Graphs and Beyond: Faster Algorithms for High Dimensional Convex Optimization

Jakub Pachocki

CMU-CS-16-107
May 2016

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

This research was sponsored by the National Science Foundation under grant numbers CCF-1018463 and CCF-1065106 and by the Sansom Graduate Fellowship in Computer Science. Parts of this work were done while at the Simons Institute for for the Theory of Computing, UC Berkeley. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Convex Optimization, Linear System Solvers, Spectral Graph Theory

To my parents

Abstract

Convex optimization is one of the most robust tools for automated data analysis. It has widespread applications in fields such as machine learning, computer vision, combinatorial optimization and scientific computing. However, the rapidly increasing volume and complexity of data that needs to be processed often renders general-purpose algorithms unusable.

This thesis aims to address this issue through development of very fast algorithms for core convex optimization problems, with a focus on graphs. To this end, we:

- Develop nearly optimal algorithms for the Fermat-Weber (geometric median) problem, a fundamental task in robust estimation and clustering.
- Investigate how to approximate massive amounts of high-dimensional data in a restrictive streaming setting, achieving optimal tradeoffs.
- Develop the fastest algorithm for solving linear systems on undirected graphs. This is achieved through improved clustering algorithms and better understanding of the interplay between combinatorial and numerical aspects of previous approaches.
- Show the existence of clustering and oblivious routing algorithms for a broad family of directed graphs, in hopes of advancing the search for faster maximum flow algorithms.

Most of the presented algorithms work in time nearly linear in the sparsity of the input data. The unifying theme of these results is careful analysis of optimization algorithms in high-dimensional spaces, and mixing combinatorial and numerical approaches.

Acknowledgments

First and foremost, I would like to thank my advisor, Gary Miller. His mentorship and support have been invaluable and fundamentally shaped the course of my studies.

I am extremely grateful to the many co-authors I have been fortunate to work with over the past three years. Many of the ideas of this thesis have been shaped by long and fruitful collaborations with Michael Cohen, Richard Peng and Aaron Sidford. The results presented here represent joint work with Michael Cohen, Alina Ene, Rasmus Kyng, Yin-Tat Lee, Gary Miller, Cameron Musco, Richard Peng, Anup Rao, Aaron Sidford and Shen Chen Xu.

I would like to thank my thesis committee members, Anupam Gupta, Daniel Sleator and Shang-Hua Teng for help and advice during the dissertation process. I am grateful to Daniel Sleator for his guidance throughout my entire stay in graduate school.

I am indebted to my high school teacher, Ryszard Szubartowski, for igniting my interest in computer science. I am grateful for the countless discussions, his infectious enthusiasm and unwavering support.

Finally, I would like to thank my friends for their help and encouragement, and my family for their love and support.

Contents

List of Figures

Chapter 1

Introduction

The amount and complexity of objects subject to automated analysis increases at a rapid pace. Tasks such as processing satellite imagery in real time, predicting social interactions and semi-automated medical diagnostics become a reality of everyday computing. In this thesis, we shall explore multiple facets of the fundamental problem of analyzing such data effectively.

1.1 Dealing with High Dimensional Data

In order to enable rigorous reasoning, we will associate our problems with a space $\mathbb{R}^d$ such that all the features of each object in our problem class can be conveniently represented as d real numbers. We refer to the integer d as the *dimensionality* of our problem.

The different dimensions can correspond to pixels in an image; users in a social network; characteristics of a hospital patient. Often, the complexity of the problems of interest translates directly to high dimensionality.

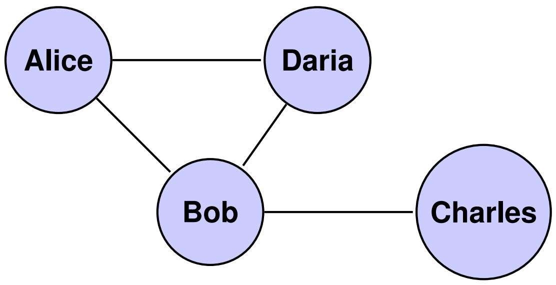
1.1.1 Representation

The most common way to describe a d-dimensional problem involving n objects is simply as an $n \times d$ real matrix. In the case of images, the numerical values might describe the colors of pixels. In a graph, they might be nonzero only for the endpoints of a particular edge (see Figure 1.1). In machine learning applications, they might be 0-1, describing the presence or absence of binary features.

1.1.2 Formulating Convex Objectives

Given a dataset, we want to conduct some form of reasoning. Examples of possible questions include:

- What is the community structure of this social network?
- What does this image look like without the noise?
- What are the characteristics of a typical patient?



$$\begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \\ \end{array}$$
$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \end{bmatrix}$$

Figure 1.1: Social network as graph; corresponding matrix $\mathbf{B}$.

- How likely is this user to like a given movie?

To make these problems concrete, we turn such questions into *optimization problems*. This amounts to defining a set S of feasible solutions and an objective function f. For the above examples:

- (Clustering a social network.) S is the possible clusterings of the nodes of the network; f measures the quality of a clustering.
- (Denoising an image.) S is the set of all images of a given size; f is the similarity of the image to the original, traded off against its smoothness.
- (Finding a typical patient.) S are the possible characteristics of a patient; f measures the averaged difference with the patients in the database.
- (Predicting whether a user likes a movie.) S are the possible models for predicting user preferences; f measures the prediction quality of a model, traded off against its simplicity.

The problem can then be stated succinctly as

$$\begin{aligned} &\text{minimize } f(\boldsymbol{x}) \\ &\text{subject to } \boldsymbol{x} \in S. \end{aligned} \tag{1.1}$$

In order to design efficient algorithms for finding the optimal (or otherwise satisfactory) solution $\boldsymbol{x}$ to the above problem, we need to make further assumptions on f and S. In this thesis, we focus on the case where f and S are *convex*. It is useful to note that in some problem domains, non-convex models such as deep learning yield better results in practice.

The primary reason for restricting ourselves to convex models is the need for solving the optimization problem (1.1) efficiently, while retaining fairly large expressive power of the model. There exist general polynomial time methods for optimizing convex functions, see e.g. [LSW15].

In addition to serving as an objective in and of itself, the convex optimization problem can also serve as a proxy for optimizing a harder, non-convex objective (see e.g. [Can06]).

1.1.3 Working with Massive Data

The focus of this thesis is on problems involving large amounts of data. The fundamental idea in analyzing and reasoning from large datasets is that even fairly simple models perform very well given enough information. However, given limited computational resources, even polynomial time algorithms may prove insufficient when processing massive data. This leads to the need for *scalable* algorithms: ones working in time nearly linear in the input size[1].

We explore multiple approaches to designing such algorithms:

- Through improved analysis of optimization methods (Chapters 2 and 5 and Section 6.5).
- Through improved understanding of the combinatorial structure of the data (Chapters 4 and 6).
- Through subsampling the data (Chapters 3 and 5 and Section 2.5).
- Through operating in a streaming setting (Chapter 3).

1.2 Convex Optimization and Solving Linear Systems

1.2.1 Gradient Descent and Beyond

Gradient descent, or more generally steepest descent, is one of the most basic strategies for optimizing convex and non-convex objectives alike. It amounts to repeatedly nudging the solution vector $\boldsymbol{x}$ in the direction in which f decreases most quickly. For some convex functions, it can be shown to converge quickly. We give specialized analyses of various gradient descent algorithms in Chapter 5 and Sections 2.5 and 6.5.

Gradient descent is a *first order* optimization method: it only uses the information about the first derivative (the gradient) of f. An extremely powerful idea in designing fast algorithms for optimizing convex function is utilizing information about the second derivative (the Hessian) of f. This leads to *second order* methods, such as Newton's method.

Interior point is a general method for solving constrained convex optimization problems, utilizing Newton's method as a subroutine. In Chapter 2, we give the first analysis of this method that shows it to run in nearly linear time in the input size for a particular natural problem. The results of Chapters 4 and 5, when utilized as subroutines in interior point methods, lead to the fastest theoretical runtimes for many fundamental graph problems [Mad13, LS14].

The idea of Newton's method is to locally approximate the function by a quadratic polynomial given by the Hessian and the gradient. Each step of the method amounts to minimizing this approximation and moving towards the minimizer. This requires solving a linear system, which is typically the bottleneck in the runtime of the entire algorithm. A large part of this thesis focuses on solving linear systems efficiently (Chapters 3 to 5).

[1]We say that an algorithm works in *nearly linear time* in the input size m if it works in $\mathcal{O}(m \log^c m)$ time for some constant c.

1.2.2 Spectral Approximation and Row Sampling

The problem of solving a linear system $\boldsymbol{Ax} = \boldsymbol{b}$ is equivalent to the following unconstrained convex optimization problem, called 'least squares':

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^\top \boldsymbol{b} \\ &= \|\boldsymbol{x}\|^2_{\boldsymbol{A}^\top \boldsymbol{A}} - 2\boldsymbol{x}^\top \boldsymbol{b}. \end{aligned}$$

A useful approach to solving this problem for a very large matrix $\boldsymbol{A}$ is to replace it with a *spectral approximation* $\tilde{\boldsymbol{A}}$ such that

$$\|\boldsymbol{y}\|_{\tilde{\boldsymbol{A}}^\top \tilde{\boldsymbol{A}}} \approx_{1+\epsilon} \|\boldsymbol{y}\|_{\boldsymbol{A}^\top \boldsymbol{A}}$$

for all $\boldsymbol{y}$. Chapter 3 deals exclusively with computing such approximations efficiently through *row sampling*, that is selecting a random subset of rows of $\boldsymbol{A}$ to form $\tilde{\boldsymbol{A}}$.

Given $\tilde{\boldsymbol{A}}$, a very accurate solution to the original linear system can be obtained by coupling solving linear systems in $\tilde{\boldsymbol{A}}$ with gradient descent; this method is called preconditioned Richardson iteration. In Chapter 5 we show how to use a method similar to Richardson iteration and subsampling to solve linear systems accurately, without actually obtaining spectral approximations along the way (and thus obtaining better running time).

1.3 Structure of This Thesis

1.3.1 Geometric Median

[CLM+16] In Chapter 2, we investigate a fundamental problem in Euclidean space: the geometric median. It is a basic tool for robust estimation and clustering. While scalable low-dimensional approximations are known, the general problem has not been efficiently solvable. We give the fastest known methods, motivated by recent advancements in multiplicative weights, interior point methods and stochastic gradient descent.

1.3.2 Online Sampling

[CMP15] In Chapter 3, we discuss a new algorithm for subsampling general matrices, allowing us to deal with amounts of data that do not necessarily fit in memory. We show that it is possible to perform surprisingly well while subsampling a matrix in a very restrictive streaming setting. We also show the optimality of our methods.

1.3.3 Laplacian Solvers

[CKM+14, CMP+14, CKP+14] One natural kind of innately high dimensional structure is the graph, where only pairs of dimensions are directly related. Graphs admit combinatorial analysis, which can be applied in conjunction with new numerical techniques to obtain very efficient algorithms. The first nearly linear time solvers for Laplacian systems were developed by Spielman and Teng [ST04a], starting the field of modern algorithmic spectral graph theory. Their runtime was later improved by Koutis, Miller and Peng [KMP10, KMP11], with the fastest iterative algorithm working in time $\mathcal{O}(m \log n)$.

In Chapter 4, we study new clustering algorithms for graphs, with the goal of improving linear system solvers and solving other optimization problems, such as maximum flow.

In Chapter 5, we turn to solving the linear systems themselves. We find tighter connections between the combinatorial and numerical approaches, leading us to develop the fastest known solver of Laplacian systems, working in $\mathcal{O}(m\sqrt{\log n})$ time.

1.3.4 Directed Graphs

[EMPS16] The combinatorial structure of directed graphs remains much less understood from the spectral viewpoint. It is of great interest to understand it better, and generalize some of the algorithms fundamental to recent progress on undirected graphs. In Chapter 6, we give new algorithms for oblivious routings and maximum flow on directed graphs parameterized by their *balance*.

Chapter 2

Geometric Median

The geometric median of a finite set of points in Euclidean space $S = \{p_1, \ldots, p_n\} \in \mathbb{R}^d$ is the point $x \in \mathbb{R}^d$ minimizing the sum of distances to the points. Despite its simple statement, the problem of finding the geometric median for a given S has been a subject of active research for many years. It is otherwise known as the the Fermat-Weber problem.

The Fermat-Weber problem is equivalent to minimizing $f_S : \mathbb{R}^d \to \mathbb{R}$ given as:

$$f_S(x) := \sum_{i=1}^{n} \|x - p_i\|_2.$$

Since f_S is convex for any S, this problem can be solved in polynomial time using convex optimization.

We demonstrate an interior point method that solves the Fermat-Weber problem in $\tilde{\mathcal{O}}(nd \log^3(1/\epsilon))$ time, which is close to optimal. We also show a stochastic gradient descent algorithm, working in time $\tilde{\mathcal{O}}(d/\epsilon^2)$, independent of the number of input points.

Some of our ideas are inspired by recent developments in optimization methods for graph problems [CKM+11]; it is an interesting question for future research whether the tools we construct can be brought back and applied to graphs.

The results presented in this chapter are joint work with Michael Cohen, Yin-Tat Lee, Gary Miller and Aaron Sidford [CLM+16].

2.1 Introduction

One of the oldest easily-stated nontrivial problems in computational geometry is the Fermat-Weber problem: given a set of n points in d *dimensions*, $a^{(1)}, \ldots, a^{(n)} \in \mathbb{R}^d$, find a point $x^* \in \mathbb{R}^d$ that minimizes the sum of Euclidean distances to them:

$$x^* \in \operatorname{argmin}_{x \in \mathbb{R}^d} f(x) \quad \text{where} \quad f(x) \stackrel{\text{def}}{=} \sum_{i \in [n]} \|x - a^{(i)}\|_2$$

This problem, also known as the *geometric median problem,* is well studied and has numerous applications. It is often considered over low dimensional spaces in the context of the facility location problem [Web09] and over higher dimensional spaces it has applications to clustering in machine learning and data analysis. For example, computing the geometric median is a subroutine in popular expectation maximization heuristics for k-medians clustering.

