_{i+1}$, conditional on the previous decisions, that edge $e_{i+1}$ belongs to the tree. Edge $e_{i+1}$ is then added to the tree with probability $p_{i+1}$.

To estimate the probability that edge $e_{i+1}$ belongs to the tree conditional on the decisions made on $e_1, ..., e_i$, we can use Fact 2.4.7. Let $E_T \subset \{e_1, ..., e_i\}$ be the set of edges that were included in the tree, and $F_T \subset \{e_1, ..., e_i\}$ the subset of edges that were not included. Then, $p_{i+1}$ is equal to the leverage score of edge $e_{i+1}$ in the graph $G^{(i+1)} := (G \backslash F_T) / E_T$ obtained by deleting edges $E_T$ from $G$ and then contracting edges $E_{T^c}$. In other words, we get $G^{(i+1)}$ from $G^{(i)}$ by either deleting the edge $e_i$ or contracting it, depending on if $e_i$ was not added to the tree or added to the tree, respectively. Note that as we move along the sequence, some of the original edges may no longer exist in the updated graph due to edge contractions. In that case, we just skip the edge and move to the next one.

Computing leverage score of an edge, with $\epsilon$ multiplicative error, requires $\widetilde{O}(m \log 1/\epsilon)$ runtime. Since we potentially have to compute leverage score of every edge, this immediately gives a total runtime of $\widetilde{O}(m^2)$.

Our algorithm will similarly make decisions on edges in a sequential order. Where it differs from the above algorithm is the graph we use to compute the leverage score of the edge. Instead of computing the leverage score of an edge in the original graph updated with appropriate contractions and deletions, we deal with potentially much smaller graphs containing the edge such that the effective resistance of the edge in the smaller graph is approximately same as in the original graph. In the next section, we describe the sampling procedure that we use to sample from the true distribution, when we have access to a cheap but approximate routine to compute the sampling probability.

### 4.1.1 Structure of the Recursion

We now describe the recursive structure of the algorithm given in Algorithm 5. The structure of the recursion is same as in [HX16]. Let the input graph be $G = (V_G, E_G)$. Suppose at some stage of the algorithm, we have a graph $\widetilde{G}$. The task is to make decisions on edges in $E_G \cap E_{\widetilde{G}}$. We initially divide the vertex set into two equal sized sets $V_{\widetilde{G}} = V_1 \cup V_2$. Recursively, we first make decisions on

edges in $\widetilde{G}(V_1) \cap E_G$, then make decisions on edges in $\widetilde{G}(V_2) \cap E_G$ and finally make decisions on the remaining edges. To make decisions on $\widetilde{G}(V_1) \cap E_G$, we use the fact that the effective resistance of edges are preserved under Schur complement. We work with the graph $G_1 = \textsc{SchurApx}(\widetilde{G}, V_1, \epsilon)$ and recursively make decisions on edges in $E_G \cap G(V_1)$. Having recursively made decisions on edges in $E_G \cap \widetilde{G}(V_1)$, let $E_T$ be the set of tree edges from this set. We now need to update the graph $\widetilde{G}$ by contracting edges in $E_T$ and deleting all the edges in $E_T^c \cap \widetilde{G}(V_1) \cap E_G$. Then we do the same for the edges in $E_G \cap \widetilde{G}(V_2)$.

Finally, we treat the edges $E_G \cap (V_1, V_2)$ that cross $V_1, V_2$ in a slightly different way, and is handled by the subroutine SampleAcross in the algorithm. If we just consider the edges in $E_G$, this is trivially a bipartite graph. This property is maintained in all the recursive calls by the routine SampleAcross. The routine SampleAcross works by dividing $V_1, V_2$ both into two equal sized sets $V_1 = L_1 \cup L_2$ and $V_2 = R_1 \cup R_2$ and making four recursive calls, one each for edges in $E_G \cap (L_i, R_j), i = 1, 2; j = 1, 2$. To make decisions on edges in $E_G \cap (L_i, R_j)$, it recursively calls SampleAcross on the graph $G_{ij} = \textsc{SchurApx}(\widetilde{G}, (L_i, R_j)^c, \epsilon)$ obtained by computing approximate Schur complement on to vertices in $(L_i, R_j)$ of vertices outside it.

### Exact Schur Complement and $O(n^\omega)$ Time Algorithm

Here we note how we can get a $O(n^\omega)$ algorithm. Note that this is very similar to the algorithm and analysis in [HX16]. If in SchurApx calls, we set $\epsilon = 0$, i.e., we compute exact Schur complements, then we have a $O(n^\omega)$ algorithm. Whenever we make a decision on an edge by instantiating SampleEdge$(e)$, we just have to compute the leverage score $l_e$ of the edge $e$ in a constant sized graph. This can be done in constant time and since we do exact Schur complements, $l_e = l_e(G)$. We can therefore use this to decide if $e$ belongs to the tree and then update the graph by either contracting the edge or deleting it depending on if it is included or excluded in the tree. In a graph with $n_1$ vertices, it takes $O(n_1^\omega)$ time to compute the Schur complement. Let $T(n)$ be the time taken by SampleWithin on a graph of size $n$ and $B(n)$ be the time taken by SampleAcross when called on a graph of size $n$. We then have the following recursion

$$T(n) = 2T(n/2) + B(n) + O(n^\omega)$$
$$B(n) = 4B(n/2) + O(n^\omega).$$

We therefore have $T(n) = O(n^\omega)$.

### Approximate Schur Complement and Expected $\widetilde{O}(\max\{n^{4/3}m^{1/2}, n^2\})$ Time Algorithm

We speed up $O(n^\omega)$ algorithm by computing approximate Schur complements faster. Having access only to approximate Schur complements, which preserves leverage score only approximately, introduces an issue with computing sampling probability. It is a-priori not clear how to make decisions on edges when we preserve leverage scores only approximately during the recursive calls. The key idea here is as follows. Suppose we want to decide if a particular edge $e$ belongs to the tree. Tracing the recursion tree produced by Algorithm 5, we see that we have a sequence of graphs $G, G_1, G_2, ..., G_k$ all containing the edge $e$, starting from the original input graph $G$ all the way down to $G_k$ which has a constant number of vertices. We also have $V(G_i) \subset V(G_{i-1})$ for all $k \geq i \geq 1$, all of them being subsets of $V(G)$.

Let $n = |V(G)|, m = |E_G|$ be the number of vertices and edges in the input graph, When setting the error parameters, we choose $\epsilon$ and some threshold values in ways that depend on whether $m \leq n^{4/3}$ holds. In the case $m > n^{4/3}$, we define $\epsilon$ in terms of the level $i$ as

$$\epsilon(i) = 2^{i/2} n^{-1/6} m^{-1/4} \log^{-2} n. \tag{4.1}$$

In the case $m \leq n^{4/3}$, we define $\epsilon$ in terms of the level $i$ as

$$\epsilon(i) = 2^{i/2} n^{-1/2} \log^{-2} n. \tag{4.2}$$

The threshold value is $t_1$ is such that $2^{2t_1} = \frac{n^2}{m}$.

Our $\epsilon(\cdot)$ function will ensures for all $i$, $l_e(G) \in [(1-\epsilon_i)l_e(G_i), (1+\epsilon_i)l_e(G_i)]$ for an appropriate $\epsilon_i$. We sample a uniform random number $r \in [0,1]$, and initially compute $l_e(G_k)$. If $r$ lies outside the interval $[(1-\epsilon_i)l_e(G_k), (1+\epsilon_i)l_e(G_k)]$, then we can make a decision on the edge $e$. Otherwise, we estimate $l_e(G)$ to a higher accuracy by computing $l_e(G_{k-1})$. We continue this way, and if $r$ lies inside the interval $[(1-\epsilon_i)l_e(G_i), (1+\epsilon_i)l_e(G_i)]$ for every $i$, then we compute $l_e(G)$ in the input graph $G$. In the next section we describe SampleEdge in more detail.

At this point, we find it important to mention that the spectral error guarantees from the SchurApx subroutine only hold with probability $\geq 1 - O(\delta)$. The explanation of the SampleEdge subroutine above relied on these spectral guarantees, and the error in our algorithm for generating random spanning trees will be entirely due to situations in which the sparsification routine does not give a spectrally similar Schur complement. For the time being we will work under the following assumption and later use the fact that it is true w.h.p. to bound the error of our algorithm.

**Assumption 4.1.1.** *Every call to* SchurApx *with error parameter $\epsilon$ always computes an $\epsilon$-approximate Schur Complement.*

### Sampling Scheme: SampleEdge

In this section we describe the routine SampleEdge($e$) for an edge $e \in G$ in the input graph. By keeping track of the recursion tree, we have $G_0, G_1, ..., G_k$ and $e \in G_i$ for all $i$.

**Lemma 4.1.2.** *For graph $G$ and $G_i$, the respective conditional leverage scores $l_e$ and $l_e^{(i)}$ for edge $e$ are such that $l_e \in [(1-2\epsilon(i)\log n)l_e^{(i)}, (1+2\epsilon(i)\log n)l_e^{(i)}]$*

This will now allow us to set $\epsilon_i = 2\epsilon(i)\log n$. We will delay the proof of Lemma 4.1.2 until later in this section in favor of first giving the sampling procedure. The sampling procedure is as follows. We generate a uniform random number in $r \in [0,1]$. We want to sample edge $e$ if $r \leq l_e(G)$. Instead, we use $l_e(G_k)$ as a proxy. Note that using fast Laplacian solvers, we can in $\widetilde{O}$(no. of edges) time compute leverage score of an edge upto a factor of $1 + 1/\text{poly}(n)$. Since $l_e(G) \in [(1-\epsilon_k)l_e(G_k), (1+\epsilon_k)l_e(G_k)]$, we include the edge in the tree if $r \leq (1-\epsilon_k)l_e(G_k)$, otherwise if $r > (1+\epsilon_k)l_e(G_k)$, we don't include it in the tree. If $r \in [(1-\epsilon_k)l_e(G_k), (1+\epsilon_k)l_e(G_k)]$, which happens with probability $2\epsilon_k l_e(G_k)$, we get a better estimate of $l_e(G)$ by computing $l_e(G_{k-1})$. We can make a decision as long as $r \notin [(1-\epsilon_{k-1})l_e(G_{k-1}), (1+\epsilon_{k-1})l_e(G_{k-1})]$, otherwise, we consider the bigger graph $G_{k-2}$. In general, if $r \notin [(1-\epsilon_i)l_e(G_i), (1+\epsilon_i)l_e(G_i)]$, then we can make a decision on $e$, otherwise we get a better approximation of $l_e(G)$ by computing $l_e(G_{i-1})$. If we can't make a decision in any of the $k$ steps, which happens if $r \in [(1-\epsilon_i)l_e(G_i), (1+\epsilon_i)l_e(G_i)]$ for all $i$, then we compute the leverage score of $e$ in $G$ updated with edge deletions and contractions resulting from decisions made on all the edges that were considered before $e$.

Note that when we fail to get a good estimate at level $i$ for some $i \geq t_1$, we always compute the next estimate with respect to the original graph.

Finally, note that in the final step, we can compute $l_e(G)$ up to an approximation factor of $1+\rho$ in $\widetilde{O}(m \log 1/\rho)$. We can therefore start with $\delta_0 = 1/n$ and if $r \in [(1-\rho)\widetilde{l}_e(G), (1+\rho)\widetilde{l}_e(G)]$, we set $\rho = \rho_0/2$ and repeat. This terminates in $\widetilde{O}(m)$ expected (over randomness in $r$) time.

For our algorithm, assume that we have an efficient data structure that gives access to each graph $G_0, ...G_k$ in which $e$ appears.

**Algorithm 5:** GENERATESPANNINGTREE($\widetilde{G} = (E_{\widetilde{G}}, \widetilde{V})$) : Recurse using Schur Complement

**Input:** Graph $\widetilde{G}$. Let $E_G$, a global variable, denote the edges in the original (input) graph $G$.
**Output:** $E_T$ is the set of edges in the sampled tree.

$E_T \leftarrow$ SAMPLEWITHIN($G$)
**return** $E_T$
**Procedure** SAMPLEWITHIN($\widetilde{G}$)
Set $E_T \leftarrow \{\}$
**if** $|\widetilde{V}| = 1$ **then**
    **return**
**else**
    Divide $V$ into equal sets $V = V_1 \cup V_2$.
    **for** $i = 1, 2$ **do**
        Compute $G_i =$ SCHURAPX($\widetilde{G}, V_i, \epsilon$(level)) (see Equations (4.2) and (4.1))
        $E_T \leftarrow E_T \cup$ SAMPLEWITHIN($G_i$)
        Update $\widetilde{G}$ by deleting edges in $\widetilde{G}(V_i) \cap E_T^c$ and contracting edges in $\widetilde{G}(V_i) \cap E_T$. (Note the convention $E_T^c := E_G \backslash E_T$ )
    $E_T \leftarrow E_T \cup$ SAMPLEACROSS($\widetilde{G}, (V_1, V_2)$)
    **return** $E_T$
**Procedure** SAMPLEACROSS($\widetilde{G}, (L, R)$) **if** $|L| = |R| = 1$ **then**
    $E_T =$ SAMPLEEDGE($\widetilde{G}, (L, R) \cap E_G$)
    **return** $E_T$
Divide $L, R$ into two equal sized sets: $L = L_1 \cup L_2$, $R = R_1 \cup R_2$.
**for** $i = 1, 2$ **do**
    **for** $j = 1, 2$ **do**
        $\widetilde{G}_{ij} \leftarrow$ SCHURAPX($\widetilde{G}, (L_i \cup R_j), \epsilon$(level)) (see Equations (4.2) and (4.1))
        $E_T \leftarrow E_T \cup$ SAMPLEACROSS($\widetilde{G}_{ij}, (L_i, R_j)$)
        Update $\widetilde{G}$ by contracting edges $E_T$ and deleting edges in $E_T^c \cap (L_i, R_j)$
**return** $E_T$

**Algorithm 6:** SAMPLEEDGE($e$) : Sample an edge using conditional leverage score

**Input:** An edge $e$ and access to graphs $G_0, ...G_k$ in which $e$ appears
**Output:** Returns $\{e\}$ if edge belongs to the tree, and $\{\}$ if it doesn't

Generate a uniform random number $r$ in $[0, 1]$
$l_e \leftarrow$ ESTIMATELEVERAGESCORE($e$)
**if** $r < l_e$ **then**
    **return** $\{e\}$
**else**
    **return** $\{\}$

**Procedure** ESTIMATELEVERAGESCORE($e$) Compute $l_e^{(k)}$ to error $1/n$
**if** ISGOOD($l_e^{(k)}, \epsilon$) **then**
    **return** $l_e^{(k)}$
**for** $i = t_1$ *to* $\log n$ **do**
    Compute $l$, an estimate for $l_e^{(i)}$ with error $1/n$
    **if** ISGOOD($l, \epsilon(i)$) **then**
        **return** $l$
**for** $i = 0$ *to* $\infty$ **do**
    Compute $l$, an estimate for $l_e^{(0)}$ with error $2^{-i}n$
    **if** ISGOOD($l, 2^{-i}n$) **then**
        **return** $l$

**Procedure** ISGOOD($l_e, \epsilon$) **if** $r < (1-\epsilon)l_e$ *or* $r > (1+\epsilon)l_e$ **then**
    **return** *True*
**return** *False*

### Proof of Lemma 4.1.2

This edge sampling scheme relies upon the error in the leverage score estimates remaining small as we work our way down the subgraphs and remaining small when we contract and delete edges. Theorem 1.2.5 implies leverage score estimates will have small error between levels, so we will only have compounding of small errors. However, it does not imply that these errors remain small after edge contractions and deletions, which becomes necessary to prove in the following lemma.

**Lemma 4.1.3.** *Given a graph $G = (V, E)$, vertex partition $V_1, V_2$, and edges $e \in E \cap (V_1, V_1)$, then*

$$\textsc{SchurApx}(G, V_1, \epsilon)/e \approx_\epsilon \textsc{Sc}[G/e]_{V_1}, \textsc{SchurApx}(G, V_1, \epsilon) \setminus e \approx_\epsilon \textsc{Sc}[G \setminus e]_{V_1}$$

*Proof.* $\textsc{SchurApx}(G, V_1, \epsilon)/e \approx_\epsilon \textsc{Sc}[G]_{V_1}/e$ because spectral approximations are maintained under contractions. Furthermore, $\textsc{SchurApx}(G, V_1, \epsilon) = \mathbf{U}_{V_1} + \tilde{S_{V_2}}$ where $\mathbf{U}_{V_1}$ is the Laplacian of the edges in $E \cap (V_1, V_1)$. Similarly, write $\textsc{Sc}[G]_{V_1} = \mathbf{U}_{V_1} + \mathbf{S}_{V_2}$, and because $\tilde{\mathbf{S}}_{V_2} \approx_\epsilon \mathbf{S}_{V_2}$ then $\mathbf{U}_{V_1} \setminus e + \tilde{\mathbf{S}}_{V_2} \approx_\epsilon \mathbf{U}_{V_1} \setminus e + \mathbf{S}_{V_2}$. Combining these facts with Fact 2.4.8 gives the desired result. □

*Proof.* (of Lemma 4.1.2)

By construction, $\epsilon(i) \leq \epsilon(k)$ for every $i \leq k$. Iteratively applying Theorem 1.2.5 and Lemma 4.1.3, gives $l_e \in [e^{-\epsilon(k)k} l_e^{(k)}, e^{\epsilon(k)k} l_e^{(k)}]$, and using $\epsilon(k) \leq 1/\log^2 n$ for all $k$, and $k \leq \log n$ finishes the proof

□

### Correctness

Under Assumption 4.1.1, we were able to prove Lemma 4.1.2. This, in turn, implies the correctness of our algorithm, which is to say that it generates a tree from a $\boldsymbol{w}$-uniform distribution on trees. We now remove Assumption 4.1.1, and prove the approximate correctness of our algorithm, and the first part of Theorem 1.2.6.