The problem is also important to robust estimation, where we seek to find a point representative of given set of points that is resistant to outliers. The geometric median is a rotation and translation invariant estimator that achieves the optimal *breakdown point* of 0.5, i.e. it is a good estimator even when up to half of the input data is arbitrarily corrupted [LR91]. Moreover, if a large constant fraction of the points lie in a ball of diameter ϵ then the geometric median lies within distance $O(\epsilon)$ of that ball (see Lemma 2.5.1). Consequently, the geometric median can be used to turn expected results into high probability results: e.g. if the $a^{(i)}$ are drawn independently such that $\mathbb{E}\|x - x^*\|_2 \leq \epsilon$ for some $\epsilon > 0$ and $x \in \mathbb{R}^d$ then this fact, Markov bound, and Chernoff Bound, imply $\left\|x^* - a^{(i)}\right\|_2 = O(\epsilon)$ with high probability in n.

Despite the ancient nature of the Fermat-Weber problem and its many uses there are relatively few theoretical guarantees for solving it (see Table 2.1). To compute a $(1+\epsilon)$-approximate solution, i.e. $x \in \mathbb{R}^d$ with $f(x) \leq (1+\epsilon)f(x^*)$, the previous fastest running times were either $O(d \cdot n^{4/3}\epsilon^{-8/3})$ by [CMMP13], $\tilde{O}(d \exp \epsilon^{-4} \log \epsilon^{-1})$ by [BHI02], $\tilde{O}(nd + \text{poly}(d, \epsilon^{-1}))$ by [FL11], or $O((nd)^{O(1)} \log \frac{1}{\epsilon})$ time by [PS01, XY97]. In this work we improve upon these running times by providing an $O(nd \log^3 \frac{n}{\epsilon})$ time algorithm[1] as well as an $O(d/\epsilon^2)$ time algorithm, provided we have an oracle for sampling a random $a^{(i)}$. Picking the faster algorithm for the particular value of ϵ improves the running time to $O(nd \log^3 \frac{1}{\epsilon})$. We also extend these results to compute a $(1+\epsilon)$-approximate solution to the more general Weber's problem, $\min_{x \in \mathbb{R}^d} \sum_{i \in [n]} w_i \left\|x - a^{(i)}\right\|_2$ for non-negative w_i, in time $O(nd \log^3 \frac{1}{\epsilon})$ (see Section 2.6).

Our $O(nd \log^3 \frac{n}{\epsilon})$ time algorithm is a careful modification of standard interior point methods for solving the geometric median problem. We provide a long step interior point method tailored to the geometric median problem for which we can implement every iteration in nearly linear time. While our analysis starts with a simple $O((nd)^{O(1)} \log \frac{1}{\epsilon})$ time interior point method and shows how to improve it, our final algorithm is quite non-standard from the perspective of interior point literature. Our result is one of very few cases we are aware of that outperforms traditional interior point theory [Mad13, LS14] and the only one we are aware of using interior point methods to obtain a nearly linear time algorithm for a canonical optimization problem that traditionally requires superlinear time. We hope our work leads to further improvements in this line of research.

Our $O(d\epsilon^{-2})$ algorithm is a relatively straightforward application of sampling techniques and stochastic subgradient descent. Some additional insight is required simply to provide a rigorous analysis of the robustness of the geometric median and use this to streamline our application of stochastic subgradient descent. We provide its proof in Section 2.5. The bulk of the work in this chapter is focused on developing our $O(nd \log^3 \frac{n}{\epsilon})$ time algorithm which we believe uses a set of techniques of independent interest.

[1]If z is the total number of nonzero entries in the coordinates of the $a^{(i)}$ then a careful analysis of our algorithm improves our running time to $O(z \log^3 \frac{n}{\epsilon})$.

2.1.1 Previous Work

The geometric median problem was first formulated for the case of three points in the early 1600s by Pierre de Fermat [KV97, DKSW02]. A simple elegant ruler and compass construction was given in the same century by Evangelista Torricelli. Such a construction does not generalize when a larger number of points is considered: Bajaj has shown the even for five points, the geometric median is not expressible by radicals over the rationals [Baj88]. Hence, the $(1+\epsilon)$-approximate problem has been studied for larger values of n.

Many authors have proposed algorithms with runtime polynomial in n, d and $1/\epsilon$. The most cited and used algorithm is Weiszfeld's 1937 algorithm [Wei37]. Unfortunately Weiszfeld's algorithm may not converge and if it does it may do so very slowly. There have been many proposed modifications to Weiszfeld's algorithm [CK81, PE08, Ost78, BY82, VZ00, Kuh73] that generally give non-asymptotic runtime guarantees. In light of more modern multiplicative weights methods his algorithm can be viewed as a re-weighted least squares Chin et al. [CMMP13] considered the more general L_2 embedding problem: placing the vertices of a graph into $\mathbb{R}^d$, where some of the vertices have fixed positions while the remaining vertices are allowed to float, with the objective of minimizing the sum of the Euclidean edge lengths. Using the multiplicative weights method, they obtained a run time of $O(d \cdot n^{4/3}\epsilon^{-8/3})$ for a broad class of problems, including the geometric median problem.[2]

Many authors consider problems that generalize the Fermat-Weber problem, and obtain algorithms for finding the geometric median as a specialization. For example, Badoiu et al. give an approximate k-median algorithm by sub-sampling with the runtime for $k=1$ of $\tilde{O}(d \cdot \exp(O(\epsilon^{-4})))$ [BHI02]. Parrilo and Sturmfels demonstrated that the problem can be reduced to semidefinite programming, thus obtaining a runtime of $\tilde{O}(\text{poly}(n,d)\log\epsilon^{-1})$ [PS01]. Furthermore, Bose et al. gave a linear time algorithm for fixed d and ϵ^{-1}, based on low-dimensional data structures [BMM03] and it has been show how to obtain running times of $\tilde{O}(nd + \text{poly}(d,\epsilon^{-1}))$ for this problem and a more general class of problems.[HPK05, FL11].

An approach very related to ours was studied by Xue and Ye [XY97]. They give an interior point method with barrier analysis that runs in time $\tilde{O}((d^3 + d^2 n)\sqrt{n}\log\epsilon^{-1})$.

2.1.2 Overview of $O(nd\log^3\frac{n}{\epsilon})$ Time Algorithm

Interior Point Primer

Our algorithm is broadly inspired by interior point methods, a broad class of methods for efficiently solving convex optimization problems [Ye11, NN94]. Given an instance of the geometric median problem we first put the problem in a more natural form for applying interior point methods. Rather than writing the problem as minimizing a convex function over $\mathbb{R}^d$

$$\min_{x\in\mathbb{R}^d} f(x) \quad \text{where} \quad f(x) \overset{\text{def}}{=} \sum_{i\in[n]} \left\|x - a^{(i)}\right\|_2 \tag{2.1}$$

[2]The result of [CMMP13] was stated in more general terms than given here. However, it easy to formulate the geometric median problem in their model.

Year	**Authors**	**Runtime**	**Comments**
1659	Torricelli [Viv59]	-	Assuming $n = 3$
1937	Weiszfeld [Wei37]	-	Does not always converge
1990	Chandrasekaran and Tamir[CT89]	$\tilde{O}(n \cdot \text{poly}(d) \log \epsilon^{-1})$	Ellipsoid method
1997	Xue and Ye [XY97]	$\tilde{O}((d^3 + d^2 n) \sqrt{n} \log \epsilon^{-1})$	Interior point with barrier method
2000	Indyk [ID00]	$\tilde{O}(dn \cdot \epsilon^{-2})$	Optimizes only over x in the input
2001	Parrilo and Sturmfels [PS01]	$\tilde{O}(\text{poly}(n, d) \log \epsilon^{-1})$	Reduction to SDP
2002	Badoiu et al. [BHI02]	$\tilde{O}(d \cdot \exp(O(\epsilon^{-4})))$	Sampling
2003	Bose et al. [BMM03]	$\tilde{O}(n)$	Assuming $d, \epsilon^{-1} = O(1)$
2005	Har-Peled and Kushal [HPK05]	$\tilde{O}(n + \text{poly}(\epsilon^{-1}))$	Assuming $d = O(1)$
2011	Feldman and Langberg [FL11]	$\tilde{O}(nd + \text{poly}(d, \epsilon^{-1}))$	Coreset
2013	Chin et al. [CMMP13]	$\tilde{O}(dn^{4/3} \cdot \epsilon^{-8/3})$	Multiplicative weights
-	**This work**	$O(nd \log^3(n/\epsilon))$	Interior point with custom analysis
-	**This work**	$O(d\epsilon^{-2})$	Stochastic gradient descent

Table 2.1: Selected Previous Results.

we instead write the problem as minimizing a linear function over a larger convex space:

$$\min_{\{\alpha,x\} \in S} 1^\top \alpha \quad \text{where } S = \left\{ \alpha \in \mathbb{R}^n, x \in \mathbb{R}^d \;\middle|\; \left\| x - a^{(i)} \right\|_2 \leq \alpha_i \text{ for all } \; i \in [n] \right\} . \tag{2.2}$$

Clearly, these problems are the same, as at optimality $\alpha_i = \left\| x - a^{(i)} \right\|_2$.

To solve problems of the form (2.2) interior point methods relax the constraint $\{\alpha, x\} \in S$ through the introduction of a *barrier function*. In particular they assume that there is a real valued function p such that as $\{\alpha, x\}$ moves towards the boundary of S the value of p goes to infinity. A popular class of interior point methods, known as *path following methods [Ren88, Gon92]*, consider relaxations of (2.2) of the form $\min_{\{\alpha,x\} \in \mathbb{R}^n \times \mathbb{R}^d} t \cdot 1^\top \alpha + p(\alpha, x)$. The minimizers of this function form a path, known as the central path, parameterized by t. The methods then use variants of Newton's method to follow the path until t is large enough that a high quality approximate solution is obtained. The number of iterations of these methods are then typically governed by a property of p known as its self concordance ν. Given a ν-self concordant barrier, typically interior point methods require $O(\sqrt{\nu} \log \frac{1}{\epsilon})$ iterations to compute a $(1 + \epsilon)$-approximate solution.

For our particular convex set, the construction of our barrier function is particularly simple, we consider each constraint $\left\| x - a^{(i)} \right\|_2 \leq \alpha_i$ individually. In particular, it is known that the function $p^{(i)}(\alpha, x) = -\ln \alpha_i - \ln \left(\alpha_i^2 - \left\| x - a^{(i)} \right\|_2 \right)$ is a $O(1)$ self-concordant barrier function for the set $S^{(i)} = \left\{ x \in \mathbb{R}^d, \alpha \in \mathbb{R}^n \mid \left\| x - a^{(i)} \right\|_2 \leq \alpha_i \right\}$[Nes03]. Since $\cap_{i \in [n]} S^{(i)} = S$ we can use the barrier $\sum_{i \in [n]} p^{(i)}(\alpha, x)$ for $p(\alpha, x)$ and standard self-concordance theory shows that this is an $O(n)$ self concordant barrier for S. Consequently, this easily yields an interior point method for solving the geometric median problem in $O((nd)^{O(1)} \log \frac{1}{\epsilon})$ time.

Difficulties

Unfortunately obtaining a nearly linear time algorithm for geometric median using interior point methods as presented poses numerous difficulties. Particularly troubling is the number of iterations required by standard interior point algorithms. The approach outlined in the previous section produced an $O(n)$-self concordant barrier and even if we use more advanced self concordance machinery, i.e. the universal barrier [NN94], the best known self concordance of any barrier for the convex set $\sum_{i\in[n]}\left\|x-a^{(i)}\right\|_2 \leq c$ is $O(d)$. An interesting open question still left open by our work is to determine what is the minimal self concordance of a barrier for this set.

Consequently, even if we could implement every iteration of an interior point scheme in nearly linear time it is unclear whether one should hope for a nearly linear time interior point algorithm for the geometric median. While there are instances of outperforming standard self-concordance analysis [Mad13, LS14], these instances are few, complex, and to varying degrees specialized to the problems they solve. Moreover, we are unaware of any interior point scheme providing a provable nearly linear time for a general nontrivial convex optimization problem.

Beyond Standard Interior Point

Despite these difficulties we do obtain a nearly linear time interior point based algorithm that only requires $O(\log\frac{n}{\epsilon})$ iterations, i.e. increases to the path parameter. After choosing the natural penalty functions $p^{(i)}$ described above, we optimize in closed form over the α_i to obtain the following penalized objective function:[3]

$$\min_x f_t(x) \quad \text{where} \quad f_t(x) = \sum_{i\in[n]} \sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2} - \ln\left[1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}\right]$$

We then approximately minimize $f_t(x)$ for increasing t. We let $x_t \stackrel{\text{def}}{=} \text{argmin}_{x\in\mathbb{R}^d} f_t(x)$ for $x \geq 0$, and thinking of $\{x_t : t \geq 0\}$ as a continuous curve known as the central path, we show how to approximately follow this path. As $\lim_{t\to\infty} x_t = x^*$ this approach yields a $(1+\epsilon)$-approximation.

So far our analysis is standard and interior point theory yields an $\Omega(\sqrt{n})$ iteration interior point scheme. To overcome this we take a more detailed look at x_t. We note that for any t if there is any rapid change in x_t it must occur in the direction of the smallest eigenvector of $\bigtriangledown^2 f_t(x)$, denoted v_t, what we henceforth may refer to as the *bad direction* at x_t. Furthermore, we show that this bad direction is *stable* in the sense that for all directions $d \perp v_t$ it is the case that $d^\top(x_t - x_{t'})$ is small for $t' \leq ct$ for a small constant c.

In fact, we show that this movement over such a *long step,* i.e. constant multiplicative increase in t, in the directions orthogonal to the bad direction is small enough that for any movement around a ball of this size the Hessian of f_t only changes by a small multiplicative constant. In short, starting at x_t there exists a point y obtained just by moving from x_t in the bad direction, such that y is close

[3]It is unclear how to extend our proof for the simpler function: $\sum_{i\in[n]}\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}$.

enough to $x_{t'}$ that standard first order method will converge quickly to $x_{t'}$! Thus, we might hope to find such a y, quickly converge to $x_{t'}$ and repeat. If we increase t by a multiplicative constant in every such iterations, standard interior point theory suggests that $O(\log \frac{n}{\epsilon})$ iterations suffices.

Building an Algorithm

To turn the structural result in the previous section into a fast algorithm there are several further issues we need to address. We need to

(1) Show how to find the point along the bad direction that is close to $x_{t'}$.

(2) Show how to solve linear systems in the Hessian to actually converge quickly to $x_{t'}$.

(3) Show how to find the bad direction.

(4) Bound the accuracy required by these computations.

Deferring (1) for the moment, our solutions to the rest are relatively straightforward. Careful inspection of the Hessian of f_t reveals that it is well approximated by a multiple of the identity matrix minus a rank 1 matrix. Consequently, using explicit formulas for the inverse of of matrix under rank 1 updates, i.e. the Sherman-Morrison formula, we can solve such systems in nearly linear time thereby addressing (2). For (3), we show that the well known power method carefully applied to the Hessian yields the bad direction if it exists. Finally, for (4) we show that a constant approximate geometric median is near enough to the central path for $t = \Theta(\frac{1}{f(x_*)})$ and that it suffices to compute a central path point at $t = O(\frac{n}{f(x_*)\epsilon})$ to compute a $1+\epsilon$-geometric median. Moreover, for these values of t, the precision needed in other operations is clear.

The more difficult operation is (1). Given x_t and the bad direction exactly, it is still not clear how to find the point along the bad direction line from x_t that is close to $x_{t'}$. Just performing binary search on the objective function a priori might not yield such a point due to discrepancies between a ball in Euclidean norm and a ball in hessian norm and the size of the distance from the optimal point in euclidean norm. To overcome this issue we still line search on the bad direction, however rather than simply using $f(x_t + \alpha \cdot v_t)$ as the objective function to line search on, we use the function $g(\alpha) = \min_{\left\|x - x_t - \alpha \cdot v_t\right\|_2 \leq c} f(x)$ for some constant c, that is given an α we move α in the bad direction and take the best objective function value in a ball around that point. For appropriate choice of c the minimizers of α will include the optimal point we are looking for. Moreover, we can show that g is convex and that it suffices to perform the minimization approximately.

Putting these pieces together yields our result. We perform $O(\log \frac{n}{\epsilon})$ iterations of interior point (i.e. increasing t), where in each iteration we spend $O(nd \log \frac{n}{\epsilon})$ time to compute a high quality approximation to the bad direction, and then we perform $O(\log \frac{n}{\epsilon})$ approximate evaluations on $g(\alpha)$ to binary search on the bad direction line, and then to approximately evaluate g we perform gradient descent in approximate Hessian norm to high precision which again takes $O(nd \log \frac{n}{\epsilon})$ time. Altogether this yields a $O(nd \log^3 \frac{n}{\epsilon})$ time algorithm to compute a $1+\epsilon$ geometric median. Here we made minimal effort to improve the log factors and plan to investigate this further in future work.

2.1.3 Overview of $O(d\epsilon^{-2})$ Time Algorithm

In addition to providing a nearly linear time algorithm we provide a stand alone result on quickly computing a crude $(1+\epsilon)$-approximate geometric median in Section 2.5. In particular, given an oracle for sampling a random $a^{(i)}$ we provide an $O(d\epsilon^{-2})$, i.e. sublinear, time algorithm that computes such an approximate median. Our algorithm for this result is fairly straightforward. First, we show that random sampling can be used to obtain some constant approximate information about the optimal point in constant time. In particular we show how this can be used to deduce an Euclidean ball which contains the optimal point. Second, we perform stochastic subgradient descent within this ball to achieve our desired result.

2.1.4 Chapter Organization

The rest of the chapter is structured as follows. After covering preliminaries in Section 2.2, in Section 2.3 we provide various results about the central path that we use to derive our nearly linear time algorithm. In Section 2.4 we then provide our nearly linear time algorithm. All the proofs and supporting lemmas for these sections are deferred to Appendix 2.A and Appendix 2.B. In Section 2.5 we provide our $O(d/\epsilon^2)$ algorithm. In Section 2.6 we show how to extend our results to Weber's problem, i.e. weighted geometric median.

In Appendix 2.C we provide the derivation of our penalized objective function and in Appendix 2.D we provide general technical machinery we use throughout.

2.2 Notation

2.2.1 General Notation

We use boldface letters to denote matrices. For a symmetric positive semidefinite matrix (PSD), $\mathbf{A}$, we let $\lambda_1(\mathbf{A}) \geq ... \geq \lambda_n(\mathbf{A}) \geq 0$ denote the eigenvalues of $\mathbf{A}$ and let $v_1(\mathbf{A}), ..., v_n(\mathbf{A})$ denote corresponding eigenvectors. We let $\left\|x\right\|_{\mathbf{A}} \stackrel{\text{def}}{=} \sqrt{x^\top \mathbf{A} x}$ and for PSD matrices we use $\mathbf{A} \preceq \mathbf{B}$ and $\mathbf{A} \succeq \mathbf{B}$ to denote the conditions that $x^\top \mathbf{A} x \leq x^\top \mathbf{B} x$ for all x and $x^\top \mathbf{A} x \geq x^\top \mathbf{B} x$ for all x respectively.

2.2.2 Problem Notation

The central problem of this chapter is as follows: we are given points $a^{(1)}, ..., a^{(n)} \in \mathbb{R}^d$ and we wish to compute a geometric median, i.e. $x_* \in \operatorname{argmin}_x f(x)$ where $f(x) = \sum_{i\in[n]} \left\|a^{(i)} - x\right\|_2$. We call a point $x \in \mathbb{R}^d$ an $(1+\epsilon)$-approximate geometric median if $f(x) \leq (1+\epsilon) f(x_*)$.

2.2.3 Penalized Objective Notation

To solve this problem we relax the objective function f and instead consider the following family of penalized objective functions parameterized by $t > 0$

$$\min_x f_t(x) \quad \text{where} \quad f_t(x) = \sum_{i\in[n]} \sqrt{1 + t^2 \left\|x - a^{(i)}\right\|_2^2} - \ln\left[1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}\right].$$

This penalized objective function arises naturally in considering a standard interior point formulation of the geometric median problem. (See Section 2.C for the derivation.) For all *path parameters* $t > 0$, we let $x_t \stackrel{\text{def}}{=} \text{argmin}_x f_t(x)$. Our primary goal is to obtain good approximations to elements of the *central path* $\{x_t : t > 0\}$ for increasing values of t.

We let $g_t^{(i)}(x) \stackrel{\text{def}}{=} \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}$ and $f_t^{(i)}(x) \stackrel{\text{def}}{=} g_t^{(i)}(x) - \ln(1 + g_t^{(i)}(x))$ so $f_t(x) = \sum_{i\in[n]} f_t^{(i)}(x)$. We refer to the quantity $w_t(x) \stackrel{\text{def}}{=} \sum_{i\in[n]} \frac{1}{1+g_t^{(i)}(x)}$ as weight as it is a natural measure of total contribution of the $a^{(i)}$ to the Hessian $\bigtriangledown^2 f_t(x)$. We let

$$\bar{g}_t(x) \stackrel{\text{def}}{=} w_t(x)\left[\sum_{i\in[n]} \frac{1}{(1 + g_t^{(i)}(x_t))g_t^{(i)}(x_t)}\right]^{-1}$$

denote a natural term that helps upper bound the rate of change of the central path. Furthermore we use $u^{(i)}(x) \stackrel{\text{def}}{=} x - a^{(i)}/\left\|x - a^{(i)}\right\|_2$ for $\left\|x - a^{(i)}\right\|_2 \neq 0$ and $u^{(i)}(x) = 0$ otherwise to denote the unit vector corresponding to $x - a^{(i)}$. Finally we let $\mu_t(x) \stackrel{\text{def}}{=} \lambda_d(\bigtriangledown^2 f_t(x))$, the minimum eigenvalue of $\bigtriangledown^2 f_t(x)$, and let $v_t(x)$ denote a corresponding eigenvector. To simplify notation we often drop the (x) in these definitions when $x = x_t$ and t is clear from context.

2.3 Properties of the Central Path

Here provide various facts regarding the penalized objective function and the central path. While we use the lemmas in this section throughout the chapter, the main contribution of this section is Lemma 2.3.5 in Section 2.3.3. There we prove that with the exception of a single direction, the change in the central path is small over a constant multiplicative change in the path parameter. In addition, we show that our penalized objective function is stable under changes in a $O(\frac{1}{t})$ Euclidean ball (Section 2.3.1), and we bound the change in the Hessian over the central path (Section 2.3.2). Furthermore, in Section 2.3.4, we relate $f(x_t)$ to $f(x^*)$.

2.3.1 How Much Does the Hessian Change in General?

Here, we show that the Hessian of the penalized objective function is stable under changes in a $O(\frac{1}{t})$ sized Euclidean ball. This shows that if we have a point which is close to a central path point in the Euclidean norm, then we can use Newton's method to find the latter point.

Lemma 2.3.1 *Suppose that $\left\|x - y\right\|_2 \leq \frac{\epsilon}{t}$ with $\epsilon \leq \frac{1}{20}$. Then, we have*

$$(1 - 6\epsilon^{2/3}) \bigtriangledown^2 f_t(x) \preceq \bigtriangledown^2 f_t(y) \preceq (1 + 6\epsilon^{2/3}) \bigtriangledown^2 f_t(x).$$

2.3.2 How Much Does the Hessian Change Along the Path?

Here we bound how much the Hessian of the penalized objective function can change along the central path. First we provide the following lemma bound several aspects of the penalized objective function and proving that the weight, w_t, only changes by a small amount multiplicatively given small multiplicative changes in the path parameter, t.

Lemma 2.3.2 *For all $t \geq 0$ and $i \in [n]$, the following inequalities hold:*

$$\left\|\frac{d}{dt}x_t\right\|_2 \leq \frac{1}{t^2}\bar{g}_t(x_t) \ , \quad \left|\frac{d}{dt}g_t^{(i)}(x_t)\right| \leq \frac{1}{t}\left(g_t^{(i)}(x_t) + \bar{g}_t\right) \ , \text{ and } \quad \left|\frac{d}{dt}w_t\right| \leq \frac{2}{t}w_t.$$

Consequently, for all $t' \geq t$ we have that $\left(\frac{t}{t'}\right)^2 w_t \leq w_{t'} \leq \left(\frac{t'}{t}\right)^2 w_t$.

Next we use this lemma to bound the change in the Hessian with respect to t.

Lemma 2.3.3 *For all $t \geq 0$, we have*

$$-12 \cdot t \cdot w_t\mathbf{I} \preceq \frac{d}{dt}\left[\bigtriangledown^2 f_t(x_t)\right] \preceq 12 \cdot t \cdot w_t\mathbf{I} \tag{2.3}$$

and therefore for all $\beta \in [0, \frac{1}{8}]$

$$\bigtriangledown^2 f(x_t) - 15\beta t^2 w_t\mathbf{I} \preceq \bigtriangledown^2 f(x_{t(1+\beta)}) \preceq \bigtriangledown^2 f(x_t) + 15\beta t^2 w_t\mathbf{I}\,. \tag{2.4}$$

2.3.3 Where is the Next Optimal Point?

Here we prove our main result of this section. We prove that over a long step the central path moves very little in directions orthogonal to the smallest eigenvector of the Hessian. We begin by noting the Hessian is approximately a scaled identity minus a rank 1 matrix.

Lemma 2.3.4 *For all t, we have $\frac{1}{2}\left[t^2 \cdot w_t\mathbf{I} - (t^2 \cdot w_t - \mu_t)v_tv_t^\top\right] \preceq \bigtriangledown^2 f_t(x_t) \preceq t^2 \cdot w_t\mathbf{I} - (t^2 \cdot w_t - \mu_t)v_tv_t^\top$.*

Using this and the lemmas of the previous section we bound the amount x_t can move in every direction far from v_t.

Lemma 2.3.5 (The Central Path is Almost Straight) *For all $t \geq 0$, $\beta \in [0, \frac{1}{600}]$, and any unit vector y with $|\langle y, v_t\rangle| \leq \frac{1}{t^2 \cdot \kappa}$ where $\kappa = \max_{\delta\in[t,(1+\beta)t]} \frac{w_\delta}{\mu_\delta}$, we have $y^\top(x_{(1+\beta)t} - x_t) \leq \frac{6\beta}{t}$.*

2.3.4 Where is the End?

In this section, we bound the quality of the central path with respect to the geometric median objective. In particular, we show that if we can solve the problem for some $t = \frac{2n}{\epsilon f(x^*)}$ then we can obtain an $(1+\epsilon)$-approximate solution. As we will ultimately derive an algorithm that starting from initial $t = 1/O(f(x^*))$ and doubles t in every iteration, this will yield a $O(\log \frac{n}{\epsilon})$ iteration algorithm to obtain an $(1+\epsilon)$-approximate solution.

Lemma 2.3.6 *For all $t > \frac{1}{400 f(x^*)}$, we have $f(x_t) - f(x^*) \leq 100n\sqrt{\frac{f(x^*)}{t}}$.*

2.4 Nearly Linear Time Geometric Median

Here we show how to use the results from the previous section to obtain a nearly linear time algorithm for computing the geometric median. Our algorithm follows a simple structure (See Algorithm 1). First we use our result from Section 2.5 to compute an $(1+\epsilon_0)$-approximate median, denoted $x^{(1)}$. Then for a number of iterations we repeatedly move closer to x_t for some path parameter t, compute the minimum eigenvector of the Hessian, and line search in that direction to find an approximation to a point further along the central path. Ultimately, this yields a precise enough approximation to a point along the central path with large enough t that it is a high quality approximation to the geometric median.

Algorithm 1: `AccurateMedian`(ϵ)

```
// Compute a 2-approximate geometric median and use it to
   center
$x^{(0)} :=$ ApproximateMedian$(2)$
Let $\widetilde{f}_* := f(x^{(0)})$, $t_i = \frac{1}{400\widetilde{f}_*}(1+\frac{1}{600})^{i-1}$
$x^{(1)} =$ LineSearch$(x^{(0)}, t_1, t_1, 0, \epsilon_c)$ with $\epsilon_c = \frac{1}{10^{15} n^3 t_1^9 \cdot \widetilde{f}_*^3}$.

// Iteratively improve quality of approximation
for $i \in [1, 1000\log\left(\frac{3000n}{\epsilon}\right)]$ do
    // Compute $\epsilon_v$-approximate minimum eigenvalue and
       eigenvector of $\bigtriangledown^2 f_{t_i}(x^{(i)})$
    $(\lambda^{(i)}, u^{(i)}) =$ ApproxMinEig$(x^{(i)}, t_i, \epsilon_v)$ with $\epsilon_v = \frac{1}{10^8 n^2 t_i^2 \cdot \widetilde{f}_*^2}$.

    // Line search to find $x^{(i+1)}$ such that $\left\|x^{(i+1)} - x_{t_{i+1}}\right\|_2 \leq \frac{\epsilon_c}{t_{i+1}}$
    $x^{(i+1)} =$ LineSearch$(x^{(i)}, t_i, t_{i+1}, u^{(i)}, \epsilon_c)$ with $\epsilon_c = \frac{1}{10^{15} n^3 t_i^3 \cdot \widetilde{f}_*^3}$.
end
Output: $\epsilon$-approximate geometric median $x^{(k)}$
```

We split the remainder of the algorithm specification and its analysis into several parts. First in Section 2.4.1 we show how to compute an approximate minimum eigenvector and eigenvalue of the Hessian of the penalized objective function. Then in Section 2.4.2 we show how to use this eigenvector to line search for the next central path point. Finally, in Section 2.4.3 we put these results together to obtain our nearly linear time algorithm. Throughout this section we will want an upper bound to $f(x_*)$ and will use $\widetilde{f}_*$ as this upper bound.