**Theorem 1.2.6** (Sampling Random Spanning Trees)**.** *For any $0 < \delta < 1$, the routine* GenerateSpanningTree *(Algorithm 5) outputs a random spanning tree from the $\boldsymbol{w}$-uniform distribution with probability at least $1 - \delta$ and takes expected time $\widetilde{O}(\max\{n^{4/3}m^{1/2}, n^2\}\log^4(1/\delta))$.*

*Proof.* Each subgraph makes at most 6 calls to SchurApx, and there are $\log n$ recursive levels, so $O(n^3)$ total calls are made to SchurApx. Setting $\delta' = \frac{\delta}{O(n^3)}$ for each call to SchurApx, Assumption 4.1.1 holds with probability $(1-\delta')^{O(n^3)} = 1 - \delta$, and $\log^4\left(\frac{O(n^3)}{\delta}\right) = \tilde{O}(\log^4(1/\delta))$. Therefore, our algorithm will only fail to generate a random tree from the $\boldsymbol{w}$-uniform distribution on trees with probability at most $\delta$.

□

### Runtime Analysis

We will now analyze the runtime of the algorithm. Let $T(n)$ be the time taken by SampleWithin on input a graph $\widetilde{G}$ with $n$ vertices and let $B(n)$ be the time taken by SampleAcross on a graph with $n$ vertices. We recall that the recursive structure then gives $T(n) = 2T(n/2) + 4B(n/2)$ and $B(n/2) = 4B(n/4)$. To compute the total runtime, we separate out the work done in the leaves of the recursion tree from the rest. Note that SampleEdge is invoked only on the leaves.

First we bound the total number of nodes of the recursion tree as a function of the depth in the tree.

**Lemma 4.1.4.** *Level $i$ of the recursion tree has at most $4^{i+1} - 2^i$ nodes, the number of vertices in the graphs at each of the nodes is at most $n/2^i$.*

*Proof.* It is clear that the size of the graph at a node at depth $i$ is at most $n/2^i$. We will bound the number of nodes by induction. There are two types of nodes in the recursion tree due to the recurrence having two kinds of branches corresponding to $T(n), B(n)$. We will call the nodes corresponding to $T(n)$ as the first type and it is clear from the recurrence relation that there are $2^i$ such nodes. Let us call the other type of nodes the second type, and it is clear that every node (both first and second type) at depth $i-1$ branches into four type two nodes. Therefore, if $a_i$ is the total number of nodes at level $i$, then $a_i = 4a_{i-1} + 2^i$. We will now prove by induction that $a_i \leq 4^{i+1} - 2^i$. Given $a_0 = 1$, the base case follows trivially. Suppose it is true for $i-1$, then we have $a_i = 4a_{i-1} + 2^i \leq 4(4^i - 2^{i-1}) + 2^i = 4^{i+1} - 2^i$, proving the lemma.

□

Now we will compute the total work done at all levels other than the leaves. We recall the error parameter in SchurApx calls is a function of the depth in the tree: In the case $m > n^{4/3}$, we define $\epsilon$ in terms of the level $i$ as

$$\epsilon(i) = 2^{i/2} n^{-1/6} m^{-1/4} \log^{-2} n.$$

In the case $m \leq n^{4/3}$, we define $\epsilon$ in terms of the level $i$ as

$$\epsilon(i) = 2^{i/2} n^{-1/2} \log^{-2} n.$$

Note that when $m > n^{4/3}$, we have $n^{-1/6} m^{-1/4} < n^{-1/2}$. Further, the maximum value of $i$ is $\log n$ so $2^{i/2} = n^{1/2}$. This means we always have $\epsilon(i) \leq 2^{i/2} n^{-1/2} \log^{-2} n \leq \log^{-2} n$.

The threshold value $t_1$ is such that $2^{2t_1} = \frac{n^2}{m}$.

**Lemma 4.1.5.** *The total work done at all levels of the recursion tree excluding the leaves is bounded by $\widetilde{O}(\max\left\{n^{4/3}m^{1/2}, n^2\right\} \log^4(1/\delta))$.*

*Proof.* From Theorem 1.2.5 the work done in a node at depth $i$ is $\widetilde{O}\left(((n/2^i)^2 + n2^{-i}\epsilon(i)^{-2}) \log^4(1/\delta)\right)$. The $\log^4(1/\delta)$ factor is left out from the remaining analysis for simplicity. By Lemma 4.1.4, the total work done at depth $i$ is $\widetilde{O}\left(n^2 + n2^i\epsilon(i)^{-2}\right)$. Finally, bound for the total running time across all levels follows from

$$\sum_{i=0}^{\log n} n^2 + 2^i \frac{n}{\epsilon(i)^2} = \widetilde{O}\left(n^2 \log n + n \max\left\{n^{1/3}m^{1/2}, n\right\}\right).$$

□

We will now analyze the total work done at the leaves of the recursion tree. We first state a lemma which gives the probability that approximate leverage score of an edge can be used to decide if the edge belongs to the tree.

**Corollary 4.1.6.** *If $r$ is drawn uniformly randomly from $[0,1]$, then the probability that $r \in [1 - \hat{\epsilon}\hat{l_e} \log^2 n, 1 + \hat{\epsilon}\hat{l_e} \log^2 n]$ is $\widetilde{O}(\hat{\epsilon}l_e)$ w.h.p.*

*Proof.* The exact probability is $2\hat{\epsilon}\hat{l_e} \log^2 n$, and from Lemma 5.8, we know $l_e \leq 2\hat{l_e}$ w.h.p.

□

We now consider the expected work done at a single leaf of the recursion tree.

**Lemma 4.1.7.** *Let $l_e$ be the leverage score of an edge $e$ in the $G$ which is obtained by updating the input graph based on the decisions made on all the edges considered before $e$. The routine* SampleEdge *takes*

$$\widetilde{O}\left(1 + l_e \max\left\{n, n^{1/3}m^{1/2}\right\}\right).$$

*Proof.* It takes $O(1)$ time to compute the leverage score at a leaf of the recursion tree. The routine SampleEdge successively climbs up the recursion tree to compute the leverage score if the leverage score estimation at the current level is not sufficient. The probability that the outcome of $r$ is such that we cannot make a decision at level $i$ is $\widetilde{O}(\epsilon(i)l_e)$.

The time required to compute the leverage score of edge $e$ in the graph at a node at depth $i$ in the recursion tree is $\widetilde{O}((n/2^i)^2)$.

Finally, with probability $\widetilde{O}(\epsilon(t_1)l_e)$ we need to compute the leverage score in the input graph and the expected running time is $\widetilde{O}(m)$. Therefore, when $m > n^{4/3}$ and $\epsilon(i) = 2^{i/2}n^{-1/6}m^{-1/4}\log^{-2} n$, the total expected running time is

$$\begin{aligned}\widetilde{O}\left(1 + m\epsilon(t_1)l_e + l_e\sum_{i=t_1}^{i=\log n}\epsilon(i)\frac{n^2}{4^i}\right) &= \widetilde{O}\left(1 + l_e m n^{-1/6}m^{-1/4}n^{1/2}m^{-1/4}\right)\\ &= \widetilde{O}\left(1 + l_e n^{1/3}m^{1/2}\right).\end{aligned}$$

When $m \leq n^{4/3}$ and $\epsilon(i) = 2^{i/2}n^{-1/2}\log^{-2} n$, the total expected running time is

$$\begin{aligned}\widetilde{O}\left(1 + m\epsilon(t_1)l_e + l_e\sum_{i=t_1}^{i=\log n}\epsilon(i)\frac{n^2}{4^i}\right) &= \widetilde{O}\left(1 + l_e m n^{-1/2}n^{1/2}m^{-1/4}\right)\\ &= \widetilde{O}\left(1 + l_e m^{3/4}\right)\\ &= \widetilde{O}\left(1 + l_e n\right).\end{aligned}$$

Note that $n^{1/3}m^{1/2} \geq n$ if and only if $m \geq n^{4/3}$, so we can summarize this as the expected running time being bounded by $\widetilde{O}\left(1 + l_e \max\left\{n, n^{1/3}m^{1/2}\right\}\right)$.

□

We now want to give the runtime cost over all edges. Let us label the edges $e_1, ..., e_m$ in the order in which the decisions are made on them. In the following, when we talk about leverage score $l_{e_i}$ of an edge $e_i$, we mean the leverage score of the edge $e_i$ in the graph obtained by updating $G$ based on the decisions made on $e_1, ..., e_{i-1}$.

**Lemma 4.1.8.** *Let $e_i$ be the first edge sampled to be in the tree, and $X = l_{e_1} + l_{e_2} + l_{e_3} + ... + l_{e_i}$ be a random variable. Then,*

$$\Pr(X > C) \leq e^{-C}.$$

*Proof.* Let $p_j = l_{e_j}$, we have $0 \leq p_j \leq 1$. If $\sum_j p_j \geq C$, then the probability that the edges $e_1, ..., e_{i-1}$ is deleted is

$$\prod_{j=1}^{i}(1 - p_j) \leq \left(1 - \frac{C}{i}\right)^i \leq e^{-C}.$$

□

We thus have $E(X) = O(1)$, and also, with probability at least $1 - 1/\text{poly}(n)$ we have $X = O(\log n)$. Applying this iteratively until $n-1$ edges are sampled to be in the tree, we have that the expected sum of conditional leverage scores is $O(n)$, and is $O(n \log n)$ with probability $1-1/\text{poly}(n)$.

**Corollary 4.1.9.** *The total expected work done over all the leaves of the recursion tree is* $\widetilde{O}(\max\left\{n^{4/3}m^{1/2}, n^2\right\})$.

*Proof.* This immediately follows from Lemma 4.1.7 by plugging in $\sum_e l_e = O(n \log n)$, which holds with probability at least $1 - 1/\text{poly}(n)$, and observing that the work done at the leaves is $\text{poly}(n)$ in the worst case.

□

# Chapter 5

# Approximate Gaussian Elimination for Directed Laplacians

In this section, we will give an overview of the key components of our algorithm for LU factorization and show how to use an LU factorization it generates to solve Eulerian Laplacians. We will also give proofs of our two top level statements on Eulerian Laplacians (Theorem 1.2.3 and Corollary 1.2.4) assuming lower level statements proven in this section and later in the chapter.

We will assume that the input matrix, $\mathbf{L}$, is a strongly connected Eulerian Laplacian. The main algorithm for LU factorization, Algorithm 8, has $p_{\max}$ phases or iterations. The routine for a single phase, given in Algorithm 7, iteratively eliminates vertices belonging to a random set selected by Algorithm 9. In every iteration within any phase of the algorithm, a vertex, say $v$, is eliminated. Let $\boldsymbol{\chi}_v \in \mathbb{R}^n$ be the vector whose $v$'th coordinate is set to one and all others to be zero, and $\boldsymbol{b}_{(u,v)} = \boldsymbol{\chi}_v - \boldsymbol{\chi}_u$. We write

$$\textsc{St}[\mathbf{L}]_v = \sum_{(v,u)\in E} w_{(v,u)} \boldsymbol{b}_{(u,v)} \boldsymbol{\chi}_v^T + \sum_{(u,v)\in E} w_{(u,v)} \boldsymbol{\chi}_v \boldsymbol{b}_{(u,v)}^T,$$

to denote the directed star-graph at vertex $v$ in $\mathbf{L}$. We can write the Schur complement obtained by eliminating vertex $v$ as

$$\textsc{Sc}[\mathbf{L}]_{V\setminus\{v\}} = \mathbf{L} - \textsc{St}[\mathbf{L}]_v + \textsc{St}[\mathbf{L}]_v - \frac{1}{\mathbf{L}(v,v)} \mathbf{L}(:,v)\mathbf{L}(v,:).$$

Since $\textsc{St}[\mathbf{L}]_v - \frac{1}{\mathbf{L}(v,v)} \mathbf{L}(:,v)\mathbf{L}(v,:)$ is in general a dense matrix, we use the routine $\textsc{SingleVertexElim}(\cdot)$, given in Subsection 5.1.2, to sparsify it at every iteration. We make $\widetilde{O}(1)$ calls to $\textsc{SingleVertexElim}(\cdot)$ at every elimination step, and this results in a slow increase in the the total number of edges at every iteration. We will use $\textsc{SparsifyEulerian}(\cdot)$ from [CKP+16b] to sparsify the current graph every once in a while based on the total edge count. This is a routine to sparsify any Eulerian graph, and can be found in [CKP+16b]. Recalling the notion of asymmetric spectral approximation from Definition 2.1.3, the guarantees stated in Theorem 3.16 of [CKP+16b] can be stated as follows:

**Theorem 5.0.1.** *For Eulerian Laplacian $\mathbf{L} \in \mathbb{R}^{n\times n}$ and $\epsilon, \delta, \in (0,1)$ with probability at least $1-p$ the routine $\textsc{SparsifyEulerian}(\mathbf{L}, \delta, \epsilon)$ computes in $\widetilde{O}(\mathrm{nnz}(\mathbf{L}) + n\epsilon^{-2}\log(1/\delta))$ time an Eulerian Laplacian $\widetilde{\mathbf{L}} \in \mathbb{R}^{n\times n}$ such that*

1. *$\widetilde{\mathbf{L}}$ is an $\epsilon$-asymmetric spectral approximation of $\mathbf{L}$.*

2. $\widetilde{\mathbf{L}}$ *has* $\widetilde{O}(n\epsilon^{-2}\log(1/\delta))$ *non-zeros.*

3. *the weighted in and out degrees of the graphs associated with* $\mathbf{L}$ *and* $\widetilde{\mathbf{L}}$ *are identical.*

A key thing to note is that the graph we maintain is always Eulerian. Let the vertices be labeled so that in phase $p$ of the algorithm, we eliminate vertices $i_p \dots i_{p+1} - 1$. This means that at the start of phase $p$, we have a graph on $n - i_p + 1$ vertices. We will then index the elimination steps using the superscript $^{(i)}$ to denote the state of things **just before** we make the $i^{\text{th}}$ elimination step. $\mathbf{S}^{(i)}$ denotes the matrix that we work on before the $i^{\text{th}}$ elimination step. We define $\mathbf{S}^{(i)}$ as an $n \times n$ matrix, but it is always non-zero only on entries that correspond to pairs variables neither of which have been eliminated. $\mathbf{L}^{(i)}$ denotes the original matrix perturbed by the perturbations introduced by SingleVertexElim$(\cdot)$ and SparsifyEulerian$(\cdot)$. We can express it as:

$$\mathbf{L}^{(i+1)} \stackrel{\text{def}}{=} \mathbf{L}^{(i)} + \left(\mathbf{S}^{(i+1)} - \text{Sc}\left[\mathbf{L}^{(i)}\right]_{\{i+1,\dots n-1\}}\right).$$

The quantity $\mathbf{S}^{(i)}$ is formally defined in Algorithm 8, along with an index set $i_j$ (sometimes denoted $i_p$) used to index into it for $j = 0 \dots p_{\max} - 1$, where $p_{\max}$ is the total number of phases.

Using a matrix martingale concentration inequality, we can prove the following statement about the distortions being bounded in each phase.

**Theorem 5.0.2.** *Given an* $n \times n$ *Eulerian matrix* $\mathbf{L}$ *with* $m$ *non-zeros and a* $0.1$*-RCDD subset* $J$*, and an error parameter* $\epsilon \leq 1/2$*, and probability bound* $\delta < 1/n$*,* SinglePhase$(\mathbf{L}, \delta, \epsilon)$ *(Algorithm 7) creates with probability* $1 - O(\delta)$ *matrices* $\widetilde{\mathbf{S}}, \boldsymbol{\mathcal{L}}', \boldsymbol{\mathcal{U}}'$*, where* $\widetilde{\mathbf{S}}$ *is an Eulerian Laplacian, and* $\boldsymbol{\mathcal{L}}', \boldsymbol{\mathcal{U}}'$ *are upper and lower triangular respectively. The algorithm also finds a subset* $\widehat{J}$ *such that*

$$\widetilde{\mathbf{S}} = \text{Sc}\left[\widetilde{\mathbf{L}}\right]_{V \setminus \widehat{J}}$$

*for the matrix* $\widetilde{\mathbf{L}} = \widetilde{\mathbf{S}} + \boldsymbol{\mathcal{L}}'\boldsymbol{\mathcal{U}}'$ *such that*

$$\left\|\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\left(\widetilde{\mathbf{L}} - \mathbf{L}\right)\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\right\|_2 \leq \epsilon \tag{5.1}$$

*and* $|\widehat{J}| \geq \frac{1}{2}|J|$.

*Furthermore, the number of non-zeroes in* $\widetilde{\mathbf{S}}$*,* $\boldsymbol{\mathcal{L}}'$*, and* $\boldsymbol{\mathcal{U}}'$ *will be at most* $\widetilde{O}(n\epsilon^{-6}\log^5(1/\delta))$ *and the runtime will be at most* $\widetilde{O}(m + n\epsilon^{-8}\log^7(1/\delta))$.

This Theorem is proven in Section 5.1.

This means that with high probability we can bound the distortion taking place within each phase by:

$$\left\|\mathbf{U}_{\mathbf{S}^{(i_p)}}^{\dagger 1/2}\left(\mathbf{L}^{(i_p)} - \mathbf{L}^{(i_{p+1})}\right)\mathbf{U}_{\mathbf{S}^{(i_p)}}^{\dagger 1/2}\right\| \leq \theta_p \epsilon,$$

while we also have $(n - i_{p+1}) \leq 0.99(n - i_p)$

We will now define a positive semi-definite matrix which we use for measuring the accumulation of errors:

$$\mathbf{F}^{(p)} \stackrel{\text{def}}{=} \sum_{p' \leq p} \theta_{p'} \mathbf{U}_{\mathbf{S}^{(i_p)}},$$

and for convenience, we will let $\mathbf{F}$ be

$$\mathbf{F} \stackrel{\text{def}}{=} \mathbf{F}^{(p_{\max})} = \sum_{0 \leq p < p_{\max}} \theta_p \mathbf{U}_{\mathbf{S}^{(i_p)}}. \tag{5.2}$$

We will set $\theta_p = \frac{1}{p_{\max}}$ so that $\sum_{p=0}^{p_{\max}-1} \theta_p = 1$. We will bound the final error in terms of $\mathbf{F}$ as stated in the following lemma.

**Algorithm 7:** SinglePhase($\mathbf{L}, \delta, \epsilon$)

**Input:** an Eulerian Laplacian $\mathbf{L}$ on vertex set $V$, and an error parameter $\epsilon$

**Output:** Set of vertices eliminated, $F$, $\mathbf{S}^{|F|}$ an Eulerian Laplacian on $V \setminus F$, matrices $\boldsymbol{\mathcal{U}}^{(|F|)}$, $\boldsymbol{\mathcal{L}}^{(|F|)}$ with non-zeros only in the rows/columns corresponding to $F$ respectively that are upper/lower triangle upon rearranging the vertices in $F$, and $\mathbf{L} \approx \mathbf{S}^{(|F|)} + \boldsymbol{\mathcal{L}}^{(|F|)}\boldsymbol{\mathcal{U}}^{(|F|)}$

$P \leftarrow \Theta(\log^2(1/\delta)/\epsilon^2)$

$\epsilon' \leftarrow \Theta(\epsilon/P)$

Compute a sparsifier of $\mathbf{L}$, $\mathbf{S}^{(0)} \leftarrow$ SparsifyEulerian($\mathbf{L}, \delta/P, \epsilon'$)

$T \leftarrow \widetilde{O}(n\epsilon^{-6}\log^5(1/\delta))$ (the upper bound on $\text{nnz}(\mathbf{S}^{(0)})$ from Theorem 5.0.1)

Pick a 0.1-RCDD subset (Algorithm 9) of vertices $F^{(0)}$ from $\mathbf{S}^{(0)}$

Initialize $\boldsymbol{\mathcal{U}}^{(0)}, \boldsymbol{\mathcal{L}}^{(0)} \leftarrow 0$

Set $k_{\max} \leftarrow \lfloor |F^{(0)}|/2 \rfloor$

**for** $k = 1 \ldots k_{\max}$ **do**

  Among vertices in $F^{(k-1)}$ with degree at most twice the average, pick a random $v_k$.

  $F^{(k)} \leftarrow F^{(k-1)} \setminus \{v_k\}$

  Set $d^{(k)} \leftarrow \mathbf{S}^{(k-1)}(v_k, v_k)$

  Update the factorization, set $\boldsymbol{\mathcal{U}}^{(k)}$ to $\boldsymbol{\mathcal{U}}^{(k-1)}$ with row $v_k$ replaced by $\frac{1}{d^{(k)}}\mathbf{S}^{(k-1)}(v_k, :)$ and $\boldsymbol{\mathcal{L}}^{(k)}$ to $\boldsymbol{\mathcal{L}}^{(k-1)}$ with column $v_k$ replaced by $\mathbf{S}^{(k-1)}(v_k, :)$.

  Set $\boldsymbol{l}^{(k)}$, $\boldsymbol{r}^{(k)}$ to length $n$ vectors containing the the off-diagonal non-zeros in the column and row of $v_k$ in $\mathbf{S}^{(k-1)}$ respectively.

  Initialize the first matrix of the inner loop to be the exact Schur complement of pivoting out $v_k$ from $\mathbf{S}^{(k-1)}$: $\mathbf{S}^{(k,0)} \leftarrow \mathbf{S}^{(k-1)} - \frac{1}{d^{(k)}}\boldsymbol{l}^{(k)}\boldsymbol{r}^{(k)\top}$

  **for** $t = 1 \ldots P$ **do**

    $\mathbf{S}^{(k,t)} \leftarrow \mathbf{S}^{(k,t-1)} - \frac{1}{P}\left(\text{SingleVertexElim}\left(d^{(k)}, \boldsymbol{l}^{(k)}, \boldsymbol{r}^{(k)}\right) - \frac{1}{d^{(k)}}\boldsymbol{l}^{(k)}\boldsymbol{r}^{(k)\top}\right)$, (see Algorithm 10)

  **if** $\text{nnz}(\mathbf{S}^{(k,P)}) \geq 2T$ **then**

    $\mathbf{S}^{(k)} \leftarrow$ SparsifyEulerian($\mathbf{S}^{(k,P)}, \delta/P, \epsilon'$)

  **else**

    $\mathbf{S}^{(k)} \leftarrow \mathbf{S}^{(k,P)}$

Return $\mathbf{S}^{(k_{\max})}$, $\boldsymbol{\mathcal{U}}^{(k_{\max})}$, $\boldsymbol{\mathcal{L}}^{(k_{\max})}$, and $k_{\max}$

**Algorithm 8:** EULERIANLU$(\mathbf{L}, \delta, \epsilon)$

**Input:** an Eulerian Laplacian $\mathbf{L}$ and error parameter $0 < \epsilon < 1/2$

**Output:** lower and upper triangular matrices $\boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{U}}$ whose product approximates $\mathbf{L}$

$\mathbf{L} \leftarrow$ SPARSIFYEULERIAN$(\mathbf{L}, \delta/2, O(\epsilon/\log n))$

$\mathbf{S}^{(0)} \leftarrow \mathbf{L}$ and set $\boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{U}}$ to be empty matrices.

$i \leftarrow [0]$, $j \leftarrow 0$

**while** $i_j < n$ *(i.e., for $p_{\max} = O(\log n)$ iterations)* **do**

  $(\mathbf{T}, \boldsymbol{\mathcal{L}}', \boldsymbol{\mathcal{U}}', k_{\max}) \leftarrow$ SINGLEPHASE $\left(\mathbf{S}^{(i_j)}, \frac{\delta}{2 \log n}, O(\epsilon/\log n)\right)$

  $i_j \leftarrow i_{j-1} + k_{\max}$, $\mathbf{S}^{(i_j)} \leftarrow \mathbf{T}$

  Insert the nonzero vectors from the partial LU factorization $\boldsymbol{\mathcal{L}}', \boldsymbol{\mathcal{U}}'$ into their corresponding locations $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$, respectively.

  $j \leftarrow j + 1$

**Lemma 5.0.3.** *With high probability the final accumulation of errors satisfies*

$$\left\| \mathbf{F}^{\dagger 1/2} \left( \mathbf{L} - \mathbf{L}^{(i)} \right) \mathbf{F}^{\dagger 1/2} \right\|_2 \leq \epsilon$$

*for every $i = 0 \ldots n$.*

Proof of Lemma 5.0.3 is deferred to Section A.2. This alone will not be sufficient for using $\mathbf{L}^{(n)}$ as a preconditioner. Therefore, we show the following additional guarantee on $\mathbf{F}$.

**Lemma 5.0.4.** *With high probability, the final norm that we use, $\mathbf{F}$, satisfies*

$$\mathbf{L}^{(n)\top} \mathbf{F}^{\dagger} \mathbf{L}^{(n)} \succeq 1/O(\log^2 n) \cdot \mathbf{F}.$$

Proof is deferred to Section A.2. Lemma 5.0.4 and the ability to solve linear systems in $\mathbf{L}^{(n)}$ enable us to solve linear systems in $\mathbf{L}$. To formalize this, we need to draw upon the definition of approximate pseudoinverses from [CKP+16b].

**Definition 5.0.5** (Approximate Pseudoinverse)**.** *Matrix $\mathbf{Z}$ is an $\epsilon$-*approximate pseudoinverse of matrix $\mathbf{M}$ with respect to a symmetric positive semidefinite matrix $\mathbf{F}$, *if* $\ker(\mathbf{F}) \subseteq \ker(\mathbf{M}) = \ker(\mathbf{M}^\intercal) = \ker(\mathbf{Z}) = \ker(\mathbf{Z}^\intercal)$, *and*

$$\left\| \mathbf{I}_{im(\mathbf{M})} - \mathbf{Z}\mathbf{M} \right\|_{\mathbf{F} \to \mathbf{F}} \leq \epsilon.$$ [1]

The reason why approximate pseudoinverses are useful is that if one preconditions with a solver for an approximate pseudoinverse, one can quickly solve the original system.

**Lemma 5.0.6** (Preconditioned Richardson, [CKP+16b] Lemma 4.2, pg. 30)**.** *Let $\boldsymbol{b} \in \mathbb{R}^n$ and $\mathbf{M}, \mathbf{Z}, \mathbf{F} \in \mathbb{R}^{n \times n}$ such that $\mathbf{F}$ is symmetric positive semidefinite,* $\ker(\mathbf{F}) \subseteq \ker(\mathbf{M}) = \ker(\mathbf{M}^\intercal) = \ker(\mathbf{Z}) = \ker(\mathbf{Z}^\intercal)$, *and $b \in \mathrm{im}\,(\mathbf{M})$. Then if one performs $t \geq 0$ iterative refinement steps with step size $\eta > 0$, one obtains a vector $\boldsymbol{x}_t =$* PRECONRICHARDSON$(\mathbf{M}, \mathbf{Z}, \boldsymbol{b}, \eta, t)$ *such that*

$$\left\| \boldsymbol{x}_t - \mathbf{M}^\dagger \boldsymbol{b} \right\|_{\mathbf{F}} \leq \left\| \mathbf{I}_{\mathrm{im}\,(\mathbf{M})} - \eta \mathbf{Z}\mathbf{M} \right\|_{\mathbf{F} \to \mathbf{F}}^t \left\| \mathbf{M}^\dagger \boldsymbol{b} \right\|_{\mathbf{F}}.$$

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

1. Note that the ordering of $\mathbf{Z}$ and $\mathbf{M}$ is crucial: this definition is not equivalent to $\|\mathbf{I}_{\mathrm{im}\,\mathbf{M}} - \mathbf{M}\mathbf{Z}\|_{\mathbf{F} \to \mathbf{F}}$ being small.

We now argue that the properties of the approximate LU factorization produced by our algorithm imply that a solver for systems in it is an approximate pseudoinverse of the original Laplacian.

**Lemma 5.0.7.** *Suppose we are given matrices* $\mathbf{L}$, $\widetilde{\mathbf{L}}$ *and a positive semi-definite matrix* $\mathbf{F}$ *such that* $\ker(\mathbf{F}) \subseteq \ker(\mathbf{L}) = \ker(\mathbf{L}^\intercal) = \ker(\widetilde{\mathbf{L}}) = \ker(\widetilde{\mathbf{L}}^\intercal)$ *and*

1. $\left\|\mathbf{F}^{\dagger 1/2}(\mathbf{L} - \widetilde{\mathbf{L}})\mathbf{F}^{\dagger 1/2}\right\|_2 \leq \epsilon$,
2. $\widetilde{\mathbf{L}}^T\mathbf{F}^\dagger\widetilde{\mathbf{L}} \succeq \gamma\mathbf{F}$.

*Then* $\widetilde{\mathbf{L}}^\dagger$ *is an* $\sqrt{\epsilon^2\gamma^{-1}}$*-approximate pseudoinverse for* $\mathbf{L}$ *w.r.t. the norm* $\mathbf{F}$.

*Proof.* Note $\mathbf{I}_{\mathrm{im}(\mathbf{L})} = \mathbf{\Pi}$, but we will abuse notation and simply write $\mathbf{I}$ for matrix. The condition we need to show $\left\|\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right\|_{\mathbf{F}\to\mathbf{F}} \leq \sqrt{\epsilon^2\gamma^{-1}}$ is equivalent to

$$\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right)^\top \mathbf{F}\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right) \preceq \epsilon^2\gamma^{-1}\mathbf{F}.$$

By rearranging a factor of $\widetilde{\mathbf{L}}$ on the LHS, we get

$$\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right)^\top \mathbf{F}\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right) = \left(\widetilde{\mathbf{L}} - \mathbf{L}\right)^\top \widetilde{\mathbf{L}}^{\dagger\top}\mathbf{F}\widetilde{\mathbf{L}}^\dagger\left(\widetilde{\mathbf{L}} - \mathbf{L}\right) \tag{5.3}$$

$$\preceq \gamma^{-1}\left(\mathbf{L}^{(n)} - \mathbf{L}\right)^\top \mathbf{F}^\dagger\left(\mathbf{L}^{(n)} - \mathbf{L}\right), \tag{5.4}$$

where in the last inequality we used $\widetilde{\mathbf{L}}^{\dagger\top}\mathbf{F}\widetilde{\mathbf{L}}^\dagger \preceq \gamma^{-1}\mathbf{F}^\dagger$. Condition 1, i.e., $\left\|\mathbf{F}^{\dagger 1/2}\left(\mathbf{L}^{(n)} - \mathbf{L}\right)\mathbf{F}^{\dagger 1/2}\right\|_2 \leq \epsilon$ is equivalent to

$$\left(\mathbf{L}^{(n)} - \mathbf{L}\right)^\top \mathbf{F}^\dagger\left(\mathbf{L}^{(n)} - \mathbf{L}\right) \preceq \epsilon^2\mathbf{F}. \tag{5.5}$$

Combining inequalities 5.4 and 5.5, we get $\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right)^\top \mathbf{F}\left(\mathbf{I} - \widetilde{\mathbf{L}}^\dagger\mathbf{L}\right) \preceq \epsilon^2\gamma^{-1}\mathbf{F}$. □

We would like for the error guarantees of our solver to be in terms of $\mathbf{U}_\mathbf{L}$. In order to provide such guarantees, we need to relate this matrix to $\mathbf{F}$.

**Lemma 5.0.8.**

$$\mathbf{U}_\mathbf{L}/O(\log(n)) \preceq \mathbf{F} \preceq O(n^2\log^5 n)\cdot\mathbf{U}_\mathbf{L}$$

*Proof.* Since $\mathbf{F} = \sum_{p'\leq p}\theta_{p'}\mathbf{U}_{\mathbf{S}^{(i_p)}}$ and $\theta_{p'} = \frac{1}{O(\log n)}$, we have

$$\mathbf{F} \succeq \frac{1}{O(\log n)}\mathbf{U}_\mathbf{L}.$$

We have by Lemma 5.0.4 that w.h.p.,

$$\begin{aligned}\mathbf{F} &\preceq O(\log^2 n)\cdot\widetilde{\mathbf{L}}^T\mathbf{F}^\dagger\widetilde{\mathbf{L}}\\ &\preceq O(\log^4 n)\cdot\mathbf{L}^T\mathbf{F}^\dagger\mathbf{L}\\ &\preceq O(\log^5 n)\cdot\mathbf{L}^T\mathbf{U}_\mathbf{L}^\dagger\mathbf{L}\\ &\preceq O(n^2\log^5 n)\cdot\mathbf{U}_\mathbf{L},\end{aligned}$$

where we used Lemma A.4.1 and Lemma 5.0.3 for the second step and Lemma 13 from [CKP+16a] pg. 19 for the last step. □

We have now stated the key theorems and lemmas needed to analyze correctness. With these tools, we can obtain the main theorem statement about finding sparse LU factorizations (Theorem 1.2.3) as follows.

*Proof of Theorem 1.2.3.* It is clear from the statement of the algorithm and the guarantees of Theorem 5.0.2 that EulerianLU(Algorithm 8) outputs an LU factorization with the sparsity claimed and with the claimed bound on running time and error probability. The remaining correctness guarantees were proven as Lemma 5.0.8, Lemma 5.0.4, and Lemma 5.0.3, respectively. □

We now have all the tools we need to obtain a fast solver for strongly connected Eulerian Laplacian systems.

*Proof of Corollary 1.2.4.* Suppose we have an Eulerian Laplacian $\mathbf{L}$ and find a $1/O(\log^2(n))$-approximate LU factorization in nearly linear time using Theorem 1.2.3 in the sense that $\|\mathbf{F}^{\dagger/2}(\mathbf{L} - \boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{U}})\mathbf{F}^{\dagger/2}\|_2 \leq 1/O(\log^2 n)$. Because it is an LU factorization, we can solve systems in it in linear time. By Lemma 5.0.7, such a solver is an 0.1-approximate pseudoinverse of $\mathbf{L}$ with respect to $\mathbf{F}$, provided we pick an appropriately small constant in the error guarantee we invoke our LU factorization algorithm EulerianLU(Algorithm 8) with. By Lemma 5.0.6, if we precondition the original system with this solver, we can find a solution $x$ to the original system with $\epsilon/\text{poly}(n)$ error in the sense that $\left\|x - \mathbf{L}^{\dagger}b\right\|_{\mathbf{F}} \leq \frac{\epsilon}{\text{poly}(n)} \cdot \left\|\mathbf{L}^{\dagger}b\right\|_{\mathbf{F}}$ in nearly linear time. Since $\mathbf{F} \approx_{\text{poly}(n)} \mathbf{U}_{\mathbf{L}}$, this implies $\left\|x - \mathbf{L}^{\dagger}b\right\|_{\mathbf{U}_{\mathbf{L}}} \leq \epsilon \cdot \left\|\mathbf{L}^{\dagger}b\right\|_{\mathbf{U}_{\mathbf{L}}}$. □

## 5.1 Analysis of the LU Factorization Algorithm

In this section, we prove the guarantees of our LU Factorization Algorithm.

### 5.1.1 Finding an $\alpha$-RCDD Block

First we provide a basic result (an analog of [LPS15]) showing we can find large $\alpha$-RCDD blocks of vertices efficiently.

**Algorithm 9:** FindRCDDBlock$(G, \alpha)$

**Input:** a directed graph $G$ and a parameter $\alpha$
**Output:** an $\alpha$-RCDD set of vertices $F$ of size at least $n/16$

$F \leftarrow \emptyset$
**while** $|F| < \frac{n}{16(1+\alpha)}$ **do**
  $F \leftarrow \emptyset$.
  Randomly sample $k = \frac{n}{8(1+\alpha)}$ vertices.
  Discard any vertices that are not $\alpha$-RCDD.
  Set $F$ equal to the resulting set.

**Theorem 5.1.1.** *Given a directed graph $G$, the function* FindRCDDBlock$(G, \alpha)$ *(Algorithm 9) outputs an $\alpha$-RCDD set of vertices of size at least $\frac{n}{16(1+\alpha)}$ in time $O(m \log(1/\delta))$ with probability at least $1 - O(\delta)$.*

*Proof.* This is immediate from Lemma A.3.2 is in the Appendix. □

### 5.1.2 Single Vertex Elimination Algorithm

The algorithm below produces a sparse approximation of the clique created by Gaussian Elimination on an Eulerian directed Laplacian. It can be implemented to run in $O(\deg(v)\log\deg(v))$ time where deg is the combinatorial degree of the vertex $v$ being eliminated. The algorithm has two key features. When including self-loops, it preserves the weighted in and out degree of each vertex. This ensures the graph created by replacing the clique with the sparse approximation is still Eulerian. Note that we may get self-loops which will cancel out and change the degree of vertices, but it won't change the fact that each vertex still has in-degree equal to out-degree.