2.4.1 Eigenvector Computation and Hessian Approximation

Here we show how to compute the minimum eigenvector of $\bigtriangledown^2 f_t(x)$ and thereby obtain a concise approximation to $\bigtriangledown^2 f_t(x)$. Our main algorithmic tool is the well known power method and the fact that it converges quickly on a matrix with a large eigenvalue gap. We provide and analyze this method for completeness in Section 2.B.1. Using this tool we estimate the top eigenvector as follows.

Algorithm 2: `ApproxMinEig`(x, t, ϵ)

```
Input: Point $x \in \mathbb{R}^d$, path parameter $t$, and target accuracy $\epsilon$.
Let $\mathbf{A} = \sum_{i \in [n]} \frac{t^4 (x - a^{(i)})(x - a^{(i)})^\top}{(1 + g_t^{(i)}(y))^2 g_t^{(i)}(y)}$
Let $u :=$ PowerMethod($\mathbf{A}, \Theta(\log(\frac{d}{\epsilon}))$)
Let $\lambda = u^\top \bigtriangledown^2 f_t(x) u$
Output: $(\lambda, u)$
```

Lemma 2.4.1 (Hessian Eigenvector Computation and Approximation) *For any $x \in \mathbb{R}^d$, $t > 0$, and $\epsilon \in (0, \frac{1}{4})$, The algorithm* `ApproxMinEig`$(x, t, \epsilon)$ *outputs (λ, u) in $O(nd \log \frac{d}{\epsilon})$ time with high probability in d such that $\langle v_t(x), u \rangle^2 \geq 1 - \epsilon$ if $\mu_t(x) \leq \frac{1}{4} t^2 w_t(x)$. Furthermore, if $\epsilon \leq \frac{\mu_t(x)}{4t^2 \cdot w_t(x)} \leq \frac{1}{10^9 n^2 t^2 \cdot \widetilde{f}_*^2}$ and $\mathbf{Q} \stackrel{\text{def}}{=} t^2 \cdot w_t(x) - (t^2 \cdot w_t(x) - \lambda) u u^\top$ then $\frac{1}{4}\mathbf{Q} \preceq \bigtriangledown^2 f_t(x) \preceq 4\mathbf{Q}$.*

Furthermore, we show that the $v^{(i)}$ computed by this algorithm is sufficiently close to the bad direction. Using Lemma 2.D.4, a minor technical lemma regarding the transitivity of large inner products, yields:

Lemma 2.4.2 *Let $u =$* `ApproxMinEig`(x, t, ϵ_v) *for some x such that $\left\|x - x_t\right\|_2 \leq \frac{\epsilon_c}{t}$ for $\epsilon_c \leq \frac{1}{10^6}$. Suppose that $\mu_t \leq \frac{1}{4} t^2 \cdot w_t$. Then, for all unit vectors $y \perp u$, we have $\langle y, v_t \rangle^2 \leq \max\{500 \epsilon_c^{2/3}, 10 \epsilon_v\}$.*

Note that the lemma above assumed μ_t is small. For the case μ_t is large, we show that the next central path point is close to the current point and hence we do not need to know the bad direction.

Lemma 2.4.3 *Suppose that $\mu_t \geq \frac{1}{4} t^2 \cdot w_t$. Then, we have $\left\|x_s - x_t\right\|_2 \leq \frac{1}{100t}$ for all $s \in [1, 1.001t]$.*

2.4.2 Line Searching

Here we show how to line search along the bad direction to find the next point on the central path. Unfortunately, it is not clear if you can binary search on the objective function directly. It might be the case that if we searched over α to minimize $f_{t_{i+1}}(y^{(i)}+\alpha v^{(i)})$ we might obtain a point far away from x_{t+1} and therefore be unable to move towards x_{t+1} efficiently.

To overcome this difficulty, we use the fact that over the region $\left\|x-y\right\|_2 = O(\frac{1}{t})$ the Hessian changes by at most a constant and therefore we can minimize $f_t(x)$ over this region extremely quickly. Therefore, we instead line search on the following function

$$g_{t,y,v}(\alpha) \stackrel{\text{def}}{=} \min_{\left\|x-(y+\alpha v)\right\|_2 \leq \frac{1}{100t}} f_t(x) \tag{2.5}$$

and use that we can evaluate $g_{t,y,v}(\alpha)$ approximately by using an appropriate centering procedure. We can show (See Lemma 2.D.6) that $g_{t,y,v}(\alpha)$ is convex and therefore we can minimize it efficiently just by doing an appropriate binary search. By finding the approximately minimizing α and outputting the corresponding approximately minimizing x, we can obtain $x^{(i+1)}$ that is close enough to $x_{t_{i+1}}$. For notational convenience, we simply write $g(\alpha)$ if t, y, v is clear from the context.

First, we show how we can locally center and provide error analysis for that algorithm.

Algorithm 3: `LocalCenter`(y, t, ϵ)

```
Input: Point $y \in \mathbb{R}^d$, path parameter $t$, target accuracy $\epsilon$.
Let $(\lambda, v) :=$ ApproxMinEig$(x, t, 10^{-9}n^{-2}t^{-2} \cdot \widetilde{f}_*^{-2})$.
Let $\mathbf{Q} = t^2 \cdot w_t(y)\mathbf{I} - (t^2 \cdot w_t(y) - \lambda)\, vv^\top$
Let $x^{(0)} = y$
for $i = 1, ..., k = 64 \log \frac{1}{\epsilon}$ do
    Let $x^{(i)} = \min_{\left\|x-y\right\|_2 \leq \frac{1}{100t}} f(x^{(i-1)}) + \langle \bigtriangledown f_t(x^{(i-1)}), x - x^{(i-1)} \rangle + 4\left\|x - x^{(i-1)}\right\|_{\mathbf{Q}}^2$.
end
Output: $x^{(k)}$
```

Lemma 2.4.4 *Given some $y \in \mathbb{R}^d$, $t > 0$ and $\epsilon > 0$. In $O(nd \log(\frac{nt\cdot\widetilde{f}_*}{\epsilon}))$ time with high probability,* `LocalCenter`(y, t, ϵ) *computes $x^{(k)}$ such that*

$$f_t(x^{(k)}) - \min_{\left\|x-y\right\|_2 \leq \frac{1}{100t}} f_t(x) \leq \epsilon \left(f_t(y) - \min_{\left\|x-y\right\|_2 \leq \frac{1}{100t}} f_t(x) \right).$$

Using this local centering algorithm as well as a general result for minimizing one dimensional convex functions using a noisy oracle (See Section 2.D.3) we obtain our line search algorithm.

Lemma 2.4.5 *Given x such that $\left\|x - x_t\right\|_2 \leq \frac{\epsilon_c}{t}$ with $t \geq \frac{1}{400 f(x^*)}$ and $\epsilon_c \leq \frac{1}{10^{15} n^3 t^3 \cdot \widetilde{f}_*^3}$. Let u = `ApproxMinEig`(x, t, ϵ_v) with $\epsilon_v \leq \frac{1}{10^8 n^2 t^2 \cdot \widetilde{f}_*^2}$. Then, in $O(nd \log^2(\frac{nt\cdot\widetilde{f}_*}{\varepsilon}))$ time,* `LineSearch`(x, t, t', u, ϵ) *output y such that $\left\|y - x_{t'}\right\|_2 \leq \frac{\epsilon}{t'}$.*

```
Algorithm 4: LineSearch$(x, t, t', u, \epsilon)$
    Input: Point $x \in \mathbb{R}^d$, current path parameter $t$, next path parameter $t'$, bad direction $u$, target accuracy $\epsilon$
    Let $\epsilon_O = \frac{\epsilon^2}{10^{10} t^3 n^3 \cdot \widetilde{f}_*^3}$, $\ell = -12\widetilde{f}_*$, $u = 12\widetilde{f}_*$.
    Define the oracle $q : \mathbb{R} \to \mathbb{R}$ by $q(\alpha) = f_{t'}(\texttt{LocalCenter}(x + \alpha u, t', \epsilon_O))$
    Let $\alpha' = \texttt{OneDimMinimizer}(\ell, u, \epsilon_O, q, tn)$
    Output: $x' = \textbf{LocalCenter}(x + \alpha u, t', \epsilon_O)$
```

We also provide the following lemma useful for finding the first center.

Lemma 2.4.6 *Given x such that $\left\|x - x_t\right\|_2 \leq \frac{1}{100t}$ with $t \geq \frac{1}{400 f(x^*)}$. Then, in $O(nd \log^2(\frac{nt \cdot \widetilde{f}_*}{\varepsilon}))$ time,* `LineSearch`(x, t, t, u, ϵ) *output y such that $\left\|y - x_t\right\|_2 \leq \frac{\epsilon}{t}$ for any vector u.*

2.4.3 Putting It All Together

Here we show how to put together the results of the previous sections to prove our main theorem.

Theorem 2.4.7 *In $O(nd \log^3(\frac{n}{\epsilon}))$ time, Algorithm 1 outputs an $(1 + \epsilon)$-approximate geometric median with constant probability.*

2.5 Pseudo Polynomial Time Algorithm

Here we provide a self-contained result on computing a $1 + \epsilon$ approximate geometric median in $O(d\epsilon^{-2})$ time. Note that it is impossible to achieve such approximation for the mean, $\min_{x \in \mathbb{R}^d} \sum_{i \in [n]} \left\|x - a^{(i)}\right\|_2^2$, because the mean can be changed arbitrarily by changing only 1 point. However, [LR91] showed that the geometric median is far more stable. In Section 2.5.1, we show how this stability property allows us to get an constant approximate in $O(d)$ time. In Section 2.5.2, we show how to use stochastic subgradient descent to then improve the accuracy.

2.5.1 A Constant Approximation of Geometric Median

We first prove that the geometric median is stable even if we are allowed to modify up to half of the points. The following lemma is a strengthening of the robustness result in [LR91].

Lemma 2.5.1 *Let x^* be a geometric median of $\{a^{(i)}\}_{i \in [n]}$ and let $S \subseteq [n]$ with $\left\|S\right\| < \frac{n}{2}$. For all x*

$$\left\|x^* - x\right\|_2 \leq \left(\frac{2n - 2|S|}{n - 2|S|}\right) \max_{i \notin S} \left\|a^{(i)} - x\right\|_2 .$$

Proof For notational convenience let $r = \left\|x^* - x\right\|_2$ and let $M = \max_{i \notin S} \left\|a^{(i)} - x\right\|_2$.

For all $i \notin S$, we have that $\left\|x - a^{(i)}\right\|_2 \leq M$, hence, we have

$$\begin{aligned}\left\|x^* - a^{(i)}\right\|_2 &\geq r - \left\|x - a^{(i)}\right\|_2 \\ &\geq r - 2M + \left\|x - a^{(i)}\right\|_2 .\end{aligned}$$

Furthermore, by triangle inequality for all $i \in S$, we have

$$\left\|x^* - a^{(i)}\right\|_2 \geq \left\|x - a^{(i)}\right\|_2 - r .$$

Hence, we have that

$$\sum_{i\in[n]} \left\|x^* - a^{(i)}\right\|_2 \geq \sum_{i\in[n]} \left\|x - a^{(i)}\right\|_2 + (n - |S|)(r - 2M) - |S|r .$$

Since x^* is a minimizer of $\sum_i \left\|x^* - a^{(i)}\right\|_2$, we have that

$$(n - |S|)(r - 2M) - |S|r \leq 0.$$

Hence, we have

$$\left\|x^* - x\right\|_2 = r \leq \frac{2n - 2|S|}{n - 2|S|}M.$$

■

Now, we use Lemma 2.5.1 to show that the algorithm `CrudeApproximate` outputs a constant approximation of the geometric median with high probability.

Algorithm 5: $\texttt{CrudeApproximate}_K$

```
Input: $a^{(1)}, a^{(2)}, \cdots , a^{(n)} \in \mathbb{R}^d$.
Sample two independent random subset of $[n]$ of size $K$. Call them $S_1$ and $S_2$.
Let $i^* \in \mathrm{argmin}_{i\in S_2}\alpha_i$ where $\alpha_i$ is the 65 percentile of the numbers $\{\left\|a^{(i)} - a^{(j)}\right\|_2\}_{j\in S_1}$.
Output: Output $a^{(i^*)}$ and $\alpha_{i^*}$.
```

Lemma 2.5.2 *Let x^* be a geometric median of $\{a^{(i)}\}_{i\in[n]}$ and $(\widetilde{x}, \lambda)$ be the output of* $\texttt{CrudeApproximate}_K$. *We define $d^k_T(x)$ be the k-percentile of $\left\{\left\|x - a^{(i)}\right\|\right\}_{i\in T}$. Then, we have that $\left\|x^* - \widetilde{x}\right\|_2 \leq 6d^{60}_{[n]}(\widetilde{x})$. Furthermore, with probability $1 - e^{-\Theta(K)}$, we have*

$$d^{60}_{[n]}(\widetilde{x}) \leq \lambda = d^{60}_{S_1}(\widetilde{x}) \leq 2d^{70}_{[n]}(x^*).$$

Proof Lemma 2.5.1 shows that for all x and $T \subseteq [n]$ with $|T| \leq \frac{n}{2}$

$$\left\|x^* - x\right\|_2 \leq \left(\frac{2n - 2|T|}{n - 2|T|}\right) \max_{i\notin T} \left\|a^{(i)} - x\right\|_2 .$$

Picking T to be the indices of largest 40% of $\left\|a^{(i)} - \widetilde{x}\right\|_2$, we have

$$\left\|x^* - \widetilde{x}\right\|_2 \leq \left(\frac{2n - 0.8n}{n - 0.8n}\right) d_{[n]}^{60}(\widetilde{x}) = 6d_{[n]}^{60}(\widetilde{x}). \tag{2.6}$$

For any point x, we have that $d_{[n]}^{60}(x) \leq d_{S_1}^{65}(x)$ with probability $1 - e^{-\Theta(K)}$ because S_1 is a random subset of $[n]$ with size K. Taking union bound over elements on S_2, with probability $1 - Ke^{-\Theta(K)} = 1 - e^{-\Theta(K)}$, for all points $x \in S_2$

$$d_{[n]}^{60}(x) \leq d_{S_1}^{65}(x). \tag{2.7}$$

yielding that $d_{[n]}^{60}(\widetilde{x}) \leq \lambda$.