**Algorithm 10:** SINGLEVERTEXELIM($\boldsymbol{l}, \boldsymbol{r}$)

**Input:** $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{R}^n$ s.t. both have non-negative entries and $\mathbf{1}^\top \boldsymbol{l} = \mathbf{1}^\top \boldsymbol{r}$.
**Output:** $\mathbf{N} \in \mathbb{R}^{n \times n}$

$s \leftarrow \mathbf{1}^\top \boldsymbol{l}$
**if** $s = 0$ **then**
  **return** $\mathbf{0}$
**else if** $\min(\boldsymbol{l}) \leq \min(\boldsymbol{r})$ **then**
  $i \leftarrow \arg\min(\boldsymbol{l})$
  Pick index $j \leftarrow k$ with probability $\boldsymbol{r}(k)/s$
  **return** $\boldsymbol{l}(i)\boldsymbol{\chi}_i\boldsymbol{\chi}_j^\top +$ SINGLEVERTEXELIM$(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i, \boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j)$
**else**
  $i \leftarrow \arg\min(\boldsymbol{r})$
  Pick index $j \leftarrow k$ with probability $\boldsymbol{l}(k)/s$
  **return** $\boldsymbol{r}(i)\boldsymbol{\chi}_j\boldsymbol{\chi}_i^\top +$ SINGLEVERTEXELIM$(\boldsymbol{l} - \boldsymbol{r}(i)\boldsymbol{\chi}_j, \boldsymbol{r} - \boldsymbol{r}(i)\boldsymbol{\chi}_i)$

**Lemma 5.1.2.** *The matrix* $\mathbf{A}$ *returned by* SINGLEVERTEXELIM$(\boldsymbol{l}, \boldsymbol{r})$ *has* $\mathrm{nnz}(\mathbf{A}) \leq \mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r})$. *Also, the algorithm makes* $\leq \mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r})$ *recursive calls to itself, and all the recursive calls are made to vectors with non-negative entries satisfying* $\mathbf{1}^\top \boldsymbol{l} = \mathbf{1}^\top \boldsymbol{r}$.

*Proof.* We prove this by induction on $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r})$. Base case: $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = 0$, then $\boldsymbol{l}, \boldsymbol{r} = \vec{\mathbf{0}}$, so $\mathbf{A} = \mathbf{0}$, and $\mathrm{nnz}(\mathbf{A}) = 0$. This proves the base case. For the inductive step, we suppose $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = k+1$ and that the lemma holds whenever $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) \leq k$. W.l.o.g. consider the case of $\min(\boldsymbol{l}) \leq \min(\boldsymbol{r})$. Note that $\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i, \boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j \geq \vec{\mathbf{0}}$, and that

$$\mathrm{nnz}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i) + \mathrm{nnz}(\boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j) \leq k$$

so by the induction hypothesis with $\mathbf{A}' =$ SINGLEVERTEXELIM$(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i, \boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j)$ we have $\mathrm{nnz}(\mathbf{A}') \leq k$, and so $\mathrm{nnz}(\mathbf{A}) \leq k+1$. This proves the lemma by induction. The number of recursive calls can be bounded in the same way. □

**Lemma 5.1.3.** *The matrix* $\mathbf{A}$ *returned by* SINGLEVERTEXELIM$(\boldsymbol{l}, \boldsymbol{r})$ *has only non-negative entries and satisfies* $\mathbf{A}\mathbf{1} = \boldsymbol{l}$, *and* $\mathbf{1}^\top \mathbf{A} = \boldsymbol{r}^\top$.

*Proof.* We prove the lemma by induction on $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r})$. It is true in the base case $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = 0$, where $\boldsymbol{l}, \boldsymbol{r} = \vec{\mathbf{0}}$, so $\mathbf{A} = \mathbf{0}$, and $\mathbf{A}\mathbf{1} = \vec{\mathbf{0}} = \boldsymbol{l}$, and $\mathbf{1}^\top \mathbf{A} = \vec{\mathbf{0}}^\top = \boldsymbol{r}^\top$.

For the inductive step, we suppose $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = k+1$ and that the lemma holds whenever $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) \leq k$. W.l.o.g. consider the case of $\min(\boldsymbol{l}) \leq \min(\boldsymbol{r})$.

Let $\mathbf{A}' = \textsc{SingleVertexElim}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i, \boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j)$. By the induction hypothesis,

$$\mathbf{A}\mathbf{1} = \boldsymbol{l}(i)\boldsymbol{\chi}_i\boldsymbol{\chi}_j^\top\mathbf{1} + \mathbf{A}'\mathbf{1} = \boldsymbol{l}(i)\boldsymbol{\chi}_i + \boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i = \boldsymbol{l}.$$

and similarly

$$\mathbf{1}^\top\mathbf{A} = \mathbf{1}^\top\boldsymbol{l}(i)\boldsymbol{\chi}_i\boldsymbol{\chi}_j^\top + \mathbf{1}^\top\mathbf{A}' = \boldsymbol{l}(i)\boldsymbol{\chi}_j^\top + \boldsymbol{r}^\top - \boldsymbol{l}(i)\boldsymbol{\chi}_j^\top = \boldsymbol{r}.$$

□

**Lemma 5.1.4.** *Given $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{R}^n$ s.t. both have non-negative entries and $\mathbf{1}^\top\boldsymbol{l} = \mathbf{1}^\top\boldsymbol{r} = s$, let $\mathbf{A} = \textsc{SingleVertexElim}(\boldsymbol{l}, \boldsymbol{r})$. Then $\mathbb{E}[\mathbf{A}] = \boldsymbol{l}\boldsymbol{r}^\top/s$.*

*Proof.* We prove the lemma by induction on $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r})$. It is true in the base case $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = 0$, where $\boldsymbol{l}, \boldsymbol{r} = \vec{\mathbf{0}}$, so $\mathbf{A} = \mathbf{0}$, so $\mathbb{E}[\mathbf{A}] = \mathbf{0}$. For the inductive step, we suppose $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) = k+1$ and that the lemma holds whenever $\mathrm{nnz}(\boldsymbol{l}) + \mathrm{nnz}(\boldsymbol{r}) \leq k$. W.l.o.g. consider the case of $\min(\boldsymbol{l}) \leq \min(\boldsymbol{r})$.

Then

$$\begin{aligned}
\mathbb{E}[\mathbf{A}] &= \sum_j \frac{\boldsymbol{r}(j)}{s}\left(\boldsymbol{l}(i)\boldsymbol{\chi}_i\boldsymbol{\chi}_j^\top + \frac{1}{s - \boldsymbol{l}(i)}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i)(\boldsymbol{r} - \boldsymbol{l}(i)\boldsymbol{\chi}_j)^\top\right) \\
&= \sum_j \frac{\boldsymbol{l}(i)}{s}\boldsymbol{\chi}_i\boldsymbol{r}(j)\boldsymbol{\chi}_j^\top + \frac{\boldsymbol{r}(j)}{s}\frac{1}{s - \boldsymbol{l}(i)}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i)\boldsymbol{r}^\top - \frac{\boldsymbol{l}(i)}{s}\frac{1}{s - \boldsymbol{l}(i)}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i)\boldsymbol{r}(j)\boldsymbol{\chi}_j^\top \\
&= \frac{\boldsymbol{l}(i)}{s}\boldsymbol{\chi}_i\boldsymbol{r}^\top + \frac{1}{s - \boldsymbol{l}(i)}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i)\boldsymbol{r}^\top - \frac{\boldsymbol{l}(i)}{s}\frac{1}{s - \boldsymbol{l}(i)}(\boldsymbol{l} - \boldsymbol{l}(i)\boldsymbol{\chi}_i)\boldsymbol{r}^\top \\
&= \boldsymbol{l}(i)\boldsymbol{\chi}_i\boldsymbol{r}^\top\left(\frac{1}{s} - \frac{1}{s - \boldsymbol{l}(i)} + \frac{\boldsymbol{l}(i)}{s}\frac{1}{s - \boldsymbol{l}(i)}\right) + \boldsymbol{l}\boldsymbol{r}^\top\frac{1}{s - \boldsymbol{l}(i)}\left(1 - \frac{\boldsymbol{l}(i)}{s}\right) \\
&= \boldsymbol{l}\boldsymbol{r}^\top/s.
\end{aligned}$$

□

A crucial matrix used in analyzing the elimination of a single vertex is the Schur complement of of the star incident on the eliminated vertex in the undirectification of the whole matrix.

**Definition 5.1.5.** *Given an Eulerian Laplacian $\mathbf{L}$ and a vertex $v$, let $\mathbf{U}_{local} = \textsc{Cl}[\mathbf{U}_\mathbf{L}]_v$ (see Equation (2.4)).*

Note that

$$\mathbf{U}_{local} \preceq \textsc{St}[\mathbf{U}_\mathbf{L}]_v.$$

And hence for a random choice of vertex $v$ in a graph with $n$ remaining vertices

$$\mathbb{E}_v[\mathbf{U}_{local}] \preceq \frac{2}{n}\mathbf{U}_\mathbf{L} \tag{5.6}$$

**Lemma 5.1.6** (Single Vertex Elimination Routine)**.** *There is a routine* SingleVertexElim *that takes the in and out adjacency list vectors $\boldsymbol{l}$ and $\boldsymbol{r}$ of a vertex $u$ in an Eulerian Laplacian with $d$ non-zeros, and produces a matrix $\mathbf{A}$ with at most $d$ non-zeros such that the error matrix*

$$\mathbf{X} = \frac{1}{\boldsymbol{r}^\top\mathbf{1}}\boldsymbol{l}\boldsymbol{r}^\top - \mathbf{A}$$

*satisfies*

1. $\mathbf{X}\mathbf{1} = 0$, $\mathbf{X}^\top \mathbf{1} = 0$, *and*

2. $\mathbb{E}[\mathbf{X}] = 0$, *and*

3. *For the local undirectification,* $\mathbf{U}_{local}$ *as given in Definition 5.1.5,*

$$\left\| \mathbf{U}_{local}^{\dagger 1/2} \mathbf{X} \mathbf{U}_{local}^{\dagger 1/2} \right\|_2 \leq 4.$$

*Proof.* We already have Parts 1 and 1 from Lemma 5.1.3 and Lemma 5.1.4.

For part 3, we note that by $\mathbf{A}$ (by Lemma 5.1.3) has the sum of the absolute value of its entries in the $i$th row and $i$th column of at most $\boldsymbol{l}(i) + \boldsymbol{r}(i)$. This similarly applies to the the expectation $\frac{1}{\boldsymbol{r}^\top \mathbf{1}} \boldsymbol{l}\boldsymbol{r}^\top$. Thus, the sums for the error matrix $\mathbf{X}$ are at most double this: $2(\boldsymbol{l}(i) + \boldsymbol{r}(i))$.

Now, we define a diagonal matrix $\mathbf{D}_{local}$, whose $i$th diagonal entry is $\frac{\boldsymbol{l}(i)+\boldsymbol{r}(i)}{2}$.

safe Because the sums of the absolute values of the $i$th row and column are at most $4(\mathbf{D}_{local})_{ii}$, we have

$$\|\mathbf{D}_{local}^{-1/2} \mathbf{X} \mathbf{D}_{local}^{-1/2}\| \leq 4.$$

Now, $\mathbf{U}_{local}^{\dagger} \preceq \mathbf{D}_{local}^{-1}$, so $\|\mathbf{U}_{local}^{\dagger 1/2} \mathbf{D}_{local}^{1/2}\|_2 \leq 1$ and hence

$$\|\mathbf{U}_{local}^{-1/2} \mathbf{X} \mathbf{U}_{local}^{-1/2}\| = \|(\mathbf{U}_{local}^{\dagger 1/2} \mathbf{D}_{local}^{1/2})(\mathbf{D}_{local}^{-1/2} \mathbf{X} \mathbf{D}_{local}^{-1/2})(\mathbf{U}_{local}^{\dagger 1/2} \mathbf{D}_{local}^{1/2})^\top\| \leq 4.$$

□

### 5.1.3 Bounds on Schur Complements

In this section we bound the blowup of the Schur complements as we pivot away $\alpha$-RCDD subsets of vertices. This is useful for bounding the variance operators after repeated single step eliminations as in Section 5.1. The main result that we'll show is:

**Lemma 5.1.7.** *Suppose that* $\mathbf{L} = \mathbf{D} - \mathbf{A}^\top \in \mathbb{R}^{n \times n}$ *is an Eulerian Laplacian, let* $\mathbf{U} \stackrel{\text{def}}{=} \mathbf{U}_{\mathbf{L}}$ *and let* $F, C \subseteq [n]$ *be a partition of* $[n]$ *such that* $\mathbf{U}_{FF} \succeq \frac{1}{\alpha} \mathbf{D}_{FF}$ *then* $\mathbf{U}_{\mathrm{Sc}[\mathbf{L}]_F} \preceq (1 + 2\alpha)\mathbf{U}$.

**Corollary 5.1.8.** *Suppose that* $\mathbf{L} = \mathbf{D} - \mathbf{A}^\top \in \mathbb{R}^{n \times n}$ *is an Eulerian Laplacian, and* $F \subseteq [n]$ *is an* $\alpha$*-RCDD subset. Then* $\mathbf{U}_{\mathrm{Sc}[\mathbf{L}]_F} \preceq (3 + \frac{1}{\alpha})\mathbf{U}_{\mathbf{L}}$.

*Proof.* Consider an Eulerian Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}^\top \in \mathbb{R}^{n \times n}$, and an $\alpha$-RCDD subset $F \subseteq [n]$ of $\mathbf{L}$. Let $\mathbf{U} \stackrel{\text{def}}{=} \mathbf{U}_{\mathbf{L}} = \mathbf{D} - \mathbf{U}_{\mathbf{A}}$, then the $\alpha$-RCDD condition implies that $\frac{1}{1+\alpha}\mathbf{D}_{FF} - (\mathbf{U}_{\mathbf{A}})_{FF} \succeq \mathbf{0}$, since it is SDD. Hence $\mathbf{U}_{FF} = \frac{\alpha}{1+\alpha}\mathbf{D}_{FF} + \frac{1}{1+\alpha}\mathbf{D}_{FF} - (\mathbf{U}_{\mathbf{A}})_{FF} \succeq \frac{\alpha}{1+\alpha}\mathbf{D}$. The conclusion now follows from Lemma 5.1.7. □

#### Bounding General Schur Complements

We start with a simple lemma that upper bounds the Schur complement of a general matrix spectrally via its symmetrization.

**Lemma 5.1.9.** *If* $\mathbf{N} \in \mathbb{R}^{n \times n}$ *satisfies* $\mathbf{U} \stackrel{\text{def}}{=} \mathbf{U}_{\mathbf{N}} \succeq \mathbf{0}$ *and* $F, C \subseteq [n]$ *is a partition of* $[n]$ *where* $\mathbf{U}_{FF} \succ \mathbf{0}$ *and* $\mathbf{N}_{FF}$ *is invertible and*

$$\mathbf{M} \stackrel{\text{def}}{=} \begin{bmatrix} [\mathbf{N}_{FF}^{-1}]^\top \mathbf{U}_{FF} \mathbf{N}_{FF}^{-1} & \mathbf{0}_{FC} \\ \mathbf{0}_{CF} & \mathbf{0}_{CC} \end{bmatrix} \preceq \alpha [\mathbf{N}^\dagger]^\top \mathbf{U} \mathbf{N}^\dagger$$

*then* $\mathbf{U}_{\mathrm{Sc}[\mathbf{N}]_F} \preceq (1 + \alpha)\mathbf{U}$.

*Proof.* Let $\boldsymbol{z} \in \mathbb{R}^n$ be arbitrary and let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ be defined so that $\boldsymbol{x}_C = \boldsymbol{y}_C = \boldsymbol{z}_C$, $\boldsymbol{x}_F \stackrel{\text{def}}{=} -\mathbf{N}_{FF}^{-1}\mathbf{N}_{FC}\boldsymbol{z}_C$, and $\boldsymbol{y}_F = -\mathbf{U}_{FF}^{-1}\mathbf{U}_{FC}\boldsymbol{z}_C$. Note that,

$$\mathbf{L}\boldsymbol{x} = \begin{bmatrix} \mathbf{N}_{FF} & \mathbf{N}_{FC} \\ \mathbf{N}_{CF} & \mathbf{N}_{CC} \end{bmatrix} \begin{pmatrix} -\mathbf{N}_{FF}^{-1}\mathbf{N}_{FC}\boldsymbol{z}_C \\ \boldsymbol{z}_C \end{pmatrix} = \begin{pmatrix} \vec{0}_F \\ \textsc{Sc}[\mathbf{N}]_F \, \boldsymbol{z}_C \end{pmatrix}.$$

and therefore $\boldsymbol{x}^\top \mathbf{L}\boldsymbol{x} = \boldsymbol{z}^\top \textsc{Sc}[\mathbf{N}]_F \, \boldsymbol{z}$. Furthermore, note that

$$\boldsymbol{z}^\top \mathbf{U}\boldsymbol{z} = \boldsymbol{z}_F^\top \mathbf{U}_{FF}\boldsymbol{z}_F + 2\boldsymbol{z}_C\mathbf{U}_{CF}\boldsymbol{z}_F + \boldsymbol{z}_C^\top \mathbf{U}_{CC}\boldsymbol{z}_C$$

and since $\mathbf{U}_{FF} \succ 0$ we have that $\boldsymbol{z}^\top \mathbf{U}\boldsymbol{z}$ is minimized over $\boldsymbol{z}_F$ when $\boldsymbol{z}_F = -\mathbf{U}_{FF}^{-1}\mathbf{U}_{FC}\boldsymbol{z}_C$ and thus $\boldsymbol{y}^\top \mathbf{U}\boldsymbol{y} \leq \boldsymbol{z}^\top \mathbf{U}\boldsymbol{z}$. Furthermore, note that

$$\boldsymbol{y}_F - \boldsymbol{x}_F = \mathbf{N}_{FF}^{-1}[\mathbf{N}\boldsymbol{y}]_F$$

and consequently as $\boldsymbol{x}_C = \boldsymbol{y}_C$ we have that

$$\begin{aligned} \|\boldsymbol{x} - \boldsymbol{y}\|_{\mathbf{U}}^2 = (\boldsymbol{x}_F - \boldsymbol{y}_F)\mathbf{U}_{FF}(\boldsymbol{x}_F - \boldsymbol{y}_F) &= \|[\mathbf{N}\boldsymbol{y}]_F\|^2_{\mathbf{N}_{FF}^{-1}\mathbf{U}_{FF}\mathbf{N}_{FF}^{-1}} \, . \\ &= \|\mathbf{N}\boldsymbol{y}\|_{\mathbf{M}}^2 \leq \alpha \, \|\mathbf{N}\boldsymbol{y}\|^2_{[\mathbf{N}^\dagger]^\top \mathbf{U}\mathbf{N}^\dagger} \leq \alpha \, \|\boldsymbol{y}\|_{\mathbf{U}}^2 \, . \end{aligned}$$

Further, using the $\mathbf{U}$-orthogonality between $\boldsymbol{y}$ and $\boldsymbol{x} - \boldsymbol{y}$ (as $\boldsymbol{x} - \boldsymbol{y}$ is supported on $F$ and $\mathbf{U}\boldsymbol{y}$ is 0 on $F$),

$$\|\boldsymbol{x}\|_{\mathbf{U}}^2 = \|\boldsymbol{y}\|_{\mathbf{U}}^2 + \|\boldsymbol{x} - \boldsymbol{y}\|_{\mathbf{U}} \leq (1 + \alpha) \, \|\boldsymbol{y}\|_{\mathbf{U}}^2$$

and as $\|\boldsymbol{x}\|_{\mathbf{U}}^2 = \boldsymbol{z}^\top \textsc{Sc}[\mathbf{N}]_F \, \boldsymbol{z} = \boldsymbol{z}^\top \mathbf{U}_{\textsc{Sc}[\mathbf{N}]_F}\boldsymbol{z}$ and $\|\boldsymbol{y}\|_{\mathbf{U}}^2 \leq \|\boldsymbol{z}\|_{\mathbf{U}}^2$ the result follows. □

**Schur Complements of Eulerian Laplacians**

Here we show how to apply the Schur complement bounds to Schur complements of Eulerian Laplacians.