Next, for any $i \in S_2$, we have

$$\left\|a^{(i)} - a^{(j)}\right\|_2 \leq \left\|a^{(i)} - x^*\right\|_2 + \left\|x^* - a^{(j)}\right\|_2.$$

and hence

$$d_{[n]}^{70}(a^{(i)}) \leq \left\|a^{(i)} - x^*\right\|_2 + d_{[n]}^{70}(x^*).$$

Again, since S_1 is a random subset of $[n]$ with size K, we have that $d_{S_1}^{65}(a^{(i)}) \leq d_{[n]}^{70}(a^{(i)})$ with probability $1 - Ke^{-\Theta(K)} = 1 - e^{-\Theta(K)}$. Therefore,

$$d_{S_1}^{65}(a^{(i)}) \leq \left\|a^{(i)} - x^*\right\|_2 + d_{[n]}^{70}(x^*).$$

Since S_2 is an independent random subset, with probability $1 - e^{-\Theta(K)}$, there is $i \in S_2$ such that $\left\|a^{(i)} - x^*\right\|_2 \leq d_{[n]}^{70}(x^*)$. In this case, we have

$$d_{S_1}^{65}(a^{(i)}) \leq 2d_{[n]}^{70}(x^*).$$

Since i^* minimizes $d_{S_1}^{65}(a^{(i)})$ over all $i \in S_2$, we have that

$$\lambda \overset{\text{def}}{=} d_{S_1}^{65}(\widetilde{x}) \overset{\text{def}}{=} d_{S_1}^{65}(a^{(i^*)}) \leq d_{S_1}^{65}(a^{(i)}) \leq 2d_{[n]}^{70}(x^*)\,.$$

■

2.5.2 A $1 + \varepsilon$ Approximation of Geometric Median

Here we show how to improve the constant approximation in the previous section to a $1 + \epsilon$ approximation. Our algorithm is essentially stochastic subgradient where we use the information from the previous section to bound the domain in which we need to search for a geometric median.

Theorem 2.5.3 *Let x be the output of* `ApproximateMedian`(ε)*. With probability $1 - e^{-\Theta(1/\varepsilon)}$, we have*

$$\mathbb{E}\left[f(x)\right] \leq (1 + \varepsilon) \min_{x \in \mathbb{R}^d} f(x).$$

```
Algorithm 6: ApproximateMedian($\varepsilon$)
    Input: $a^{(1)}, a^{(2)}, \cdots, a^{(n)} \in \mathbb{R}^d$.
    Let $T = (120/\varepsilon)^2$ and let $\eta = \frac{6\lambda}{n}\sqrt{\frac{2}{T}}$ .
    Let $(x^{(1)}, \lambda) = \texttt{CrudeApproximate}_{\sqrt{T}}(a^{(1)}, a^{(2)}, \cdots, a^{(n)})$ .
    for $k \leftarrow 1, 2, \cdots, T$ do
        Sample $i_k$ from $[n]$ and let
            $g^{(k)} = \begin{cases} n(x^{(k)} - a^{(i_k)})/\left\|x^{(k)} - a^{(i_k)}\right\|_2 & \text{if } x^{(i)} \neq a^{(i_k)} \\ 0 & \text{otherwise} \end{cases}$
        Let $x^{(k+1)} = \text{argmin}_{\left\|x - x^{(1)}\right\|_2 \leq 6\lambda} \eta \left\langle g^{(k)}, x - x^{(k)} \right\rangle + \frac{1}{2}\left\|x - x^{(k)}\right\|_2^2$.
    end
    Output: Output $\frac{1}{T}\sum_{i=1}^{T} x^{(k)}$.
```

Furthermore, the algorithm takes $O(d/\varepsilon^2)$ time.

Proof After computing $x^{(1)}$ and λ the remainder of our algorithm is the stochastic subgradient descent method applied to $f(x)$. It is routine to check that $\mathbb{E}_{i^{(k)}} g^{(k)}$ is a subgradient of f at $x^{(k)}$. Furthermore, since the diameter of the domain, $\left\{x : \left\|x - x^{(1)}\right\|_2 \leq 6\lambda\right\}$, is clearly λ and the norm of sampled gradient, $g^{(k)}$, is at most n, we have that

$$\mathbb{E}\left[f\left(\frac{1}{T}\sum_{i=1}^{T} x^{(k)}\right) - \min_{\left\|x - x^{(1)}\right\|_2 \leq 6\lambda} f(x) \leq\right] 6n\lambda\sqrt{\frac{2}{T}}$$

(see [Bub14, Thm 6.1]). Lemma 2.5.2 shows that $\left\|x^* - x^{(1)}\right\|_2 \leq 6\lambda$ and $\lambda \leq 2d_{[n]}^{70}(x^*)$ with probability $1 - \sqrt{T}e^{-\Theta(\sqrt{T})}$. In this case, we have

$$\mathbb{E}\left[f\left(\frac{1}{T}\sum_{i=1}^{T} x^{(k)}\right) - f(x^*)\right] \leq \frac{12\sqrt{2}nd_{[n]}^{70}(x^*)}{\sqrt{T}}.$$

Since $d_{[n]}^{70}(x^*) \leq \frac{1}{0.3n}f(x^*)$, we have

$$\mathbb{E}\left[f\left(\frac{1}{T}\sum_{i=1}^{T} x^{(k)}\right)\right] \leq \left(1 + \frac{60}{\sqrt{T}}\right) f(x^*) \leq \left(1 + \frac{\varepsilon}{2}\right) f(x^*) .$$

■

2.6 Weighted Geometric Median

In this section, we show how to extend our results to the *weighted geometric median* problem, also known as the Weber problem: given a set of n points in d dimensions, $a^{(1)}, \ldots, a^{(n)} \in \mathbb{R}^d$, with corresponding weights $w^{(1)}, \ldots, w^{(n)} \in \mathbb{R}_{>0}$, find a point $x^* \in \mathbb{R}^d$ that minimizes the weighted sum of Euclidean distances to them:

$$x^* \in \text{argmin}_{x\in\mathbb{R}^d} f(x) \quad \text{where} \quad f(x) \stackrel{\text{def}}{=} \sum_{i\in[n]} w^{(i)} \|x - a^{(i)}\|_2.$$

As in the unweighted problem, our goal is to compute $(1+\epsilon)$-approximate solution, i.e. $x \in \mathbb{R}^d$ with $f(x) \leq (1+\epsilon) f(x^*)$.

First, we show that it suffices to consider the case where the weights are integers with bounded sum (Lemma 2.6.1). Then, we show that such an instance of the weighted geometric median problem can be solved using the algorithms developed for the unweighted problem.

Lemma 2.6.1 *Given points $a^{(1)}, a^{(2)}, \ldots, a^{(n)} \in \mathbb{R}^d$, weights $w^{(1)}, w^{(2)}, \ldots, w^{(n)} \in \mathbb{R}_{>0}$, and $\epsilon \in (0,1)$, we can compute in linear time weights $w_1^{(1)}, w_1^{(2)}, \ldots, w_1^{(n)}$ such that:*

- *Any $(1+\epsilon/5)$-approximate weighted geometric median of $a^{(1)}, \ldots, a^{(n)}$ with the weights $w_1^{(1)}, \ldots, w_1^{(n)}$ is also a $(1+\epsilon)$-approximate weighted geometric median of $a^{(1)}, \ldots, a^{(n)}$ with the weights $w^{(1)}, \ldots, w^{(n)}$, and*

- *$w_1^{(1)}, \ldots, w_1^{(n)}$ are nonnegative integers and$\sum_{i=1}^n w_1^{(i)} \leq 5n\epsilon^{-1}$.*

Proof Let
$$f(x) = \sum_{i\in[n]} w^{(i)} \left\| a^{(i)} - x \right\|$$
and $W = \sum_{i\in[n]} w^{(i)}$. Furthermore, let $\epsilon' = \epsilon/5$ and for each $i \in [n]$, define

$$\begin{aligned} w_0^{(i)} &= \frac{n}{\epsilon' W} w^{(i)} \\ w_1^{(i)} &= \left\lfloor w_0^{(i)} \right\rfloor \\ w_2^{(i)} &= w_0^{(i)} - w_1^{(i)} \end{aligned}$$

We also define $f_0, f_1, f_2, W_0, W_1, W_2$ analogously to f and W.

Now, assume $f_1(x) \leq (1+\epsilon') f_1(x^*)$, where x^* is the minimizer of f and f_0. Then:

$$
\begin{aligned}
f_0(x) &= f_1(x) + f_2(x) \\
&\leq f_1(x) + f_2(x^*) + W_2\|x - x^*\| \\
&= f_1(x) + f_2(x^*) + \frac{W_2}{W_1}\sum w_1^{(i)}\|x - x^*\| \\
&\leq f_1(x) + f_2(x^*) + \frac{W_2}{W_1}\sum w_1^{(i)}\left(\|a^{(i)} - x\| + \|a^{(i)} - x^*\|\right) \\
&= f_1(x) + f_2(x^*) + \frac{W_2}{W_1}(f_1(x) + f_1(x^*)) \\
&\leq f_1(x) + f_2(x^*) + \frac{\epsilon'}{1-\epsilon'}(f_1(x) + f_1(x^*)) \\
&\leq \left(1 + \frac{\epsilon'}{1-\epsilon'}\right)(1+\epsilon')f_1(x^*) + \frac{\epsilon'}{1-\epsilon'}f_1(x^*) + f_2(x^*) \\
&\leq (1+5\epsilon')f_0(x^*) \\
&= (1+\epsilon)f_0(x^*).
\end{aligned}
$$

■

We now proceed to show the main result of this section.

Lemma 2.6.2 *A $(1+\epsilon)$-approximate weighted geometric median of n points in $\mathbb{R}^d$ can be computed in $O(nd\log^3\epsilon^{-1})$ time.*

Proof By applying Lemma 2.6.1, we can assume that the weights are integer and their sum does not exceed $n\epsilon^{-1}$. Note that computing the weighted geometric median with such weights is equivalent to computing an unweighted geometric median of $O(n\epsilon^{-1})$ points (where each point of the original input is repeated with the appropriate multiplicity). We now show how to simulate the behavior of our unweighted geometric median algorithms on such a set of points without computing it explicitly.

If $\epsilon > n^{-1/2}$, we will apply the algorithm ApproximateMedian(ϵ), achieving a runtime of $O(d\epsilon^{-2}) = O(nd)$. It is only necessary to check that we can implement weighted sampling from our points with $O(n)$ preprocessing and $O(1)$ time per sample. This is achieved by the alias method [KP79].

Now assume $\epsilon < n^{-1/2}$. We will employ the algorithm AccurateMedian(ϵ). Note that the initialization using ApproximateMedian(2) can be performed as described in the previous paragraph. It remains to check that we can implement the subroutines LineSearch and ApproxMinEig on the implicitly represented multiset of $O(n\epsilon^{-1})$ points. It is enough to observe only n of the points are distinct, and all computations performed by these subroutines are identical for identical points. The total runtime will thus be $O(nd\log^3(n/\epsilon^2)) = O(nd\log^3\epsilon^{-1})$.

■

Appendix

2.A Properties of the Central Path (Proofs)

Here we provide proofs of the claims in Section 2.3.

2.A.1 Basic Facts

Here we provide basic facts regarding the central path that we will use throughout our analysis. First, we compute various derivatives of the penalized objective function.

Lemma 2.A.1 (Path Derivatives) *We have*

$$\begin{aligned}
\triangledown f_t(x) &= \sum_{i\in[n]} \frac{t^2(x-a^{(i)})}{1+g_t^{(i)}(x)} \quad , \\
\triangledown^2 f_t(x) &= \sum_{i\in[n]} \frac{t^2}{1+g_t^{(i)}(x)} \left(\mathbf{I} - \frac{t^2(x-a^{(i)})(x-a^{(i)})^\top}{g_t^{(i)}(x)(1+g_t^{(i)}(x))} \right) \quad \text{, and} \\
\frac{d}{dt}x_t &= -\left(\triangledown^2 f_t(x_t)\right)^{-1} \sum_{i\in[n]} \frac{t(x_t-a^{(i)})}{(1+g_t^{(i)}(x_t))g_t^{(i)}(x_t)}
\end{aligned}$$

Proof Direct calculation shows that

$$\begin{aligned}
\triangledown f_t^{(i)}(x) &= \frac{t^2(x-a^{(i)})}{\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}} - \frac{1}{1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}} \left(\frac{t^2(x-a^{(i)})}{\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}} \right) \\
&= \frac{t^2(x-a^{(i)})}{1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}} = \frac{t^2(x-a^{(i)})}{1+g_t^{(i)}(x)}
\end{aligned}$$

and

$$\triangledown^2 f_t^{(i)}(x) = \frac{t^2}{1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}}\mathbf{I} - \left(\frac{1}{1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}}\right)^2 \frac{t^2(x-a^{(i)})(x-a^{(i)})^\top}{\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}}$$
$$= \frac{t^2}{1+g_t^{(i)}(x)}\left(\mathbf{I} - \frac{t^2(x-a^{(i)})(x-a^{(i)})^\top}{g_t^{(i)}(x)(1+g_t^{(i)}(x))}\right)$$

and

$$\left(\frac{d}{dt}\triangledown f_t^{(i)}\right)(x) = \frac{2t(x-a^{(i)})}{1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}} - \frac{t^2\cdot(x-a^{(i)})\cdot t\left\|x-a^{(i)}\right\|_2^2}{\left(1+\sqrt{1+t^2\left\|x-a^{(i)}\right\|}\right)^2\sqrt{1+t^2\left\|x-a^{(i)}\right\|_2^2}}$$
$$= \frac{t\cdot(x-a^{(i)})}{1+g_t^{(i)}(x)}\left(2 - \frac{g_t^{(i)}(x)^2-1}{(1+g_t^{(i)}(x))g_t^{(i)}(x)}\right)$$
$$= \frac{t\cdot(x-a^{(i)})}{1+g_t^{(i)}(x)}\left(\frac{2g_t^{(i)}(x)-(g_t^{(i)}(x)-1)}{g_t^{(i)}(x)}\right) = \frac{t\cdot(x-a^{(i)})}{g_t^{(i)}(x)}$$

Finally, by the optimality of x_t we have that $\triangledown f_t(x_t)=0$. Consequently,

$$\nabla^2 f_t(x_t)\frac{d}{dt}x_t + \left(\frac{d}{dt}\nabla f_t\right)(x_t) = 0.$$

and solving for $\frac{d}{dt}x_t$ then yields

$$\frac{d}{dt}x_t = -\left(\nabla^2 f_t(x_t)\right)^{-1}\left(\left(\frac{d}{dt}\nabla f_t\right)(x_t)\right)$$
$$= -\left(\nabla^2 f_t(x_t)\right)^{-1}\left(\left(\frac{d}{dt}\nabla f_t\right)(x_t) - \frac{1}{t}\triangledown f_t(x_t)\right)$$
$$= -\left(\nabla^2 f_t(x_t)\right)^{-1}\left(\sum_{i\in[n]}\left[\frac{t}{g_t^{(i)}} - \frac{t}{1+g_t^{(i)}}\right](x_t-a^{(i)})\right).$$

■

Next, in the following lemma we compute simple approximations of the Hessian of the penalized barrier.