We begin with a fairly self-contained lemma about Eulerian Laplacians.

**Lemma 5.1.10.** *Suppose that $\mathbf{L} = \mathbf{D} - \mathbf{A}^\top \in \mathbb{R}^{n \times n}$ is an Eulerian Laplacian associated with directed graph $G = (V, E, w)$ and let $\mathbf{U} = \frac{1}{2}(\mathbf{L} + \mathbf{L}^\top)$. Then $\mathbf{L}^\top \mathbf{D}^{-1}\mathbf{L} \preceq 2 \cdot \mathbf{U}$.*

*Proof.* Let $\boldsymbol{x} \in \mathbb{R}^n$ be arbitrary and recall that

$$[\mathbf{L}\boldsymbol{x}]_i = \sum_{(i,j) \in E} w_{ij}(\boldsymbol{x}(i) - \boldsymbol{x}(j)) \quad \text{and} \quad \mathbf{D}_{ii} = \sum_{(i,j) \in E} w_{ij} \, .$$

Furthermore, by Cauchy Schwarz we have that

$$[\mathbf{L}\boldsymbol{x}]_i^2 = \left[ \sum_{(i,j) \in E} w_{ij}(\boldsymbol{x}(i) - \boldsymbol{x}(j)) \right]^2 \leq \left[ \sum_{(i,j) \in E} w_{ij} \right] \cdot \left[ \sum_{(i,j) \in E} w_{ij}(\boldsymbol{x}(i) - \boldsymbol{x}(j))^2 \right] .$$

Consequently,

$$\boldsymbol{x}^\top \mathbf{L}^\top \mathbf{D}^{-1}\mathbf{L}\boldsymbol{x} = \sum_{i \in [n]} \frac{[\mathbf{L}\boldsymbol{x}]_i^2}{\mathbf{D}_{ii}} \leq \sum_{i \in [n]} \sum_{(i,j) \in E} w_{ij}(\boldsymbol{x}(i) - \boldsymbol{x}(j))^2 = 2 \cdot \boldsymbol{x}^\top \mathbf{U}\boldsymbol{x} \, .$$

□

Using this we prove the main bound.

*Proof of Lemma 5.1.7.* First note that $\mathbf{L}_{FF}$ and $\mathbf{U}_{FF}$ must be RCDD as $\mathbf{L}$ is Eulerian, and since $\mathbf{U}_{FF} \succeq \frac{1}{\alpha}\mathbf{D}_{FF}$ it is the case that $\mathbf{U}_{FF}$ is invertible and so is $\mathbf{L}_{FF}$. Consequently, as $\mathbf{U}_{FF} \preceq \mathbf{L}_{FF}^\top \mathbf{U}_{FF}^{-1}\mathbf{L}_{FF}$ from our general bounds on harmonic symmetrizations we have that $[\mathbf{L}_{FF}^{-1}]^\top \mathbf{U}_{FF}[\mathbf{L}_{FF}^{-1}] \preceq \mathbf{U}_{FF}^{-1}$. Furthermore, as $\mathbf{U}_{FF} \succeq \frac{1}{\alpha}\mathbf{D}_{FF}$ we have that $\mathbf{U}_{FF}^{-1} \preceq \alpha\mathbf{D}_{FF}^{-1}$ and therefore that

$$\begin{bmatrix} [\mathbf{L}_{FF}^{-1}]^\top \mathbf{U}_{FF}\mathbf{L}_{FF}^{-1} & \mathbf{0}_{FC} \\ \mathbf{0}_{CF} & \mathbf{0}_{CC} \end{bmatrix} \preceq \begin{bmatrix} \alpha\mathbf{D}_{FF}^{-1} & \mathbf{0}_{FC} \\ \mathbf{0}_{CF} & \mathbf{0}_{CC} \end{bmatrix} \preceq \alpha\mathbf{D}^{-1} \preceq \alpha[\mathbf{L}]^\top[\mathbf{L}^\dagger]^\top\mathbf{D}^{-1}[\mathbf{L}^\dagger][\mathbf{L}]\,.$$

As $[\mathbf{L}^\dagger]^\top\mathbf{D}^{-1}[\mathbf{L}^\dagger] \preceq 2\cdot\mathbf{U}$ by Lemma 5.1.10 the result then follows from Lemma 5.1.9. □

### Schur Complements after Error

Here, we show that symmetrized Schur complements are robust to small changes to the original matrix. This will be useful for analyzing the single vertex pivoting algorithm, which accumulates error as it pivots.

**Lemma 5.1.11.** *If $\mathbf{N} \in \mathbb{R}^{n\times n}$ satisfies $\mathbf{U} \stackrel{\text{def}}{=} \mathbf{U}_{\mathbf{N}} \succeq \mathbf{0}$ and $F, C \subseteq [n]$ is a partition of $[n]$ where $\mathbf{U}_{FF} \succ \mathbf{0}$ and $\mathbf{N}_{FF}$ is invertible, and $\mathbf{N}$ and $\widetilde{\mathbf{N}}$ have the same kernel and cokernel as $\mathbf{U}$ $\left\|\mathbf{U}^{\dagger 1/2}(\widetilde{\mathbf{N}} - \mathbf{N})\mathbf{U}^{\dagger 1/2}\right\|_2 \leq \epsilon$, then $\mathbf{U}_{\mathrm{Sc}[\widetilde{\mathbf{N}}]_F} \preceq \left(\frac{1+\epsilon}{1-\epsilon}\right)^2 \mathbf{U}_{\mathrm{Sc}[\mathbf{N}]_F}$.*

*Proof.* As in the proof of Lemma 5.1.9, given any vector $\boldsymbol{z}$ we define $\boldsymbol{x}$ and $\widetilde{\boldsymbol{x}}$ such that $\boldsymbol{x}_C = \widetilde{\boldsymbol{x}}_C = \boldsymbol{z}_C$, $\boldsymbol{x}_F \stackrel{\text{def}}{=} -\mathbf{N}_{FF}^{-1}\mathbf{N}_{FC}\boldsymbol{z}_C$, $\widetilde{\boldsymbol{x}}_F \stackrel{\text{def}}{=} -\widetilde{\mathbf{N}}_{FF}^{-1}\widetilde{\mathbf{N}}_{FC}\boldsymbol{z}_C$. We then have, again, $\boldsymbol{z}^T\mathbf{U}_{\mathrm{Sc}[\mathbf{N}]_F}\boldsymbol{z} = \boldsymbol{x}^T\mathbf{U}\boldsymbol{x}$ and $\boldsymbol{z}^T\mathbf{U}_{\mathrm{Sc}[\widetilde{\mathbf{N}}]_F}\boldsymbol{z} = \widetilde{\boldsymbol{x}}^T\widetilde{\mathbf{U}}\widetilde{\boldsymbol{x}}$.

Now,

$$\begin{aligned}\widetilde{x}_F - x_F &= -\widetilde{\mathbf{N}}_{FF}^{-1}[\widetilde{\mathbf{N}}x]_F \\ &= -\widetilde{\mathbf{N}}_{FF}^{-1}[(\widetilde{\mathbf{N}} - \mathbf{N})x]_F.\end{aligned}$$

By the guarantee of $\widetilde{\mathbf{N}}$, we have

$$\begin{aligned}\left\|(\widetilde{\mathbf{N}} - \mathbf{N})\boldsymbol{x}\right\|_{\widetilde{\mathbf{U}}^\dagger} &\leq \frac{1}{1-\epsilon}\left\|(\widetilde{\mathbf{N}} - \mathbf{N})\boldsymbol{x}\right\|_{\mathbf{U}^\dagger} \\ &\leq \frac{\epsilon}{1-\epsilon}\left\|\boldsymbol{x}\right\|_{\mathbf{U}}.\end{aligned}$$

This gives

$$\begin{aligned}\left\|\widetilde{\boldsymbol{x}}_F - \boldsymbol{x}_F\right\|_{\widetilde{\mathbf{U}}} &= \left\|\widetilde{\mathbf{N}}_{FF}^{-1}[(\widetilde{\mathbf{N}} - \mathbf{N})\boldsymbol{x}]_F\right\|_{\widetilde{\mathbf{U}}} \\ &\leq \left\|(\widetilde{\mathbf{N}} - \mathbf{N})\boldsymbol{x}\right\|_{\widetilde{\mathbf{U}}^\dagger} \\ &\leq \frac{\epsilon}{1-\epsilon}\left\|\boldsymbol{x}\right\|_{\mathbf{U}}\end{aligned}$$

Then we can write

$$\begin{aligned}
\|\boldsymbol{z}\|^2_{\mathbf{U}_{\mathrm{Sc}[\widetilde{\mathbf{N}}]_F}} &= \|\widetilde{\boldsymbol{x}}\|^2_{\widetilde{\mathbf{U}}} \\
&\leq \left(\|\boldsymbol{x}\|_{\widetilde{\mathbf{U}}} + \|\widetilde{\boldsymbol{x}} - \boldsymbol{x}\|_{\widetilde{\mathbf{U}}}\right)^2 \\
&\leq \left((1+\epsilon)\|\boldsymbol{x}\|_{\mathbf{U}} + \frac{\epsilon}{1-\epsilon}\|\boldsymbol{x}\|_{\mathbf{U}}\right)^2 \\
&\leq \left(\frac{1+\epsilon}{1-\epsilon}\right)^2 \|\boldsymbol{x}\|^2_{\mathbf{U}} \\
&= \left(\frac{1+\epsilon}{1-\epsilon}\right)^2 \|\boldsymbol{z}\|^2_{\mathbf{U}_{\mathrm{Sc}[\widetilde{\mathbf{N}}]_F}}
\end{aligned}$$

□

### 5.1.4 Single Phase Analysis

We need to show that in SinglePhase (Algorithm 7), we do not do an (very expensive) full sparsification of the entire graph too many times. Second, we our runtime will have a term that is nearly linear in the sum of the degrees of the vertices we eliminate. To say that these aren't too big, we need to bound how much we can blow up the average degree of the graph and number of edges in it after doing one elimination. The following lemmas help us bound the running time of SinglePhase, and the number of calls to SparsifyEulerian made inside it.

**Lemma 5.1.12.** *Suppose* $\mathbf{L}$ *is an Eulerian Laplacian on* $n$ *vertices with* $m$ *non-zeros. In* SinglePhase$(\mathbf{L}, \delta, \epsilon)$*, for all* $k$*,* $\mathrm{nnz}(\mathbf{S}^{(k+1)}) \leq \left(1 + O\left(\frac{P}{n-k}\right)\right)\mathrm{nnz}(\mathbf{S}^{(k)})$

*Proof.* This is immediate from the fact that the algorithm only selects vertices to eliminate of at most twice the average degree and that when a vertex is eliminated, $O(\text{degree})$ edges are added to the graph from each of the $P = O(\epsilon^{-2}\log^2(1/\delta))$ calls to SingleVertexElim (Algorithm 10). □

**Lemma 5.1.13.** *Suppose* $\mathbf{L}$ *is an Eulerian Laplacian on* $n$ *vertices with* $m$ *non-zeros. In the* for*-loop of* SinglePhase *(Algorithm 7),* SparsifyEulerian$(\cdot)$ *is called* $O(P) = O(\epsilon^{-2}\log^2(1/\delta))$ *times, each time after the first on an Eulerian Laplacian with* $\widetilde{O}(n\epsilon^{-6}\log^5(1/\delta))$ *edges. The total running time for a call to* SinglePhase$(\mathbf{L}, \delta, \epsilon)$ *is* $\widetilde{O}(m + n\epsilon^{-8}\log^7(1/\delta))$.

*Proof.* The first of the two calls to produces an Eulerian Laplacian $\mathbf{S}^{(k)}$, s.t. $\mathrm{nnz}(\mathbf{S}^{(k)}) \leq T$. After $t$ eliminations, we have an Eulerian Laplacian $\mathbf{S}^{(k+t,P)}$ s.t. $\mathrm{nnz}(\mathbf{S}^{(k+t,P)}) \geq 2T$. Meanwhile, by Lemma 5.1.12, and since we eliminate at most $n/2$ vertices the number of non-zeros $\mathrm{nnz}(\mathbf{S}^{(k+t,P)})$ is upper bounded by

$$\mathrm{nnz}(\mathbf{S}^{(k)})\prod_{i=0}^{t-1}\left(1 + O\left(\frac{P}{n-k-i}\right)\right) \leq \mathrm{nnz}(\mathbf{S}^{(k)})\prod_{i=0}^{t-1}\left(1 + O\left(\frac{P}{n}\right)\right) \leq \mathrm{nnz}(\mathbf{S}^{(k)})\exp\left(O(P/n)t\right).$$

This means that $\exp\left(O(P)t\right)T \geq 2T$, so $t \geq \Omega(n/P)$. Since we perform $t$ eliminations before each next call to SparsifyEulerian$(\cdot)$, and the total number of eliminations in a phase is at most $n/2$, the number of sparsifications is upper bounded by $O(n/t) = O(P)$.

The running time for all the sparsification calls combined is $\widetilde{O}(m + P \cdot (n(\epsilon/P)^{-2}\log(1/\delta)))$ [2]. This also upper bounds the running time required for vertex eliminations, as SingleVertexElim

2. Here we assume that the edge count does not grow by more than a constant factor during a single vertex elimination. This is only false if $\epsilon < n^{-1/2}$, when our $n\epsilon^{-8} = n^4$ running time is slower than doing exact Gaussian elimination, which can then be used instead in this regime.

(Algorithm 10) can be implemented to run in time $O(d \log d)$, where $d$ is the combinatorial degree of the vertex being eliminated. With probability $1 - O(\delta)$, this also upper bounds the running time required for performing the random vertex selections of low degree vertices, which can be implemented using a simple rejection sampling approach. □

**Definition 5.1.14.** *Given an Eulerian Laplacian* $\mathbf{L}$*, a* robustly bounded Schur complement set *vertex set* $J$*, is a subset of the vertices of* $\mathbf{L}$*, s.t. for any* $\widehat{J} \subseteq J$ *and any* $\widetilde{\mathbf{L}}$ *where*

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \widetilde{\mathbf{L}} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq 1/2,$$

*we have* $\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{\widehat{J}}} \preceq O(1)\mathbf{U}_{\mathbf{L}}$.

**Lemma 5.1.15.** *Given an Eulerian Laplacian* $\mathbf{L}$*, for any fixed constant* $\alpha$*, an* $\alpha$*-RCDD subset* $J$ *of the vertices of* $\mathbf{L}$ *is robustly bounded.*

*Proof.* This is immediate from Corollary 5.1.8 and Lemma 5.1.11. □

Our goal is to prove Theorem 5.0.2. Firstly we want to show

$$\Pr\left[ \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(k_{\max})} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 > \epsilon \right] \leq O(\delta) \tag{5.7}$$

where $\mathbf{L}$ is the input to SinglePhase and $\mathbf{L}^{(k_{\max})}$ is the output. Secondly, we also need to prove the required bounds on the running time. We set up a matrix martingale to help us prove the theorem. Consider the inner loops (inside $k$, inside the $t$ loops) of SinglePhase. We can define the change in each step to be:

$$\mathbf{X}^{(k,t)} \stackrel{\text{def}}{=} \mathbf{S}^{(k,t)} - \mathbf{S}^{(k,t-1)}.$$

Since Lemma 5.1.4 implies each $\mathbf{X}^{(k,t)}$ has zero expectation, we can define a zero-mean martingale sequence

$$\mathbf{M}^{(k,t)} = \sum_{\hat{k}=1}^{k} \sum_{\hat{t}=1}^{\substack{P \text{ if } \hat{k} < k \\ t \text{ if } \hat{k} = k}} \mathbf{X}^{(\hat{k},\hat{t})} = \sum_{(\hat{k},\hat{t}) \leq (k,t)} \mathbf{X}^{(\hat{k},\hat{t})}.$$

Here we overload the $\leq$ and $<$ notation to handle pair of variables in the lexicographical sense. The final step of this martingale is

$$\mathbf{M}^{(k_{\max},P)},$$

Note that it is not the case that in general

$$\mathbf{L}^{(k)} \neq \mathbf{L} + \mathbf{M}^{(k,P)},$$

because the martingale does not include the changes introduced by calls to SparsifyEulerian.