Lemma 2.A.2 *For all $t > 0$ and $x \in \mathbb{R}^d$*

$$\bigtriangledown^2 f_t(x) = \sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}} \left(\mathbf{I} - \left(1 - \frac{1}{g_t^{(i)}(x)} \right) u^{(i)}(x) u^{(i)}(x)^\top \right)$$

and therefore

$$\sum_{i \in [n]} \frac{t^2}{(1 + g_t^{(i)}) g_t^{(i)}} \mathbf{I} \preceq \bigtriangledown^2 f_t(x) \preceq \sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}(x)} \mathbf{I}$$

Proof We have that

$$\begin{aligned} \bigtriangledown^2 f_t(x) &= \sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}(x)} \left(\mathbf{I} - \frac{t^2 (x - a^{(i)})(x - a^{(i)})^\top}{g_t^{(i)}(x)(1 + g_t^{(i)}(x))} \right) \\ &= \sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}(x)} \left(\mathbf{I} - \frac{t^2 \|x - a^{(i)}\|_2^2}{(1 + g_t^{(i)}(x)) g_t^{(i)}(x)} u^{(i)}(x) u^{(i)}(x)^\top \right) \\ &= \sum_i \frac{t^2}{1 + g_i} \left(\mathbf{I} - u^{(i)}(x) u^{(i)}(x)^\top \right) \end{aligned}$$

Since

$$\frac{g_t^{(i)}(x)^2 - 1}{g_t^{(i)}(x)(1 + g_t^{(i)}(x))} = \frac{(g_t^{(i)}(x) + 1)(g_t^{(i)}(x) - 1)}{(1 + g_t^{(i)}(x)) g_t^{(i)}(x)} = 1 - \frac{1}{g_t^{(i)}(x)},$$

the result follows.

■

2.A.2 Stability of Hessian

To show that under ℓ^2 changes the Hessian does not change by much spectrally, we first show that under such changes each $g_t^{(i)}$ does not change by too much.

Lemma 2.A.3 (Stability of g) *For all $x, y \in \mathbb{R}^d$ and $t > 0$, we have*

$$g_t^{(i)}(x) - t\|x - y\|_2 \le g_t^{(i)}(y) \le g_{t\breve{a}}^{(i)}(x) + t\|x - y\|_2$$

Proof Direct calculation reveals that

$$\begin{aligned} g_t^{(i)}(y)^2 &= 1 + t^2 \|x - a^{(i)} + y - x\|_2^2 \\ &= 1 + t^2 \|x - a^{(i)}\|_2^2 + 2t^2 (x - a^{(i)})^\top (y - x) + t^2 \|y - x\|_2^2 \\ &= g_t^{(i)}(x)^2 + 2t^2 (x - a^{(i)})^\top (y - x) + t^2 \|y - x\|_2^2 . \end{aligned}$$

Consequently, by the Cauchy-Schwarz inequality,

$$\begin{aligned} g_t^{(i)}(y)^2 &\le g_t^{(i)}(x)^2 + 2t^2\big\|x - a^{(i)}\big\|_2 \cdot \big\|y - x\big\|_2 + t^2\big\|y - x\big\|_2^2 \\ &\le \left(g_t^{(i)}(x) + t\big\|y - x\big\|_2\right)^2 \end{aligned}$$

and

$$\begin{aligned} g_t^{(i)}(y)^2 &\ge g_t^{(i)}(x)^2 - 2t^2\big\|x - a^{(i)}\big\|_2 \cdot \big\|y - x\big\|_2 + t^2\big\|y - x\big\|_2^2 \\ &\ge \left(g_t^{(i)}(x) - t\big\|y - x\big\|_2\right)^2. \end{aligned}$$

■

Lemma 2.3.1 *Suppose that $\big\|x - y\big\|_2 \le \frac{\epsilon}{t}$ with $\epsilon \le \frac{1}{20}$. Then, we have*

$$(1 - 6\epsilon^{2/3}) \bigtriangledown^2 f_t(x) \preceq \bigtriangledown^2 f_t(y) \preceq (1 + 6\epsilon^{2/3}) \bigtriangledown^2 f_t(x).$$

Proof Write $y - x = \alpha_i v_i + \beta_i u^{(i)}(x)$ for some $v_i \perp u^{(i)}(x)$ with $\big\|v_i\big\|_2 = 1$. Since $\big\|x - y\big\|_2^2 \le \frac{\epsilon^2}{t^2}$, we see that $\alpha_i^2, \beta_i^2 \le \frac{\epsilon^2}{t^2}$.

Now, let $\bar{x} = x + \beta_i u^{(i)}$. Clearly, $u^{(i)}(x) = u^{(i)}(\bar{x})$ and therefore some manipulation reveals that for all unit vectors $z \in \mathbb{R}^d$

$$\begin{aligned} &\left|\left[u^{(i)}(x)^\top z\right]^2 - \left[u^{(i)}(y)z\right]^2\right| \\ &= \left|\left[u^{(i)}(\bar{x})^\top z\right]^2 - \left[u^{(i)}(y)z\right]^2\right| \\ &= \left|\left[\frac{(\bar{x} - a^{(i)})^\top z}{\big\|\bar{x} - a^{(i)}\big\|_2}\right]^2 - \left[\frac{(y - a^{(i)})^\top z}{\big\|y - a^{(i)}\big\|_2}\right]^2\right| \\ &\le \left|\left[\frac{(\bar{x} - a^{(i)})^\top z}{\big\|\bar{x} - a^{(i)}\big\|_2}\right]^2 - \left[\frac{(\bar{x} - a^{(i)})^\top z}{\big\|y - a^{(i)}\big\|_2}\right]^2\right| + \left|\left[\frac{(\bar{x} - a^{(i)})^\top z}{\big\|y - a^{(i)}\big\|_2}\right]^2 - \left[\frac{(y - a^{(i)})^\top z}{\big\|y - a^{(i)}\big\|_2}\right]^2\right| \\ &\le \left|1 - \frac{\big\|\bar{x} - a^{(i)}\big\|_2^2}{\big\|y - a^{(i)}\big\|_2^2}\right| + \frac{\left|\left[(\bar{x} - a^{(i)} + (y - \bar{x}))^\top z\right]^2 - \left[(\bar{x} - a^{(i)})^\top z\right]^2\right|}{\big\|y - a^{(i)}\big\|_2^2} \\ &= \frac{\alpha_i^2 + \left|2\left[(\bar{x} - a^{(i)})^\top z\right] \cdot \left[(y - \bar{x})^\top z\right] + \left[(y - \bar{x})^\top z\right]^2\right|}{\big\|\bar{x} - a^{(i)}\big\|_2^2 + \alpha_i^2}. \end{aligned}$$

Now, $y-\bar{x}=\alpha_i v_i$ and $\left[(y-\bar{x})^\top z\right]^2 \leq \alpha_i^2 \leq \frac{\epsilon^2}{t^2}$. Therefore, by the Young and Cauchy-Schwarz inequalities we have that for all $\gamma > 0$

$$\begin{aligned}
\left|\left[u^{(i)}(x)^\top z\right]^2 - \left[u^{(i)}(y) z\right]^2\right| &\leq \frac{2\alpha_i^2 + 2\left|\left[(\bar{x}-a^{(i)})^\top z\right]\cdot\left[(y-\bar{x})^\top z\right]\right|}{\left\|\bar{x}-a^{(i)}\right\|_2^2 + \alpha_i^2} \\
&\leq \frac{2\alpha_i^2 + \gamma\left[(\bar{x}-a^{(i)})^\top z\right]^2 + \gamma^{-1}\left[(y-\bar{x})^\top z\right]^2}{\left\|\bar{x}-a^{(i)}\right\|_2^2 + \alpha_i^2} \\
&\leq \frac{\alpha_i^2\left(2+\gamma^{-1}\left(v_i^\top z\right)^2\right)}{\left\|\bar{x}-a^{(i)}\right\|_2^2 + \alpha_i^2} + \gamma\left[(u^{(i)}(x))^\top z\right]^2 \\
&\leq \frac{\epsilon^2}{t^2\left\|\bar{x}-a^{(i)}\right\|_2^2 + \epsilon^2}\left(2+\frac{1}{\gamma}\left(v_i^\top z\right)^2\right) + \gamma\left[(u^{(i)}(x))^\top z\right]^2.
\end{aligned}$$

Note that

$$\begin{aligned}
t^2\left\|\bar{x}-a^{(i)}\right\|_2^2 &= t^2\left(\left\|x-a^{(i)}\right\|_2^2 + 2\beta_i(x-a^{(i)})^\top u^{(i)}(x) + \beta_i^2\right) = \left(t\left\|x-a^{(i)}\right\|_2 + t\beta_i\right)^2 \\
&\geq \left(\max\left\{t\left\|x-a^{(i)}\right\|_2 - \epsilon, 0\right\}\right)^2.
\end{aligned}$$

Consequently, if $t\left\|x-a^{(i)}\right\|_2 \geq 2\epsilon^{1/3}\sqrt{g_t^{(i)}(x)}$ then since $\epsilon \leq \frac{1}{20}$ we have that $t^2\left\|\bar{x}-a^{(i)}\right\|_2^2 \geq 2\epsilon^{2/3} g_t^{(i)}(x)$ and therefore letting $\gamma = \frac{\epsilon^{2/3}}{g_t^{(i)}(x)}$ we have that

$$\begin{aligned}
\left|\left[u_t^{(i)}(x)^\top z\right]^2 - \left[u_t^{(i)}(y) z\right]^2\right| &\leq \frac{\epsilon^{4/3}}{2g_t^{(i)}(x)}\left(2+\frac{g_t^{(i)}(x)}{\epsilon^{2/3}}\left[v^\top z\right]^2\right) + \frac{\epsilon^{2/3}}{g_t^{(i)}(x)}\left[(u^{(i)}(x))^\top z\right]^2 \\
&\leq 2\epsilon^{2/3}\left[v^\top z\right]^2 + \frac{\epsilon^{2/3}}{g_t^{(i)}(x)}\left[(u^{(i)}(x))^\top z\right]^2 \\
&\leq 2\epsilon^{2/3}\left(\frac{1+g_t^{(i)}(x)}{t^2}\right)\left\|z\right\|_{\nabla^2 f_t^{(i)}(x)}^2
\end{aligned}$$

and therefore if we let

$$\mathbf{H}_t^{(i)} \stackrel{\text{def}}{=} \frac{t^2}{1+g_t^{(i)}(x)}\left(\mathbf{I} - \left(1-\frac{1}{g_t^{(i)}(x)}\right)u^{(i)}(y)(u^{(i)}(y))^\top\right),$$

we see that for unit vectors z,

$$\left|z^\top\left(\mathbf{H}_t^{(i)} - \nabla^2 f_t^{(i)}(x)\right)z\right| \leq 2\epsilon^{2/3}\left\|z\right\|_{\nabla^2 f_t^{(i)}(x)}^2$$

Otherwise, $t\left\|x - a^{(i)}\right\|_2 < 2\epsilon^{1/3}\sqrt{g_t^{(i)}(x)}$ and therefore

$$g_t^{(i)}(x)^2 = 1 + t^2\left\|x - a^{(i)}\right\|_2^2 \le 1 + 4\epsilon^{2/3} g_t^{(i)}(x)$$

Therefore, we have

$$\begin{aligned} g_t^{(i)}(x) &\le \frac{4\epsilon^{2/3} + \sqrt{(4\epsilon^{2/3})^2 + 4}}{2} \\ &\le 1 + 4\epsilon^{2/3}. \end{aligned}$$

Therefore,

$$\frac{1}{1+4\epsilon^{2/3}}\mathbf{H}_t^{(i)} \preceq \frac{t^2}{(1+g_t^{(i)}(x))g_t^{(i)}(x)}\mathbf{I} \preceq \bigtriangledown^2 f_t^{(i)}(x) \preceq \frac{t^2}{(1+g_t^{(i)}(x))}\mathbf{I} \preceq \left(1+4\epsilon^{2/3}\right)\mathbf{H}_t^{(i)}.$$

In either case, we have that

$$\left| z^\top \left(\mathbf{H}_t^{(i)} - \bigtriangledown^2 f_t^{(i)}(x)\right) z \right| \le 4\epsilon^{2/3}\left\|z\right\|^2_{\bigtriangledown^2 f_t^{(i)}(x)}.$$

Now, we note that $\left\|x - y\right\|_2 \le \frac{\epsilon}{t} \le \epsilon \cdot \frac{g_t^{(i)}(x)}{t}$. Therefore, by Lemma 2.A.3. we have that

$$(1-\epsilon)g_t^{(i)}(x) \le g_t^{(i)}(y) \le (1+\epsilon)g_t^{(i)}(x)$$

Therefore, we have

$$\frac{1-4\epsilon^{2/3}}{(1+\epsilon)^2}\bigtriangledown^2 f_t^{(i)}(x) \preceq \frac{1}{(1+\epsilon)^2}\mathbf{H}_t^{(i)} \preceq \bigtriangledown^2 f_t^{(i)}(y) \preceq \frac{1}{(1-\epsilon)^2}\mathbf{H}_t^{(i)} \preceq \frac{1+4\epsilon^{2/3}}{(1-\epsilon)^2}\bigtriangledown^2 f_t^{(i)}(x)$$

Since $\epsilon < \frac{1}{20}$, the result follows.

$\blacksquare$

Consequently, so long as we have a point within a $O(\frac{1}{t})$ sized Euclidean ball of some x_t Newton's method (or an appropriately transformed first order method) within the ball will converge quickly.

2.A.3 How Much Does the Hessian Change Along the Path?

Lemma 2.3.2 *For all $t \ge 0$ and $i \in [n]$, the following inequalities hold:*

$$\left\|\frac{d}{dt}x_t\right\|_2 \le \frac{1}{t^2}\bar{g}_t(x_t) \ , \quad \left|\frac{d}{dt}g_t^{(i)}(x_t)\right| \le \frac{1}{t}\left(g_t^{(i)}(x_t) + \bar{g}_t\right) \ , \textit{ and } \quad \left|\frac{d}{dt}w_t\right| \le \frac{2}{t}w_t.$$

Consequently, for all $t' \geq t$ we have that $\left(\frac{t}{t'}\right)^2 w_t \leq w_{t'} \leq \left(\frac{t'}{t}\right)^2 w_t$.

Proof From Lemma 2.A.1 we know that

$$\frac{d}{dt}x_t = -\left(\bigtriangledown^2 f_t(x_t)\right)^{-1} \sum_{i\in[n]} \frac{t(x_t - a^{(i)})}{(1+g_t^{(i)}(x_t))g_t^{(i)}(x_t)}$$

and by Lemma 2.A.2 we know that

$$\bigtriangledown^2 f_t(x_t) \succeq \sum_{i\in[n]} \frac{t^2}{(1+g_t^{(i)}(x_t))g_t^{(i)}(x_t)}\mathbf{I} = \frac{t^2}{\bar{g}_t(x_t)} \sum_{i\in[n]} \frac{1}{1+g_t^{(i)}(x_t)}\mathbf{I}\,.$$

Using this fact and the fact that $t\left\|x_t - a^{(i)}\right\|_2 \leq g_t^{(i)}$ we have

$$\begin{aligned}
\left\|\frac{d}{dt}x_t\right\|_2 &= \left\|-\left(\bigtriangledown^2 f_t(x_t)\right)^{-1} \frac{d}{dt} \bigtriangledown f_t(x_t)\right\|_2 \\
&\leq \left(\frac{t^2}{\bar{g}_t(x_t)} \sum_{i\in[n]} \frac{1}{1+g_t^{(i)}(x_t)}\right)^{-1} \sum_{i\in[n]} \left\|\frac{t}{g_t^{(i)}(x_t)(1+g_t^{(i)}(x_t))}(x_t - a^{(i)})\right\|_2 \\
&\leq \frac{\bar{g}_t(x_t)}{t^2}
\end{aligned}$$

Next, we have

$$\begin{aligned}
\frac{d}{dt}g_t^{(i)}(x_t) &= \frac{d}{dt}\left(1 + t^2\left\|x_t - a^{(i)}\right\|_2^2\right)^{\frac{1}{2}} \\
&= \frac{1}{2}\cdot g_t^{(i)}(x_t)^{-1}\left(2t\left\|x_t - a^{(i)}\right\|_2^2 + 2t^2(x_t - a^{(i)})^\top \frac{d}{dt}x_t\right)
\end{aligned}$$

which by the Cauchy-Schwarz inequality and the fact that $t\left\|x_t - a^{(i)}\right\|_2 \leq g_t^{(i)}(x_t)$ yields the second equation. Furthermore,

$$\begin{aligned}
\left|\frac{d}{dt}w_t\right| = \left|\frac{d}{dt}\sum_i \frac{1}{1+g_t^{(i)}(x_t)}\right| &\leq \sum_i \left|\frac{d}{dt}\frac{1}{1+g_t^{(i)}(x_t)}\right| \\
&= \sum_i \left|\frac{1}{(1+g_t(x_t))^2}\frac{d}{dt}g_t(x_t)\right| \\
&\leq \frac{1}{t}\sum_i \frac{g_t^{(i)}(x_t) + \bar{g}_t}{(1+g_t^{(i)}(x_t))g_t^{(i)}(x_t)} \leq 2\frac{w_t}{t}
\end{aligned}$$

which yields the third equation.

Finally, using our earlier results and Jensen's inequality yields that

$$
\begin{aligned}
|\ln w_{t'} - \ln w_t| &= \left| \int_t^{t'} \frac{\frac{d}{d\alpha} w_\alpha}{w_\alpha} d\alpha \right| \leq \int_t^{t'} \frac{\left(2\frac{w_\alpha}{\alpha}\right)}{w_\alpha} d\alpha = 2 \int_t^{t'} \frac{1}{\alpha} d\alpha \\
&= \ln \left(\frac{t'}{t} \right)^2 .
\end{aligned}
$$

Exponentiating the above inequality yields the final inequality.

■

Lemma 2.3.3 *For all $t \geq 0$, we have*

$$
-12 \cdot t \cdot w_t \mathbf{I} \preceq \frac{d}{dt} \left[\bigtriangledown^2 f_t(x_t) \right] \preceq 12 \cdot t \cdot w_t \mathbf{I} \tag{2.3}
$$

and therefore for all $\beta \in [0, \frac{1}{8}]$

$$
\bigtriangledown^2 f(x_t) - 15\beta t^2 w_t \mathbf{I} \preceq \bigtriangledown^2 f(x_{t(1+\beta)}) \preceq \bigtriangledown^2 f(x_t) + 15\beta t^2 w_t \mathbf{I} \,. \tag{2.4}
$$

Proof Let

$$
\mathbf{A}_t^{(i)} \stackrel{\text{def}}{=} \frac{t^2 (x_t - a^{(i)})(x_t - a^{(i)})^\top}{(1 + g_t^{(i)}) g_t^{(i)}}
$$

and recall that $\bigtriangledown^2 f_t(x_t) = \sum_{i \in [n]} \frac{t^2}{1+g_t^{(i)}} \left(\mathbf{I} - \mathbf{A}_t^{(i)} \right)$. Consequently,

$$
\begin{aligned}
\frac{d}{dt} \bigtriangledown^2 f_t(x_t) &= \frac{d}{dt} \left(\sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}} \left(\mathbf{I} - \mathbf{A}_t^{(i)} \right) \right) \\
&= 2t \left(\frac{1}{t^2} \right) \bigtriangledown^2 f_t(x_t) + t^2 \sum_i \frac{-\frac{d}{dt} g_t^{(i)}}{(1 + g_t^{(i)})^2} \left(\mathbf{I} - \mathbf{A}_t^{(i)} \right) - \sum_i \frac{t^2}{1 + g_t^{(i)}} \frac{d}{dt} \mathbf{A}_t^{(i)}
\end{aligned}
$$

Now, since $\mathbf{0} \preceq \mathbf{A}_t^{(i)} \preceq \mathbf{I}$ and therefore $0 \preceq \bigtriangledown^2 f_t(x_t) \preceq t^2 w_t \mathbf{I}$. For all unit vectors v, using Lemma 2.3.2, we have that

$$
\begin{aligned}
\left| v^\top \left(\frac{d}{dt} \bigtriangledown^2 f_t(x_t) \right) v \right| &\leq 2t \cdot w_t \cdot \left\| v \right\|_2^2 + t^2 \sum_i \frac{\left| \frac{d}{dt} g_t^{(i)} \right|}{(1 + g_t^{(i)})^2} \left\| v \right\|_2^2 + \sum_i \frac{t^2}{1 + g_t^{(i)}} \left| v^\top \left(\frac{d}{dt} \mathbf{A}_t^{(i)} \right) v \right| \\
&\leq 4t \cdot w_t + \sum_{i \in [n]} \frac{t^2}{1 + g_t^{(i)}} \left| v^\top \left(\frac{d}{dt} \mathbf{A}_t^{(i)} \right) v \right| .
\end{aligned}
$$

Since

$$\frac{d}{dt}\mathbf{A}_t^{(i)} = 2t\left(\frac{1}{t^2}\right)\mathbf{A}_t^{(i)} - \left(\frac{t}{(1+g_t^{(i)})g_t^{(i)}}\right)^2\left[(1+g_t^{(i)})\frac{d}{dt}g_t^{(i)} + g_t^{(i)}\frac{d}{dt}g_t^{(i)}\right](x_t - a^{(i)})(x_t - a^{(i)})^\top$$
$$+ \frac{t^2}{(1+g_t^{(i)})g_t^{(i)}}\left[(x_t - a^{(i)})(\frac{d}{dt}x_t)^\top + (\frac{d}{dt}x_t)(x_t - a^{(i)})^\top\right],$$

we have

$$\begin{aligned}\left|v^\top\left(\frac{d}{dt}\mathbf{A}_t^{(i)}\right)v\right| &\leq \left(\frac{2}{t} + \frac{2t^2\left|\frac{d}{dt}g_t^{(i)}\right|}{(1+g_t^{(i)})(g_t^{(i)})^2}\left\|x_t - a^{(i)}\right\|_2^2 + \frac{2t^2\left\|x_t - a^{(i)}\right\|_2\left\|\frac{d}{dt}x_t\right\|_2}{(1+g_t^{(i)})g_t^{(i)}}\right)\left\|v\right\|_2^2 \\ &\leq \frac{2}{t} + \frac{2}{t}\cdot\frac{g_t^{(i)} + \bar{g}_t}{1+g_t^{(i)}} + \frac{2}{t}\cdot\frac{\bar{g}_t}{1+g_t^{(i)}} \\ &\leq \frac{4}{t} + \frac{4}{t}\frac{\bar{g}_t}{1+g_t^{(i)}}.\end{aligned}$$

Consequently, we have

$$\left|v^\top\left(\frac{d}{dt}\bigtriangledown^2 f_t(x_t)\right)v\right| \leq 8t\cdot w_t + 4t\sum_i \frac{\bar{g}_t}{(1+g_t^{(i)})^2} \leq 12t\cdot w_t$$

which completes the proof of (2.3). To prove (2.4), let v be any unit vector and note that

$$\begin{aligned}\left|v^\top\left(\bigtriangledown^2 f_{t(1+\beta)}(x) - \bigtriangledown^2 f_t(x)\right)v\right| &= \left|\int_t^{t(1+\beta)} v^\top\frac{d}{dt}\left[\bigtriangledown^2 f_\alpha(x_\alpha)\right]v\cdot d\alpha\right| \\ &\leq 12\int_t^{t(1+\beta)} \alpha\cdot w_\alpha d\alpha \\ &\leq 12\int_t^{t(1+\beta)} \alpha\left(\frac{\alpha}{t}\right)^2 w_t d\alpha \\ &\leq \frac{12}{t^2}\left(\frac{1}{4}\left[t(1+\beta)\right]^4 - \frac{1}{4}t^4\right)w_t \\ &= 3t^2\left[(1+\beta)^4 - 1\right]w_t \\ &\leq 15t^2\beta w_t\end{aligned}$$

where we used $0 \leq \beta \leq \frac{1}{8}$ in the last line.

■

2.A.4 Where is the next Optimal Point?

Lemma 2.3.4 *For all t, we have $\frac{1}{2}\left[t^2 \cdot w_t \mathbf{I} - (t^2 \cdot w_t - \mu_t) v_t v_t^\top\right] \preceq \bigtriangledown^2 f_t(x_t) \preceq t^2 \cdot w_t \mathbf{I} - (t^2 \cdot w_t - \mu_t) v_t v_t^\top$.*

Proof This follows immediately from Lemma 2.A.2 and Lemma 2.D.1.

■

Lemma 2.3.5 (The Central Path is Almost Straight) *For all $t \geq 0$, $\beta \in [0, \frac{1}{600}]$, and any unit vector y with $|\langle y, v_t\rangle| \leq \frac{1}{t^2 \cdot \kappa}$ where $\kappa = \max_{\delta \in [t,(1+\beta)t]} \frac{w_\delta}{\mu_\delta}$, we have $y^\top(x_{(1+\beta)t} - x_t) \leq \frac{6\beta}{t}$.*

Proof Clearly,

$$\begin{aligned}
y^\top(x_{(1+\beta)t} - x_t) &= \int_t^{(1+\beta)t} y^\top \frac{d}{d\alpha} x_\alpha d_\alpha \leq \int_\beta^{(1+\beta)t} \left| y^\top \frac{d}{d\alpha} x_\alpha \right| d_\alpha \\
&\leq \int_t^{(1+\beta)t} \left| y^\top \left(\bigtriangledown^2 f_\alpha(x_\alpha)\right)^{-1} \sum_{i\in[n]} \frac{\alpha}{(1+g_\alpha^{(i)})g_\alpha^{(i)}}(x_\alpha - a^{(i)}) \right| d_\alpha \\
&\leq \int_t^{(1+\beta)t} \left\| \left(\bigtriangledown^2 f_\alpha(x_\alpha)\right)^{-1} y \right\|_2 \cdot \left\| \sum_{i\in[n]} \frac{\alpha}{(1+g_\alpha^{(i)})g_\alpha^{(i)}}(x_\alpha - a^{(i)}) \right\|_2 d_\alpha
\end{aligned}$$

Now since $\alpha\left\|x_\alpha - a^{(i)}\right\|_2 \leq g_\alpha^{(i)}$, invoking Lemma 2.3.2 yields that

$$\left\| \sum_{i\in[n]} \frac{\alpha}{(1+g_\alpha^{(i)})g_\alpha^{(i)}}(x_\alpha - a^{(i)}) \right\|_2 \leq \sum_{i\in[n]} \frac{1}{1+g_\alpha^{(i)}} = w_\alpha \leq \left(\frac{\alpha}{t}\right)^2 w_t.$$

and invoking Lemma 2.3.3 and the Lemma 2.3.4, we have that

$$\bigtriangledown^2 f_\alpha(x_\alpha) \succeq \bigtriangledown^2 f_t(x_t) - 15\beta t^2 w_t \mathbf{I} \succeq \frac{1}{2}\left[t^2 \cdot w_t \mathbf{I} - (t^2 \cdot w_t - \mu_t) v_t v_t^\top\right] - 15\beta t^2 w_t \mathbf{I}.$$