To track the changes caused by SparsifyEulerian, we need to further define changes from the sparsification steps: We define $\mathbf{Z}^{(0)} \stackrel{\text{def}}{=} \mathbf{S}^{(0)} - \mathbf{S}$, and for $k > 0$, we let

$$\mathbf{Z}^{(k)} \stackrel{\text{def}}{=} \mathbf{S}^{(k)} - \mathbf{S}^{(k,P)}.$$

Now, we are able to express the output matrix as

$$\mathbf{L}^{(k)} = \mathbf{L} + \mathbf{M}^{(k,P)} + \sum_{\hat{k}=0}^{k} \mathbf{Z}^{(\hat{k})}.$$

We also define the following intermediate matrices:

$$\mathbf{L}^{(k,t)} = \mathbf{L} + \mathbf{M}^{(k,t)} + \sum_{\hat{k}=0}^{k-1} \mathbf{Z}^{(\hat{k})}.$$

Next, we define stopped martingale variables.

**Definition 5.1.16.** *We define the event of the martingale being* safe *until* $(k,t)$*, for* $t \in \{1,\ldots,P+1\}$

$$\boldsymbol{SAFE}^{(k,t)}$$

*to mean that both of the following conditions hold:*

1. *All calls to* SparsifyEulerian *strictly before the kth elimination have been successful, i.e. the approximation errors at each step are small:*

$$\left\| \mathbf{U}_{\mathbf{S}^{(\hat{k},P)}}^{\dagger 1/2} \mathbf{Z}^{(\hat{k})} \mathbf{U}_{\mathbf{S}^{(\hat{k},P)}}^{\dagger 1/2} \right\|_2 \leq \epsilon' \qquad \forall \hat{k} < k.$$

2. *For all indices* $(\hat{k},\hat{t})$ *strictly preceding* $(k,t)$*, i.e.* $(\hat{k},\hat{t}) < (k,t)$*, we had*

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{M}^{(\hat{k},\hat{t})} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon/2.$$

Note that in addition to the events $\mathbf{SAFE}^{(k,1)},\ldots,\mathbf{SAFE}^{(k,P)}$, we also consider event $\mathbf{SAFE}^{(k,P+1)}$. Index $(k,P+1)$ does not correspond to a step of the martingale, but the condition that the martingale has small norm for strictly preceding indices is still well-defined. This notation will later help us prove statements by induction on the ordered indices $(k,1) < (k,2) < \ldots < (k,P) < (k,P+1) < (k+1,1) < \ldots$.

We then define a truncated Martingale as one where the steps incur no additional error once it fails. Its variance steps is given by:

$$\overline{\mathbf{X}}^{(k,t)} \stackrel{\text{def}}{=} \begin{cases} \mathbf{X}^{(k,t)} & \text{if } \mathbf{SAFE}^{(k,t)} \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{5.8}$$

The following sequence of sums of $\overline{\mathbf{X}}^{(k,t)}$ is another zero mean martingale:

$$\overline{\mathbf{M}}^{(k,t)} = \sum_{(\hat{k},\hat{t}) \leq (k,t)} \overline{\mathbf{X}}^{(\hat{k},\hat{t})}.$$

Ultimately, we want to bound the probability of the event that $\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} (\mathbf{L}^{(k_{\max})} - \mathbf{L}) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 > \epsilon$ occurs. Later, we will prove the following lemma.

**Lemma 5.1.17.** *If* $\boldsymbol{SAFE}^{(k,t)}$ *holds,*

- *then we have for all* $(\hat{k},\hat{t}) < (k,t)$*,*

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(\hat{k},\hat{t})} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon, \text{ and } \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(\hat{k})} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon, \tag{5.9}$$

- *and we have for all* $\hat{k} < k$

$$\mathbf{U}_{\mathbf{S}^{(\hat{k})}} \preceq O(1) \cdot \mathbf{U}_{\mathbf{L}}. \tag{5.10}$$

From this lemma, we see that if we can prove $\Pr[\neg\mathbf{SAFE}^{(n+1,1)}] \leq O(\delta)$, it will imply Equation (5.7).

Note that $\left\|\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{M}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\right\|_2 > \epsilon/2$ implies $\left\|\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\overline{\mathbf{M}}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\right\|_2 > \epsilon/2$. Hence, when upper bounding the probability of $\Pr[\neg\mathbf{SAFE}^{(k,t)}]$, we can instead consider the higher probability event

$$\neg\left(\left(\forall\hat{k}\leq k.\left\|\mathbf{U}_{\mathbf{S}^{(\hat{k},P)}}^{\dagger 1/2}\mathbf{Z}^{(\hat{k})}\mathbf{U}_{\mathbf{S}^{(\hat{k},P)}}^{\dagger 1/2}\right\|_2\leq\epsilon'\right)\wedge\left(\forall(\hat{k},\hat{t})\leq(k,P).\left\|\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\overline{\mathbf{M}}^{(\hat{k},\hat{t})}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\right\|_2\leq\epsilon/2\right)\right)$$

The matrix Martingale inequality that we use the rectangular matrix Martingale from Cor 1.3. of [Tro11b].

**Lemma 5.1.18** (Matrix Freedman)**.** *Let $\mathbf{R}^{(1)}\ldots\mathbf{R}^{(N)}$ be a sequence of matrices, and use*

$$\mathbb{E}_{j-1}\left[\mathbf{R}^{(j)}\right]$$

*to denote the expectation of $\mathbf{R}^{(j)}$ conditioned on $\mathbf{R}^{(j-1)},\mathbf{R}^{(j-2)},\ldots,\mathbf{R}^{(1)}$ such that for any $i$*

$$\mathbb{E}_i\left[\mathbf{R}^{(i)}\right]=0$$

*and $\left\|\mathbf{R}^{(i)}\right\|_2\leq\rho$ over its entire support. Then for any error $t$ we have:*

$$Pr\left[\exists k\geq 0\ s.t.\ \left\|\sum_{j\leq k}\mathbf{R}^{(j)}\right\|_2\geq t\ AND\ \left\|\sum_{j\leq k}\mathbb{E}_{j-1}\left[\mathbf{R}^{(j)}\mathbf{R}^{(j)\top}+\mathbf{R}^{(j)\top}\mathbf{R}^{(j)}\right]\right\|_2\leq\sigma^2\right]$$
$$\leq n\cdot\exp\left(\frac{-t^2}{100\left(\sigma^2+t\rho\right)}\right).$$

As we are always normalizing by $\mathbf{U}_{\mathbf{L}}$, we can define rescaled versions of the martingale, as well as variances:

$$\widehat{\mathbf{M}}^{(k,t)}\stackrel{\text{def}}{=}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\overline{\mathbf{M}}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}$$
$$\widehat{\mathbf{X}}^{(k,t)}\stackrel{\text{def}}{=}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\overline{\mathbf{X}}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}$$

Together with a union bound, this allows us to bound $\Pr[\neg\mathbf{SAFE}^{(n+1,1)}]$ by:

$$\Pr[\neg\mathbf{SAFE}^{(n+1,1)}]$$
$$\leq\sum_k\Pr\left[\left\|\mathbf{U}_{\mathbf{S}^{(k,P)}}^{\dagger 1/2}\mathbf{Z}^{(k)}\mathbf{U}_{\mathbf{S}^{(k,P)}}^{\dagger 1/2}\right\|_2>\epsilon'\right] \tag{5.11}$$
$$+\Pr\left[\exists\ (k,t)\ s.t.\ \left\|\widehat{\mathbf{M}}^{(k,t)}\right\|_2\geq s\ \text{AND}\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)}\mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}+\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\right]\right\|_2\leq\sigma^2\right] \tag{5.12}$$
$$+\Pr\left[\exists(k,t)\ s.t.\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)}\mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}+\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\right]\right\|_2>\sigma^2\right]. \tag{5.13}$$

Each call to SparsifyEulerian is made with error probability parameter $\delta/P$ and by Lemma 5.1.13, we call the routine at most $O(P)$ times, so by a union bound and Theorem 5.0.1, the probability at some call fails is at most $O(\delta)$, which bounds the term (5.11).

The other two conditions rely on truncated Martingales, which rely on the condition $\mathbf{SAFE}^{(k,t)}$. Lemma 5.1.17 allows us to bound the error of $\widehat{\mathbf{X}}^{(k,t)}$.

**Lemma 5.1.19.** *We have*

$$\left\|\widehat{\mathbf{X}}^{(k,t)}\right\|_2 \leq O\left(1/P\right)$$

*over the entire support of* $\widehat{\mathbf{X}}^{(k,t)}$ *unconditionally.*

*Proof.* Note that if $\mathbf{SAFE}^{(k,t)}$ no longer holds, the truncation process sets $\widehat{\mathbf{X}}(k,t) = 0$. Otherwise, since by Lemma 5.1.17 we have

$$\mathbf{U}_{local^{(k)}} \preceq \mathbf{U}_{\mathbf{S}^{(k-1)}} \preceq O\left(1\right) \cdot \mathbf{U}_{\mathbf{L}},$$

we then get

$$\left\|\widehat{\mathbf{X}}^{(k,t)}\right\|_2 = \left\|\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{X}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\right\|_2 \leq O\left(1\right)\left\|\mathbf{U}_{local}^{\dagger 1/2}\mathbf{X}^{(k,t)}\mathbf{U}_{local}^{\dagger 1/2}\right\|_2 \leq O\left(1/P\right).$$

Here the last condition follows from the rescaling factor of $1/P$, and the bounds on the error of the single vertex elimination algorithm given by Lemma 5.1.6 Part 3. □

Lemma 5.1.19 says that the steps of $\widehat{\mathbf{M}}^{(k,t)}$ have norm bounded by $O\left(1/P\right)$. This means that Lemma 5.1.18 gives that for $\sigma^2 = \frac{\Theta(\log(1/\delta))}{P}$ and $s = \epsilon^2$, using $P = \Theta(\epsilon^{-2}\log^2(1/\delta))$ and $\log(1/\delta) \geq \Omega(\log n)$, the probability (5.12) is upper bounded by

$$n\exp\left(\frac{-t^2}{100\left(\sigma^2 + t\rho\right)}\right) \leq O(\delta). \tag{5.14}$$

**Lemma 5.1.20.**

$$Pr\left[\exists(k,t)\ s.t.\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)} \mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top} + \widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\right]\right\|_2 > \sigma^2\right] \leq O(\delta).$$

*Proof.*

$$\begin{aligned}
&\Pr\left[\exists(k,t)\ s.t.\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)} \mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top} + \widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\right]\right\|_2 > \sigma^2\right] \\
&\qquad \leq \Pr\left[\exists(k,t)\ s.t.\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)} \mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\right]\right\|_2 > \sigma^2/2\right] \qquad (5.15) \\
&\qquad +\Pr\left[\exists(k,t)\ s.t.\ \left\|\sum_{(\hat{k},\hat{t})\leq(k,t)} \mathbb{E}_{<(\hat{k},\hat{t})}\left[\widehat{\mathbf{X}}^{(\hat{k},\hat{t})}\widehat{\mathbf{X}}^{(\hat{k},\hat{t})\top}\right]\right\|_2 > \sigma^2/2\right] \qquad (5.16)
\end{aligned}$$

We will bound each of these two terms by $O(\delta)$, giving the desired result. Since the proofs for bounding each of the two terms are essentially identical, we will only handle the first of the two terms.

When $\mathbf{SAFE}^{(k,t)}$ does not hold $\widehat{\mathbf{X}}^{(\hat{t},\hat{k})} = \mathbf{0}$. When $\mathbf{SAFE}^{(k,t)}$ holds, by Lemma 5.1.17,

$$\mathbf{U}_{local^{(k)}} \preceq \mathbf{U}_{\mathbf{S}^{(k-1)}} \preceq O\left(1\right) \cdot \mathbf{U}_{\mathbf{L}},$$

and so we can show that

$$\mathbf{X}^{(k,t)\top}\mathbf{U}_{\mathbf{L}}^{\dagger}\mathbf{X}^{(k,t)\top} \preceq O(1)\mathbf{X}^{(k,t)\top}\mathbf{U}_{local^{(k)}}^{\dagger}\mathbf{X}^{(k,t)\top}$$

so that

$$\begin{aligned}\widehat{\mathbf{X}}^{(k,t)\top}\widehat{\mathbf{X}}^{(k,t)} &= \mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\overline{\mathbf{X}}^{(k,t)\top}\mathbf{U}_{\mathbf{L}}^{\dagger}\overline{\mathbf{X}}^{(k,t)}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \\ &\preceq O(1)\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{U}_{local}^{1/2}\mathbf{U}_{local}^{\dagger 1/2}\mathbf{X}^{(k,t)\top}\mathbf{U}_{local^{(k)}}^{\dagger}\mathbf{X}^{(k,t)}\mathbf{U}_{local}^{\dagger 1/2}\mathbf{U}_{local}^{1/2}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \\ &\preceq \frac{O(1)}{P^2}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{U}_{local}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\end{aligned}$$

This bound holds unconditionally, so clearly we also have

$$\mathbb{E}_{<(k,t)}\left[\widehat{\mathbf{X}}^{(k,t)\top}\widehat{\mathbf{X}}^{(k,t)}\right] \preceq \frac{O(1)}{P^2}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{U}_{local}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}.$$

Summing for a fixed $k$ over the $P$ samples made in one round of elimination, we get

$$\sum_t \mathbb{E}_{<(k,t)}\left[\widehat{\mathbf{X}}^{(k,t)\top}\widehat{\mathbf{X}}^{(k,t)}\right] \preceq \frac{O(1)}{P}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}\mathbf{U}_{local}\mathbf{U}_{\mathbf{L}}^{\dagger 1/2}.$$

We define $\mathbf{W}_0 = \mathbf{0}$ and $\mathbf{W}_k = \sum_{\hat{k}\le k}\sum_t \mathbb{E}_{<(\hat{k},t)}\left[\widehat{\mathbf{X}}^{(\hat{k},t)\top}\widehat{\mathbf{X}}^{(\hat{k},t)}\right]$. This gives $\mathbf{0} \preceq \mathbf{W}_k - \mathbf{W}_{k-1}$. Starting from Lemma 5.1.19, direct algebraic manipulations then imply that if $\mathbf{SAFE}^{(k,t)}$ holds, we have

$$\left\|\widehat{\mathbf{X}}^{(k,t)\top}\widehat{\mathbf{X}}^{(k,t)}\right\|_2 \le O\left(1/P^2\right),$$

and when $\mathbf{SAFE}^{(k,t)}$ does not hold, we get this bound trivially from $\widehat{\mathbf{X}}^{(k,t)} = \mathbf{0}$. A triangle inequality then gives

$$\|\mathbf{W}_k - \mathbf{W}_{k-1}\| \le O(P/P^2) = O(1/P) \tag{5.17}$$

and if we let $\mathbb{E}_{<k}[\cdot]$ denote expectation of $\mathbf{W}_k$ over the random choice of vertex to eliminate, conditional on all the random choices of the algorithm until the $k$th elimination, then by Equation (5.6),

$$\mathbb{E}_{<k}[\mathbf{W}_k - \mathbf{W}_{k-1}] \preceq \frac{O(1)}{P(n-k)}\mathbf{\Pi}.$$

Summing over all $\hat{k} \le k$, and using $k \le n/2$ gives

$$\sum_{\hat{k}\le k}\mathbb{E}_{<\hat{k}}[\mathbf{W}_k - \mathbf{W}_{k-1}] \preceq O\left(\frac{1}{P}\right)\mathbf{\Pi}. \tag{5.18}$$

We now construct a zero mean martingale which we will use to bound the probability in term (5.15), by an application of Theorem 3.1.3. Let $\mathbf{V}_k \overset{\text{def}}{=} \mathbf{W}_k - \mathbf{W}_{k-1} - \mathbb{E}_{<k}[\mathbf{W}_k - \mathbf{W}_{k-1}] = \mathbf{W}_k - \mathbb{E}_{<k}[\mathbf{W}_k]$. $\mathbf{V}_k$ is zero-mean conditional on the random choices up to step $k$ and so $\mathbf{R}_k = \sum_{j=1}^k \mathbf{V}_j$ is a zero-mean martingale.

$$\begin{aligned}\mathbf{R}_k &= \sum_{j=1}^k \mathbf{W}_j - \mathbf{W}_{j-1} - \mathop{\mathbb{E}}_{<j}[\mathbf{W}_j - \mathbf{W}_{j-1}] \\ &= \mathbf{W}_k - \sum_{j=1}^k \mathop{\mathbb{E}}_{<j}[\mathbf{W}_j - \mathbf{W}_{j-1}]\end{aligned}$$

Let $\mathbf{H}_k = \sum_{j\le k}\mathbb{E}_{<j}\mathbf{V}_j^2$. In the terminology of Theorem 3.1.3, $\mathbf{R}_k$ is a zero-mean martingale corresponding to $\mathbf{W}_k$, while $\mathbf{V}_k$ is the associated difference sequence, and $\mathbf{H}_k$ is the *predictable quadratic variation process* of $\mathbf{R}_k$.

Note that unconditionally

$$\mathbf{H}_k = \sum_{j\le k} \mathop{\mathbb{E}}_{<j} \mathbf{V}_j^2 = \sum_{j\le k} \mathop{\mathbb{E}}_{<j} \left(\mathbf{W}_j - \mathbf{W}_{j-1} - \mathop{\mathbb{E}}_{j} [\mathbf{W}_j - \mathbf{W}_{j-1}]\right)^2 \preceq \sum_{j\le k} \mathop{\mathbb{E}}_{<j} (\mathbf{W}_j - \mathbf{W}_{j-1})^2$$
$$\preceq \sum_{j\le k} \mathop{\mathbb{E}}_{<j} (\mathbf{W}_j - \mathbf{W}_{j-1}) \|\mathbf{W}_j - \mathbf{W}_{j-1}\| \preceq \frac{O(1)}{P^2}\mathbf{\Pi}. \tag{5.19}$$

Let $\omega^2 = \frac{C}{P^2}$, for some absolute constant $C$ chosen s.t. by Equation (5.19), we have $\Pr[\exists i : \lambda_{\max}(\mathbf{H}_i) > \omega^2] = 0$. Now, by Equation (5.18) we get

$$\begin{aligned}
\Pr[\exists k : \lambda_{\max}(\mathbf{W}_k) > \sigma^2] &= \Pr\left[\exists k : \lambda_{\max}\left(\mathbf{R}_k + \sum_{j=1}^{k} \mathop{\mathbb{E}}_{<j} [\mathbf{W}_j - \mathbf{W}_{j-1}]\right) > \sigma^2\right] \\
&\le \Pr\left[\exists k : \lambda_{\max}(\mathbf{R}_k) > \sigma^2 - \frac{O(1)}{P}\right] \\
&= \Pr\left[\exists k : \lambda_{\max}(\mathbf{R}_k) \ge \sigma^2 - \frac{O(1)}{P} \text{ and } \lambda_{\max}(\mathbf{H}_k) \le \omega^2\right].
\end{aligned}$$

We now want to apply Theorem 3.1.3, to bound the probability above, with $\mathbf{R}_i$ as the zero-mean martingale, $\mathbf{V}_i$ is the associated difference sequence, and $\mathbf{H}_i$ as the predictable quadratic variation process. By Equation (5.17)

$$\begin{aligned}
\|\mathbf{V}_k\| &= \left\|\mathbf{W}_k - \mathbf{W}_{k-1} - \mathop{\mathbb{E}}_{<k} [\mathbf{W}_k - \mathbf{W}_{k-1}]\right\| \\
&\le \max\left\{\|\mathbf{W}_k - \mathbf{W}_{k-1}\|, \left\|\mathop{\mathbb{E}}_{<k} [\mathbf{W}_k - \mathbf{W}_{k-1}]\right\|\right\} \qquad \text{(since both terms are PSD)} \\
&\le \frac{O(1)}{P},
\end{aligned}$$

which gives us a value for the norm control parameter $R$. Thus by Theorem 3.1.3, and using $\sigma^2 = \frac{\Theta(\log(1/\delta))}{P}$, $\log(1/\delta) \ge \Omega(\log n)$, and $\omega^2 = \frac{O(1)}{P^2}$, we get for an appropriate choice of constants that

$$\begin{aligned}
\Pr\left[\exists i : \lambda_{\max}(\mathbf{R}_i) \ge \sigma^2 - \frac{O(1)}{P} \text{ and } \lambda_{\max}(\mathbf{H}_i) \le \omega^2\right] &\le n \exp\left(-\frac{\left(\sigma^2 - \frac{O(1)}{P}\right)^2}{\sigma^2 + \frac{O(1)}{P^2}}\right) \\
&\le \delta.
\end{aligned}$$

This completes the bound on the probability term (5.15), and similarly, we can show the term (5.16) is bounded by $\delta$. □

*Proof of Theorem 5.0.2.* The running time guarantees we need were established in Lemma 5.1.13. Based on Lemma 5.1.17, we observed earlier that $\Pr[\neg\mathbf{SAFE}^{(n+1,1)}] \le O(\delta)$ implies Equation (5.7). Our bounds on each of the terms (5.11), (5.12) (see Equation (5.14)) and (5.13) (see Lemma 5.1.20)) establish this, hence proving the theorem.

□

Finally, we prove Lemma 5.1.17, which we used above to prove Theorem 5.0.2.

*Proof of Lemma 5.1.17.* We consider an ordering $(k,1) < (k,2) < \ldots < (k,P) < (k,P+1) < (k) < (k+1,1) < \ldots$ and so on. We prove by induction on this ordering, that the two claims of Lemma 5.1.17 hold for index $(k,t)$. From

$$\mathbf{L}^{(k,t)} - \mathbf{L} = \mathbf{M}^{(k,t)} + \sum_{\hat{k}<k} \mathbf{Z}^{(\hat{k})}$$

and using a triangle inequality we get

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(k,t)} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{M}^{(k,t)} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 + \sum_{\hat{k}<k} \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{Z}^{(\hat{k})} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 .$$

Our actual induction hypothesis for index $(k,t)$ is that Equation (5.10) hold for that index, and a slightly strengthened version of Equation (5.9) namely that $\mathbf{SAFE}^{(k,t)}$ implies

$$\sum_{\hat{k}<k} \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{Z}^{(\hat{k})} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq C\epsilon \frac{\#\text{ calls to SparsifyEulerian stricly before } (k,t)}{P} \tag{5.20}$$

for some constant $C$ s.t. the guarantee of Lemma 5.1.13 that the number of calls to SparsifyEulerian is $O(P)$ ensures $C\epsilon \frac{\#\text{ calls to SparsifyEulerian}}{P} \leq \epsilon/2$. This in turn gives Equation (5.9), from a triangle inequality combined with Part 2 of Definition 5.1.16.

As our base case, we consider the index $(1,1)$. We call SparsifyEulerian$(\mathbf{L}, \delta/P, \epsilon')$ to compute $\mathbf{S}^{(0)}$, and $\mathbf{SAFE}^{(1,1)}$ guarantees this call succeeded. So Theorem 5.0.1 immediately tells us that $\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{Z}^{(0)} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon' \leq \frac{C\epsilon}{P}$, establishing Equation (5.20) for this index. This in turn gives Equation (5.9), from a triangle inequality combined with Part 2 of Definition 5.1.16:

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(0)} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon.$$

From Lemma 5.1.15, we then immediately get Equation (5.10). Next, we consider proving the inductive statements when $\mathbf{SAFE}^{(k,t+1)}$ holds, assuming the induction hypothesis holds for $\mathbf{SAFE}^{(k,t)}$. In this case, the condition in Equation (5.10) remains unchanged, so it follows immediately from the induction hypothesis for $\mathbf{SAFE}^{(k,t)}$. The sum $\sum_{\hat{k}<k} \left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{Z}^{(\hat{k})} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2$ and upper bound we want for it also remain unchanged, so again we get Equation (5.20). This gives Equation (5.9), from a triangle inequality combined with Part 2 of Definition 5.1.16, i.e. for $(\hat{k},\hat{t}) < (k,t+1)$ we have

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(\hat{k},\hat{t})} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon.$$

Now we consider proving the inductive statements when $\mathbf{SAFE}^{(k+1,1)}$ holds, assuming the induction hypothesis holds for $\mathbf{SAFE}^{(k,P+1)}$. From the induction hypothesis for $\mathbf{SAFE}^{(k,P+1)}$, we have that

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \left( \mathbf{L}^{(k,P)} - \mathbf{L} \right) \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq \epsilon.$$

From Lemma 5.1.15, we then get $\mathbf{U}_{\mathbf{S}^{(k,P)}} \preceq O(1) \cdot \mathbf{U}_{\mathbf{L}}$. This ensures that if a call to SparsifyEulerian was made at the end of $k$th elimination, then since $\mathbf{SAFE}^{(k+1,t)}$ guarantees the call succeeded, we have

$$\left\| \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \mathbf{Z}^{(0)} \mathbf{U}_{\mathbf{L}}^{\dagger 1/2} \right\|_2 \leq O(1) \left\| \mathbf{U}_{\mathbf{S}^{(k,P)}}^{\dagger 1/2} \mathbf{Z}^{(0)} \mathbf{U}_{\mathbf{S}^{(k,P)}}^{\dagger 1/2} \right\|_2 \leq O(1)\epsilon' \leq \frac{C\epsilon}{P}.$$

This then proves Equation (5.20) for $\mathbf{SAFE}^{(k+1,1)}$, which gives Equation (5.9) from a triangle inequality combined with Part 2 of Definition 5.1.16. Finally, by Lemma 5.1.15, we then get $\mathbf{U}_{\mathbf{S}^{(k)}} \preceq O(1) \cdot \mathbf{U}_{\mathbf{L}}$, which proves Equation (5.10) for $\mathbf{SAFE}^{(k+1,1)}$.

□

# Appendix A

## A.1 Schur Complements and Pseudo-inverses

Consider a general PSD matrix of the form

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{I} \end{pmatrix}^\top \tag{A.1}$$

where $\mathbf{A}$ is invertible and $\mathbf{I}$ is the identity matrix on a subset of the indices of $\mathbf{M}$.

Based on Fact 2.1.1 for vectors orthogonal to the null space of $\mathbf{M}$ we have

$$\boldsymbol{x}^\top \mathbf{M}^+ \boldsymbol{x} = \boldsymbol{x}^\top \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{I} \end{pmatrix}^{-\top} \begin{pmatrix} \mathbf{R}^+ & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^+ \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{I} \end{pmatrix}^{-1} \boldsymbol{x}$$

Recall the general formula for blockwise inversion:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{B} & \mathbf{D} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{C}(\mathbf{D} - \mathbf{B}\mathbf{A}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{C}(\mathbf{D} - \mathbf{B}\mathbf{A}^{-1}\mathbf{C})^{-1} \\ -(\mathbf{D} - \mathbf{B}\mathbf{A}^{-1}\mathbf{C})^{-1}\mathbf{B}\mathbf{A}^{-1} & (\mathbf{D} - \mathbf{B}\mathbf{A}^{-1}\mathbf{C})^{-1} \end{pmatrix}$$

By applying the formula for blockwise inversion and simplifying, we get

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{I} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ -\mathbf{B}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix}$$

So

$$\boldsymbol{x}^\top \mathbf{M}^+ \boldsymbol{x} = \boldsymbol{x}^\top \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ -\mathbf{B}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix}^\top \begin{pmatrix} \mathbf{R}^+ & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^+ \end{pmatrix} \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ -\mathbf{B}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix} \boldsymbol{x}.$$

Suppose $\boldsymbol{x} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{y} \end{pmatrix}$, and again $\boldsymbol{x}$ is orthogonal to the null space of $\mathbf{M}$. Then

$$\begin{aligned} \boldsymbol{x}^\top \mathbf{M}^+ \boldsymbol{x} &= \begin{pmatrix} \mathbf{0} \\ \boldsymbol{y} \end{pmatrix}^\top \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ -\mathbf{B}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix}^\top \begin{pmatrix} \mathbf{R}^+ & \mathbf{0} \\ \mathbf{0} & \mathbf{T}^+ \end{pmatrix} \begin{pmatrix} \mathbf{A}^{-1} & \mathbf{0} \\ -\mathbf{B}\mathbf{A}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{0} \\ \boldsymbol{y} \end{pmatrix} \\ &= \boldsymbol{y}^\top \mathbf{T}^+ \boldsymbol{y}. \end{aligned} \tag{A.2}$$

From the above we can conclude the following fact.

**Fact A.1.1** (Schur Complements and Pseudo-Inverses)**.** *Suppose* $\mathbf{U}$ *is an undirected Laplacian. Let* $\mathbf{S} = \textsc{Sc}[\mathbf{U}]_C^\dagger$*. Then* $\mathbf{\Pi_S}(\mathbf{U}^\dagger)_{C,C}\mathbf{\Pi_S} = \textsc{Sc}[\mathbf{U}]_C^\dagger$.

## A.2 Deferred Proofs from Chapter 5

We first show the overall error accumulation from Lemma 5.0.3. The proof relies on the following lemma about the accumulation of errors.

*Proof of Lemma 5.0.3.* Lemma 5.0.2 gives

$$\left\| \mathbf{U}_{\mathbf{S}^{(i_p)}}^{\dagger 1/2} \left( \mathbf{L}^{(i_p)} - \mathbf{L}^{(i_{p+1})} \right) \mathbf{U}_{\mathbf{S}^{(i_p)}}^{\dagger 1/2} \right\|_2 \leq \theta_p \epsilon,$$

for any $0 \leq p < p_{\max}$. By Lemma B.2 of [CKP+16b], we have for every $p$

$$2\vec{x} \left( \mathbf{L}^{(i_p)} - \mathbf{L}^{(i_{p+1})} \right) \vec{y} \leq \theta_p \epsilon \left( \vec{x}^T \mathbf{U}_{\mathbf{S}^{(i_p)}} \vec{x} + \vec{y}^T \mathbf{U}_{\mathbf{S}^{(i_p)}} \vec{y} \right).$$

So summing over these gives

$$2\vec{x} \left( \mathbf{L}^{(0)} - \mathbf{L}^{(i)} \right) \vec{y} \leq \epsilon \left( \vec{x}^T \mathbf{F} \vec{x} + \vec{y}^T \mathbf{F} \vec{y} \right).$$

Again by Lemma B.2 [CKP+16b], this gives

$$\left\| \mathbf{F}^{\dagger 1/2} \left( \mathbf{L}^{(i)} - \mathbf{L}^{(0)} \right) \mathbf{F}^{\dagger 1/2} \right\| \leq \epsilon.$$

□

### A.2.1 Proof of Lemma 5.0.4

In order to prove Lemma 5.0.4, we define a new matrix $\widehat{\mathbf{F}}$, which is made up of the various Schur complements of the final matrix $\mathbf{L}^{(n)}$ onto the corresponding intermediate spaces. Let $I_p = \{i_p \dots n\}$ be the set of vertices remaining after first $p$ phases. Let

$$\widehat{\mathbf{F}} \stackrel{\text{def}}{=} \sum_{0 \leq p < p_{\max}} \theta_p \cdot \mathbf{U}_{\mathrm{Sc}\left[\mathbf{L}^{(n)}\right]_{\{i_p \dots n\}}} \tag{A.3}$$

where $\sum_p \theta_p = 1$ and $\theta_p \geq 1/O(p_{\max}) = 1/O(\log n)$.

Rewriting $\widehat{\mathbf{F}}$ as differences between consecutive steps shows that it is in fact close to $\mathbf{F}$.

**Lemma A.2.1.** *With high probability, the matrices $\mathbf{F}$ and $\widehat{\mathbf{F}}$ as defined in Equations 5.2 and A.3 respectively satisfy:*

$$\widehat{\mathbf{F}} \approx_{O(p_{\max})} \mathbf{F},$$

*where $p_{\max} = O(\log n)$ is the number of invocations of* SinglePhase *(Algorithm 7) by* EulerianLU *(Algorithm 8).*

*Proof.* The key observation is that because the Schur complement steps after step $i_p$ are completely contained among the vertices $\{i_p, \dots n\}$, the difference between $\mathbf{F}$ and $\widehat{\mathbf{F}}$ can be bounded using the discrepancies at the steps.

Formally, the choice of pivots means we have

$$\mathbf{U}_{\mathrm{Sc}\left[\mathbf{L}^{(n)}\right]_{\{i_p \dots n\}}} = \mathbf{U}_{\mathbf{S}^{(i_p)}} + \sum_{p' \geq p} \left( \mathbf{U}_{\mathbf{L}^{(i_{p'})}} - \mathbf{U}_{\mathbf{L}^{(i_{p'+1})}} \right),$$

which when substituted into the formula for $\widehat{\mathbf{F}}$ gives:

$$\widehat{\mathbf{F}} = \sum_{0 \le p < p_{\max}} \theta_p \mathbf{U}_{\mathbf{S}^{(i_p)}} + \sum_{0 \le p < p_{\max}} \sum_{p' \ge p} \theta_p \left( \mathbf{U}_{\mathbf{L}^{(i_{p'})}} - \mathbf{U}_{\mathbf{L}^{(i_{p'+1})}} \right).$$

Collecting the terms related to $\mathbf{F}$, and reversing the summation on the $p'$s turns this into:

$$\widehat{\mathbf{F}} = \mathbf{F} + \sum_{p' \ge p} \left( \sum_{p \le p'} \theta_p \right) \left( \mathbf{U}_{\mathbf{L}^{(i_{p'})}} - \mathbf{U}_{\mathbf{L}^{(n)}} \right).$$

By triangle inequality we then get:

$$\left\| \mathbf{F}^{\dagger 1/2} \left( \widehat{\mathbf{F}} - \mathbf{F} \right) \mathbf{F}^{\dagger 1/2} \right\|_2 \le \sum_{p'} \left( \sum_{p \le p'} \theta_p \right) \left\| \mathbf{F}^{\dagger 1/2} \left( \mathbf{U}_{\mathbf{L}^{(i_{p'})}} - \mathbf{U}_{\mathbf{L}^{(n)}} \right) \mathbf{F}^{\dagger 1/2} \right\|_2$$

Since $\sum \theta_p = 1$, the above is at most $\epsilon p_{\max}$ provided the maximum error over any consecutive sequences of phases is $\epsilon$, which happens WHP by Lemma 5.0.3. Since the $\epsilon$ argument to EulerianLU (Algorithm 8) is required to be $\le 1$, the desired result follows. □

**Lemma A.2.2.** *Let* $\mathbf{L}$ *be a (possibly asymmetric) matrix,* $I_0 = V, \ldots, I_{p_{\max}-1}$ *be nested subsets of indices, i.e.,* $I_0 \subseteq I_{p+1} \subseteq \ldots \subseteq I_{p_{\max}-1}$ *and* $c_0, c_1, \ldots, c_p$ *be constants. Then*

$$\widehat{\mathbf{F}} \preceq \widetilde{\mathbf{L}}^\top \widehat{\mathbf{F}}^\dagger \widetilde{\mathbf{L}},$$

*where* $\widetilde{\mathbf{L}} \stackrel{\text{def}}{=} \mathbf{L}^{(n)}$.

*Proof of Lemma A.2.2.* Let $\mathbf{M}$ be any matrix with $\mathbf{M} + \mathbf{M}^T \succeq 0$ and define

$$\mathbf{C}_{\mathbf{M}} \stackrel{\text{def}}{=} \mathbf{M}^\top \left( \frac{\mathbf{M} + \mathbf{M}^\top}{2} \right)^\dagger \mathbf{M} = \left( \frac{\mathbf{M}^\dagger + \mathbf{M}^{\dagger\top}}{2} \right)^\dagger.$$

By Equation 2.1 and Theorem 2.2 in [Mat92] we have $\mathbf{C}_{\mathbf{M}} \succeq \mathbf{U}_{\mathbf{M}}$ and $\mathbf{C}_{\text{Sc}[\mathbf{M}]_I} = \text{Sc}[\mathbf{C}_{\mathbf{M}}]_I$. Together with the identity $\text{Sc}[\mathbf{P}]_I \preceq \mathbf{P}$ when $\mathbf{P} \succeq 0$, these imply

$$\mathbf{U}_{\text{Sc}[\mathbf{M}]_I} \preceq \mathbf{C}_{\text{Sc}[\mathbf{M}]_I} \preceq \mathbf{C}_{\mathbf{M}}. \tag{A.4}$$

It suffices to show that for any $p$ we have $\mathbf{U}_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_p}} \preceq \widetilde{\mathbf{L}}^\top \widehat{\mathbf{F}}^\dagger \widetilde{\mathbf{L}}$, which is equivalent to

$$\mathbf{U}_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_p}} \preceq \text{Sc}\left[ \widetilde{\mathbf{L}}^\top \widehat{\mathbf{F}}^\dagger \widetilde{\mathbf{L}} \right]_{I_p} = \left( \left( \widetilde{\mathbf{L}}^\top \widehat{\mathbf{F}}^\dagger \widetilde{\mathbf{L}} \right)^\dagger [I_p] \right)^\dagger.$$

Inverting both sides then reduces it to $\mathbf{U}^\dagger_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_p}} \succeq \left( \widetilde{\mathbf{L}}^\dagger \widehat{\mathbf{F}} \widetilde{\mathbf{L}}^{\dagger\top} \right)[I_p]$. Using the definition of $\widehat{\mathbf{F}}$ gives

$$\left( \widetilde{\mathbf{L}}^\dagger \widehat{\mathbf{F}} \widetilde{\mathbf{L}}^{\dagger\top} \right)[I_p] = \sum_{0 \le p' < p_{\max}} c_{p'} \left( \widetilde{\mathbf{L}}^\dagger \mathbf{U}_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_{p'}}} \widetilde{\mathbf{L}}^{\dagger\top} \right)[I_p]$$

Now we consider the terms separately and show that $\left( \widetilde{\mathbf{L}}^\dagger \mathbf{U}_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_{p'}}} \widetilde{\mathbf{L}}^{\dagger\top} \right)[I_p] \preceq \mathbf{U}^\dagger_{\text{Sc}[\widetilde{\mathbf{L}}]_{I_p}}$ for every $p'$, which will imply the lemma. There are two cases.

$p \leq p'$ :

By Equation A.4, we have

$$\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p] \preceq \left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p] \preceq \left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p].$$

As the support of $\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}$ is also restricted onto $I_p$, we can replace $\widetilde{\mathbf{L}}^{\dagger}$ and $\widetilde{\mathbf{L}}^{\dagger\top}$ with their restrictions onto $I_p$. Using $\widetilde{\mathbf{L}}^{\dagger}[I_p] = \mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}$, we have

$$\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p] = \mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]^{\dagger}_{I_p}\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}\mathbf{U}^{\dagger}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]^{\top}_{I_p}\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]^{\dagger\top}_{I_p} \tag{A.5}$$

$$= \mathbf{U}^{\dagger}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}. \tag{A.6}$$

$p' < p$ :

As $I_p \subseteq I_{p'}$, we can write this operator as

$$\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p] = \left(\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_{p'}]\right)[I_p]$$

Once again, the restriction to $I_{p'}$ in both the middle and the outer terms means

$$\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_{p'}] = \mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]^{\dagger}_{I_{p'}}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}} = \left(\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\right)^{\dagger}.$$

Plugging this back in, and applying the identity about Schur complements of subsets of vertices gives:

$$\left(\widetilde{\mathbf{L}}^{\dagger}\mathbf{U}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\widetilde{\mathbf{L}}^{\dagger\top}\right)[I_p] = \left(\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_{p'}}}\right)^{\dagger}[I_p] = \mathbf{C}_{\widetilde{\mathbf{L}}}[I_p] = \left(\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}\right)^{\dagger}.$$

Now, from $\mathbf{C}_{\mathbf{M}} \succeq \mathbf{U}_{\mathbf{M}}$ we have $\left(\mathbf{C}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}\right)^{\dagger} \preceq \mathbf{U}^{\dagger}_{\mathrm{Sc}\left[\widetilde{\mathbf{L}}\right]_{I_p}}$.

□

We can now conclude things formally.

*Proof of Lemma 5.0.4.* By Lemma A.2.2, $\widehat{\mathbf{F}} \preceq \left(\mathbf{L}^{(n)}\right)^{\top}\widehat{\mathbf{F}}^{\dagger}\mathbf{L}^{(n)}$. By Lemma A.2.1, we have w.h.p. that $\widehat{\mathbf{F}} \approx_{O(\log(n))} \mathbf{F}$. Thus, we have

$$1/O(\log^2 n) \cdot \mathbf{F} \preceq \left(\mathbf{L}^{(n)}\right)^{\top}\mathbf{F}^{\dagger}\mathbf{L}^{(n)}$$

□

## A.3 Lemmas about Finding RCDD Sets

We now give some lemmas which we use to prove Theorem 5.1.1, which says that we can quickly find $\alpha$-RCDD sets in a directed graph.

**Lemma A.3.1.** *Let $\mathbf{L} \in \mathbb{R}^{n \times n}$ be an Eulerian Laplacian and let $F \subseteq V$ be a random subset of size $k$. Then the expected number of $i \in F$ such that $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ is at most $k^2(1+\alpha)/n$.*

*Proof.* We have that for all $j \neq i$

$$\Pr\left[j \notin F \,|\, i \in F\right] = \prod_{i \in [k-1]} \left(1 - \frac{1}{n-i}\right) = \prod_{i \in [k-1]} \left(\frac{n-i-1}{n-i}\right) = \frac{n-k}{n-1}$$

and therefore $\Pr\left[j \in F \,|\, i \in F\right] = \frac{k-1}{n-1}$. Since $\sum_{j \in F : j \neq i} |\mathbf{L}_{ij}| = \mathbf{L}_{ii}$ we have that $\mathbb{E}\left[\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \,|\,\right] = \left(\frac{k-1}{n-1}\right) \mathbf{L}_{ii}$ and that by Markov's inequality

$$\Pr\left[\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha} \mathbf{L}_{ii} \,|\, i \in F\right] \leq \left(\frac{k-1}{n-1}\right)(1+\alpha)$$

and since

$$\Pr\left[i \notin F\right] = \prod_{i \in [k]} \left(1 - \frac{1}{n+1-i}\right) = \prod_{i \in [k]} \frac{n-i}{n+1-i} = \frac{n-k}{n} = 1 - \frac{k}{n}$$

we have that $\Pr[i \in F] = \frac{k}{n}$ and the expected number $i \in F$ such that $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha}$ is at most

$$\begin{aligned}\sum_{i \in [n]} \Pr\left[i \in F, \sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha}\right] &= \sum_{i \in [n]} \Pr\left[i \in F\right] \Pr\left[\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha} \mathbf{L}_{ii} \,|\, i \in F\right] \\ &\leq k \left(\frac{k-1}{n-1}\right)(1+\alpha)\end{aligned}$$

Since $k \leq n$ we have $(k-1)/(n-1) \leq k/n$ and the result follows. □

**Lemma A.3.2.** *Let $\mathbf{L} \in \mathbb{R}^{n \times n}$ be an Eulerian Laplacian, let $F \subseteq V$ be a random subset of size $k$ and let $F' \subseteq V$ be the elements $i \in F$ for which $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \leq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ and $\sum_{j \in F, j \neq i} |\mathbf{L}_{ji}| \leq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ then with probability at least $1/2$ we have*

$$|F'| \geq k \left[1 - \frac{4k}{(1+\alpha)n}\right]$$

*and therefore for $k = \frac{n}{8(1+\alpha)}$, $\mathbf{L}_{F'F'}$ is $\alpha$-RCDD with $|F'| \geq \frac{n}{16(1+\alpha)}$ with probability at least $1/2$.*

*Proof.* Applying Lemma A.3.1 to $\mathbf{L}$ and $\mathbf{L}^\top$ we see that the expected number of elements $i \in F$ for which $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ is at most $k^2(1+\alpha)/n$ and the expected number of elements $i \in F$ for which $\sum_{j \in F, j \neq i} |\mathbf{L}_{ij}| \geq \frac{1}{1+\alpha} |\mathbf{L}_{ii}|$ is at most $k^2(1+\alpha)/n$ consequently an expected $2k^2(1+\alpha)/n$ are removed from $F$ to get $F'$. Consequently, by Markov's inequality with probability at least $1/2$ at most $2k^2(1+\alpha)/n$ are removed from $F$ to get $F'$ □

## A.4 Matrix Facts

**Lemma A.4.1.** *Let* $\widetilde{\mathbf{L}}, \mathbf{L}, \mathbf{F}$ *be arbitrary matrices with* $\ker(\widetilde{\mathbf{L}}) = \ker(\widetilde{\mathbf{L}}^\intercal) = \ker(\mathbf{L}) = \ker(\mathbf{L}^\intercal) = \ker(\mathbf{F}) = \ker(\mathbf{F}^\intercal)$. *Suppose* $\|\mathbf{F}^{+/2}(\mathbf{L} - \widetilde{\mathbf{L}})\mathbf{F}^{+/2}\| \leq \epsilon$ *and that* $\gamma\mathbf{F} \preceq \widetilde{\mathbf{L}}^\intercal\mathbf{F}^+\widetilde{\mathbf{L}}$. *Then* $\mathbf{L}^\intercal\mathbf{F}^+\mathbf{L} \approx_{O\left(\frac{\epsilon}{\sqrt{\gamma}}+\frac{\epsilon^2}{\gamma}\right)} \widetilde{\mathbf{L}}^\intercal\mathbf{F}^+\widetilde{\mathbf{L}}$.

*Proof.* We have

$$\begin{aligned}
\|\mathbf{F}^{+/2}(\mathbf{L} - \widetilde{\mathbf{L}})\mathbf{F}^{+/2}\| &\leq \epsilon \\
\|(\mathbf{L} - \widetilde{\mathbf{L}})x\|_{\mathbf{F}^+} &\leq \epsilon \cdot \|x\|_{\mathbf{F}} && \forall x \\
\left|\|\mathbf{L}x\|_{\mathbf{F}^+} - \|\widetilde{\mathbf{L}}x\|_{\mathbf{F}^+}\right| &\leq \epsilon/\sqrt{\gamma} \cdot \|\widetilde{\mathbf{L}}x\|_{\mathbf{F}^+} && \forall x \\
\left|x^\intercal\mathbf{L}^\intercal\mathbf{F}^+\mathbf{L}x - x^\intercal\widetilde{\mathbf{L}}^\intercal\mathbf{F}^+\widetilde{\mathbf{L}}x\right| &\leq O\left(\frac{\epsilon}{\sqrt{\gamma}} + \frac{\epsilon^2}{\gamma}\right) \cdot x^\intercal\widetilde{\mathbf{L}}^\intercal\mathbf{F}^+\widetilde{\mathbf{L}}x && \forall x,
\end{aligned}$$

which is one definition of the desired condition. □

# Bibliography


[ADK+16] Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. *CoRR*, abs/1604.02094, 2016. Available at: http://arxiv.org/abs/1604.02094.

[AGM+10] Arash Asadpour, Michel X. Goemans, Aleksander Mądry, Shayan Oveis Gharan, and Amin Saberi. An o(log n/ log log n)-approximation algorithm for the asymmetric traveling salesman problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 379–389, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[Ald90] David Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. In *SIAM Journal on Discrete Mathematics*, pages 450–465, 1990.

[BGH+06] Marshall Bern, John R. Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006.

[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(6):3264–3284, 2008.

[BP12] Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. In *Automata, Languages, and Programming*, pages 133–144. Springer, 2012.

[Bro89] Andrei Broder. Generating random spanning trees. In *Proceedings of the 30th annual Symposium on Foundations of Computer Science, FOCS 1989*, pages 442–447, 1989.

[Cal99] Ronald Calinger. *A Contextual History of Mathematics*. Prentice Hall, 1999.

[CDN89] Charles J Colbourn, Robert PJ Day, and Louis D Nel. Unranking and ranking spanning trees of a graph. *Journal of Algorithms*, 10(2):271–286, 1989.

[CKK+17] Michael B. Cohen, Jonathan A. Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. Manuscript, 2017.

[CKM+11] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, STOC '11, pages 273–282, New York, NY, USA, 2011. ACM.

[CKM+14a] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 343–352, New York, NY, USA, 2014. ACM.

[CKM+14b] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 343–352, New York, NY, USA, 2014. ACM.

[CKP+16a] Michael B. Cohen, Jon Kelner, John Peebles, Rirhcard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 583–592, Oct 2016.

[CKP+16b] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. Accepted to *STOC 2017*. Preprint available at https://arxiv.org/abs/1611.00755., 2016.

[Cla03] Kenneth L. Clarkson. Solution of linear systems using randomized rounding, 2003.

[CLM+15] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 181–190, New York, NY, USA, 2015. ACM.

[CMN96] Charles J Colbourn, Wendy J Myrvold, and Eugene Neufeld. Two algorithms for unranking arborescences. *Journal of Algorithms*, 20(2):268–281, 1996.

[CMSV17] Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}$(m10/7 log w) time: (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 752–771, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[DKP+16] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. Accepted to *STOC 2017*. Preprint available at http://arxiv.org/abs/1611.07451., 2016.

[DS08a] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 451–460, New York, NY, USA, 2008. ACM.

[DS08b] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *STOC '08*, STOC '08, pages 451–460, 2008.

[Fre75] David A. Freedman. On tail probabilities for martingales. *Ann. Probab.*, 3(1):100–118, 02 1975.

[Gre96] Keith Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996. CMU CS Tech Report CMU-CS-96-123.

[GRV09] Navin Goyal, Luis Rademacher, and Santosh Vempala. Expanders via random spanning trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 576–585, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[GSS11] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 550–559, Washington, DC, USA, 2011. IEEE Computer Society.

[Gua97] Stephen Guattery. Graph embedding techniques for bounding condition numbers of incomplete factor preconditioning. Technical report, ICASE, 1997.

[Gue83] Alain Guenoche. Random spanning tree. *Journal of Algorithms*, 4(3):214–220, 1983.

[Gus78] Ivar Gustafsson. A class of first order factorization methods. *BIT Numerical Mathematics*, 18(2):142–156, 1978.

[Hig02] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. Siam, 2002.

[HX16] Nicholas J. A. Harvey and Keyulu Xu. Generating random spanning trees via fast matrix multiplication. In *LATIN 2016: Theoretical Informatics*, volume 9644, pages 522–535, 2016.

[Kir47] Gustav Kirchhoff. *Ü*ber die auflösung der gliechungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. In *Poggendorgs Ann. Phys. Chem.*, pages 497–508, 1847.

[KLOS14] 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. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

[KLP+16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 842–850. ACM, 2016. Available at http://arxiv.org/abs/1512.01892.

[KM09a] Jonathan Kelner and Aleksander Madry. Faster generation of random spanning trees. In *Proceedings of the 50th annual Symposium on Foundations of Computer Science, FOCS 2009*, pages 13–21, 2009. Available at https://arxiv.org/abs/0908.1448.

[KM09b] Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *FOCS '09*, pages 13–21, Washington, DC, USA, 2009. IEEE Computer Society.

[KM11] Jonathan Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *Theor. Comput. Sci.*, 412(32):4123–4135, July 2011.

[KMP10] I. Koutis, G.L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 235 –244, 2010.

[KMP11] I. Koutis, G.L. Miller, and R. Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Foundations of Computer Science (FOCS), 2011 52nd Annual IEEE Symposium on*, pages 590–598, 2011.

[KMP12] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1–18, New York, NY, USA, 2012. ACM.

[KOSZ13] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 911–920. ACM, 2013.

[KR93] Douglas J Klein and Milan Randic. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, 1993.

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *Proceedings of the 57th annual Symposium on Foundations of Computer Science, FOCS 2016*, 2016. Available at https://arxiv.org/pdf/1605.02353v1.pdf.

[Kul90] Vidyadhar G. Kulkarni. Generating random combinatorial objects. *Journal of Algorithms*, 11(2):185–207, 1990.

[LPS15] Yin Tat Lee, Richard Peng, and Daniel A. Spielman. Sparsified cholesky solvers for SDD linear systems. *CoRR*, abs/1506.08204, 2015.

[LS13] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 147–156. IEEE, 2013.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}\sqrt{rank}$ iterations and faster algorithms for maximum flow. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 424–433. IEEE, 2014. Available at http://arxiv.org/abs/1312.6677 and http://arxiv.org/abs/1312.6713.

[Mad10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 245–254, 2010.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262, 2013.

[Mad16] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 593–602, 2016.

[Mat92] Roy Mathias. Matrices with positive definite hermitian part: Inequalities and linear systems. *SIAM journal on matrix analysis and applications*, 13(2):640–654, 1992.

[MST15a] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 2019–2036, 2015. Available at http://arxiv.org/pdf/1501.00267v1.pdf.

[MST15b] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2019–2036. Society for Industrial and Applied Mathematics, 2015.

[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. In *STOC '12*, pages 1141–1160, New York, NY, USA, 2012. ACM.

[PS14] Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 333–342, New York, NY, USA, 2014. ACM. Available at http://arxiv.org/abs/1311.3286.

[She09] Jonah Sherman. Breaking the multicommodity flow barrier for $O\left(\sqrt{\log n}\right)$-approximations to sparsest cut. In *FOCS 2009*, 2009.

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 263–269, Washington, DC, USA, 2013. IEEE Computer Society.

[SS11a] D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[SS11b] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. Available at http://arxiv.org/abs/0803.0929.

[SS11c] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. Announced at STOC'08.

[ST04a] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 81–90, New York, NY, USA, 2004. ACM.

[ST04b] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[ST14a] Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014. Available at http://arxiv.org/abs/cs/0607105.

[ST14b] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM. J. Matrix Anal. & Appl.*, 35:835âĂŞ885, 2014.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, August 1969.

[Str86] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[TBI97] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[Ten10] Shang-Hua Teng. The Laplacian Paradigm: Emerging Algorithms for Massive Graphs. In *Theory and Applications of Models of Computation*, pages 2–14, 2010.

[Tro11a] Joel Tropp. Freedman's inequality for matrix martingales. *Electron. Commun. Probab.*, 16:no. 25, 262–270, 2011.

[Tro11b] Joel A Tropp. Freedman's inequality for matrix martingales. *arXiv preprint arXiv:1101.3039*, 2011. Available at: https://arxiv.org/abs/1101.3039.

[Wal77] Alastair J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, September 1977.

[Wil96] David Bruce Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 296–303, New York, NY, USA, 1996. ACM.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of computing*, pages 887–898. ACM, 2012. Available at: http://theory.stanford.edu/~virgi/matrixmult-f.pdf.

[ZBL+04] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16(16):321–328, 2004.

[ZGL03] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. *ICML*, 2003.

[ZS04] Dengyong Zhou and Bernhard Schölkopf. A regularization framework for learning from graph data. In *ICML workshop on statistical relational learning and Its connections to other fields*, volume 15, pages 67–68, 2004.