Let $\mathbf{H}_\alpha = \bigtriangledown^2 f_\alpha(x_\alpha)$ and $\mathbf{H}_t = \bigtriangledown^2 f_t(x_t)$. Then Lemma 2.3.3 shows that

$$\mathbf{H}_\alpha = \mathbf{H}_t + \Delta_\alpha,$$

where $\left\|\Delta_\alpha\right\|_2 \leq 15\beta t^2 w_t$. Now, we note that

$$\mathbf{H}_\alpha^2 = \mathbf{H}_t^2 + \Delta_\alpha \mathbf{H}_t + \mathbf{H}_t \Delta_\alpha + \Delta_\alpha^2 .$$

Therefore, we have

$$\begin{aligned}\left\|\mathbf{H}_\alpha^2 - \mathbf{H}_t^2\right\|_2 &\le \left\|\Delta_\alpha \mathbf{H}_t\right\|_2 + \left\|\mathbf{H}_t\Delta_\alpha\right\|_2 + \left\|\Delta_\alpha^2\right\|_2 \\ &\le 2\left\|\Delta\right\|_2\left\|\mathbf{H}_t\right\|_2 + \left\|\Delta\right\|_2^2 \\ &\le 40\beta t^4 w_t^2.\end{aligned}$$

Let S be the subspace orthogonal to v_t. Then, Lemma 2.3.4 shows that $\mathbf{H}_t \succeq \frac{1}{2}t^2 w_t \mathbf{I}$ on S and hence $\mathbf{H}_t^2 \succeq \frac{1}{4}t^4 w_t^2 \mathbf{I}$ on S. Since $\left\|\mathbf{H}_\alpha^2 - \mathbf{H}_t^2\right\|_2 \le 40\beta t^4 w_t^2$, we have that

$$\mathbf{H}_\alpha^2 \succeq \frac{1}{4}t^4 w_t^2 - 40\beta t^4 w_t^2 \mathbf{I} \text{ on } S$$

and hence

$$\mathbf{H}_\alpha^{-2} \preceq \left(\frac{1}{4}t^4 w_t^2 - 40\beta t^4 w_t^2\right)^{-1} \mathbf{I} \text{ on } S.$$

Therefore, for any $z \in S$, we have

$$\left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} z\right\|_2 = \left\|\mathbf{H}_\alpha^{-1} z\right\|_2 \le \frac{\left\|z\right\|_2}{\sqrt{\frac{1}{4}t^4 w_t^2 - 40\beta t^4 w_t^2}}.$$

Now, we split $y = z + \langle y, v_t\rangle v_t$ where $z \in S$. Then, we have that

$$\begin{aligned}\left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} y\right\|_2 &\le \left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} z\right\|_2 + \left|\langle y, v_t\rangle\right| \left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} v_t\right\|_2 \\ &\le \frac{1}{\sqrt{\frac{1}{4}t^4 w_t^2 - 40\beta t^4 w_t^2}} + \frac{1}{t^2 \cdot \kappa}\left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} v_t\right\|_2.\end{aligned}$$

Note that, we also know that $\lambda_{\min}(\nabla^2 f_\alpha(x_\alpha)) \ge \mu_\alpha$ and hence $\lambda_{\max}(\nabla^2 f_\alpha(x_\alpha)^{-2}) \ge \mu_\alpha^{-2}$. Therefore, we have

$$\begin{aligned}\left\|\left(\nabla^2 f_\alpha(x_\alpha)\right)^{-1} y\right\|_2 &\le \frac{1}{\sqrt{\frac{1}{4} - 40\beta t^2 w_t}} + \frac{1}{t^2}\frac{\mu_\alpha}{w_\alpha}\frac{1}{\mu_\alpha} \\ &\le \frac{1}{t^2 w_t}\left(2 + \frac{1}{\sqrt{\frac{1}{4} - 40\beta}}\right) \le \frac{5}{t^2 w_t}.\end{aligned}$$

Combining these yields that

$$y^\top(x_{(1+\beta)t} - x_t) \leq \int_t^{(1+\beta)t} \frac{5}{t^2 w_t}\left(\frac{\alpha}{t}\right)^2 w_t d_\alpha \leq \frac{5}{t^4}\left(\frac{1}{3}(1+\beta)^3 t^3 - \frac{1}{3}t^3\right)$$
$$\leq \frac{5}{3t}\left[(1+\beta)^3 - 1\right] \leq \frac{6\beta}{t}.$$

■

2.A.5 Where is the End?

Lemma 2.A.4 *For all $t > 0$, we have that $f(x_t) - f(x^*) \leq \frac{2n}{t}$.*

Proof By definition of x_t we know that $\bigtriangledown f_t(x_t) = 0$. consequently $\frac{1}{t} \bigtriangledown f_t(x_t)^\top(x_t - x^*) = 0$ and by Lemma 2.A.1 we have

$$\sum_{i\in[n]} \frac{t(x_t - a^{(i)})^\top(x_t - x^*)}{1 + g_t^{(i)}(x)} = 0$$

and therefore

$$\sum_{i\in[n]} \frac{t\left\|x_t - a^{(i)}\right\|_2^2 + t(x_t - a^{(i)})^\top(a^{(i)} - x^*)}{1 + g_t^{(i)}(x_t)} = 0$$

and therefore, by the Cauchy-Schwarz inequality

$$\sum_{i\in[n]} \frac{t\left\|x_t - a^{(i)}\right\|_2^2}{1 + g_t^{(i)}(x_t)} \leq \sum_{i\in[n]} \frac{t\left\|x_t - a^{(i)}\right\|_2 \left\|a^{(i)} - x^*\right\|_2}{1 + g_t^{(i)}(x_t)}$$

Consequently, using $t\left\|x_t - a^{(i)}\right\|_2 \leq g_t^{(i)}(x_t)$ and that $1 + g_t^{(i)}(x_t) \leq 2 + t\left\|x_t - a^{(i)}\right\|_2$

$$\begin{aligned}
0 &\leq \sum_{i\in[n]} \frac{t\left\|x_t - a^{(i)}\right\|_2 (\left\|x^* - a^{(i)}\right\|_2 - \left\|x_t - a^{(i)}\right\|_2)}{1 + g_t^{(i)}(x_t)} \\
&\leq \sum_{i\in[n]} \left\|x^* - a^{(i)}\right\|_2 - \sum_{i\in[n]} \frac{t\left\|x_t - a^{(i)}\right\|_2^2}{1 + g_t^{(i)}(x_t)} \\
&\leq f(x^*) - \sum_{i\in[n]} t\left\|x_t - a^{(i)}\right\|_2 + \sum_{i\in[n]} \frac{2\left\|x_t - a^{(i)}\right\|_2}{1 + g_t^{(i)}(x_t)} \\
&\leq f(x^*) - f(x_t) + \frac{2n}{t}
\end{aligned}$$

■

Lemma 2.3.6 *For all $t > \frac{1}{400 f(x^*)}$, we have $f(x_t) - f(x^*) \leq 100n\sqrt{\frac{f(x^*)}{t}}$.*

Proof Let $S = \{i \text{ such that } \|x^* - x_t\|_2 \leq K\|a^{(i)} - x_t\|_2\}$ for some $K > 2$. By the convexity, we have that

$$\begin{aligned}\sum_{i\in S}\|a^{(i)} - x_t\|_2 &\leq \frac{1}{t}\sum_{i\in S}\sqrt{1 + t^2\|a^{(i)} - x_t\|_2^2}\\ &\leq \frac{1}{t}\sum_{i\in S}\sqrt{1 + t^2\|a^{(i)} - x^*\|_2^2} - \sum_{i\in S}\frac{t\left\langle x_t - a^{(i)}, x^* - x_t\right\rangle}{\sqrt{1 + t^2\|x_t - a^{(i)}\|_2^2}}\\ &\leq \sum_{i\in S}\|a^{(i)} - x^*\|_2 + \frac{|S|}{t} - \sum_{i\in S}\frac{t\left\langle x_t - a^{(i)}, x^* - x_t\right\rangle}{\sqrt{1 + t^2\|x_t - a^{(i)}\|_2^2}} \qquad (2.8)\end{aligned}$$

For all $i \notin S$, we have

$$\begin{aligned}\|a^{(i)} - x^*\|_2 &\geq \|x^* - x_t\|_2 - \|a^{(i)} - x_t\|_2\\ &\geq \frac{K-2}{K}\|x^* - x_t\|_2 + \|a^{(i)} - x_t\|_2.\end{aligned}$$

Putting it into (2.8), we have that

$$\begin{aligned}f(x_t) - f(x^*) &= \sum_{i\in[n]}\|a^{(i)} - x_t\|_2 - \sum_{i\in[n]}\|a^{(i)} - x^*\|_2\\ &\leq \frac{|S|}{t} - \sum_{i\in S}\frac{t\left\langle x_t - a^{(i)}, x^* - x_t\right\rangle}{\sqrt{1 + t^2\|x_t - a^{(i)}\|_2^2}} - \frac{K-2}{K}\left|S^c\right|\|x^* - x_t\|_2\end{aligned}$$

Now, we use the optimality condition

$$\sum_{i\in[n]}\frac{t^2(x - a^{(i)})}{1 + \sqrt{1 + t^2\|x - a^{(i)}\|_2^2}} = 0$$

and get

$$f(x_t) - f(x^*) \leq \frac{|S|}{t} - \frac{K-2}{K}\left|S^c\right|\|x^* - x_t\|_2 + \sum_{i\in[n]}\frac{\left\langle v^{(i)}, x^* - x_t\right\rangle}{1 + \sqrt{1 + \|v^{(i)}\|_2^2}} - \sum_{i\in S}\frac{\left\langle v^{(i)}, x^* - x_t\right\rangle}{\sqrt{1 + \|v^{(i)}\|_2^2}}.$$

where $v^{(i)} = t(x_t - a^{(i)})$. For $i \in S$, we have

$$\begin{aligned}
\left| \frac{\left\langle v^{(i)}, x^* - x_t \right\rangle}{1 + \sqrt{1 + \left\|v^{(i)}\right\|_2^2}} - \frac{\left\langle v^{(i)}, x^* - x_t \right\rangle}{\sqrt{1 + \left\|v^{(i)}\right\|_2^2}} \right| &\leq \frac{\left\|v^{(i)}\right\|_2 \left\|x^* - x_t\right\|_2}{\sqrt{1 + \left\|v^{(i)}\right\|_2^2}\left(1 + \sqrt{1 + \left\|v^{(i)}\right\|_2^2}\right)} \\
&\leq \frac{\left\|x^* - x_t\right\|_2}{1 + \sqrt{1 + \left\|v^{(i)}\right\|_2^2}} \leq \frac{K}{t}
\end{aligned}$$

where the last line used $\left\|x^* - x_t\right\|_2 \leq K\left\|a^{(i)} - x_t\right\|_2$. Therefore, we have

$$\begin{aligned}
f(x_t) - f(x^*) &\leq \frac{|S|}{t} + \frac{K|S|}{t} - \frac{K-2}{K}\left|S^c\right| \left\|x^* - x_t\right\|_2 + \sum_{i \notin S} \frac{\left\langle v^{(i)}, x^* - x_t \right\rangle}{1 + \sqrt{1 + \left\|v^{(i)}\right\|_2^2}} \\
&\leq \frac{|S|}{t} + \frac{K|S|}{t} - \frac{K-2}{K}\left|S^c\right| \left\|x^* - x_t\right\|_2 + \left|S^c\right| \left\|x^* - x_t\right\|_2 \\
&= \frac{2K|S|}{t} + \frac{2}{K}\left|S^c\right| \left\|x^* - x_t\right\|_2 \qquad (2.9)
\end{aligned}$$

where the last line used $K > 2$. If $\left\|x^* - x_t\right\|_2 \leq \frac{4}{t}$, then

$$f(x_t) - f(x^*) \leq \frac{4n}{t}. \qquad (2.10)$$

Otherwise, we put $K = \sqrt{t\left\|x^* - x_t\right\|_2} > 2$ into (2.9) and have

$$f(x_t) - f(x^*) \leq 2n\sqrt{\frac{\left\|x^* - x_t\right\|_2}{t}}.$$

Now, we use optimality condition and note that x_t is a weighted average of $a^{(i)}$ and hence

$$\left\|x^* - x_t\right\|_2 \leq \max_{i \in [n]} \left\|x^* - a^{(i)}\right\|_2 \leq f(x^*).$$

Hence, we have

$$f(x_t) - f(x^*) \leq 2n\sqrt{\frac{f(x^*)}{t}}. \qquad (2.11)$$

Putting (2.10) and (2.11) together, we have

$$f(x_t) - f(x^*) \leq \max\left(2n\sqrt{\frac{f(x^*)}{t}}, \frac{4n}{t}\right).$$

Now, the result follows from $t \geq \frac{1}{400 f(x^*)}$.

■

2.A.6 Simple Lemmas

Here we provide various small technical results that we will use to bound the accuracy with which we need to carry out various operations in our algorithm.

Lemma 2.A.5 *For any x, we have that $\left\|x - x_t\right\|_2 \leq f(x)$.*

Proof Since $\sum_{i\in[n]} \left\|x - a^{(i)}\right\|_2 = f(x)$, we have that $\left\|x - a^{(i)}\right\|_2 \leq f(x)$ for all i. By the optimality condition of x_t, x_t is a weighted average of $a^{(i)}$ and hence $\left\|x - x_t\right\|_2 \leq f(x)$.

■

Lemma 2.A.6 *Suppose that $\left\|x - x_t\right\|_2 \leq \frac{1}{100t}$ and $t \geq \frac{1}{400 f(x^*)}$, then we have $f(x) \leq 3000n \cdot f(x^*)$.*

Proof Lemma 2.3.6 shows that

$$f(x_t) \leq f(x^*) + 100n\sqrt{\frac{f(x^*)}{t}} \leq 2001 n f(x^*).$$

Therefore, we have

$$f(x) \leq f(x_t) + n\left\|x - x_t\right\|_2 \leq f(x_t) + \frac{n}{100t} \leq f(x_t) + 4nf(x^*).$$

■

Lemma 2.A.7 *For all $t \geq 0$, we have*

$$\frac{t^2 \cdot w_t(x)}{\mu_t(x)} \leq \bar{g}_t(x) \leq 1 + t \cdot f(x).$$

In particular, if $\left\|x - x_t\right\|_2 \leq \frac{1}{100t}$ and $t \geq \frac{1}{400 f(x^*)}$, we have $\frac{t^2 \cdot w_t(x)}{\mu_t(x)} \leq \bar{g}_t(x) \leq 3100nt \cdot f(x^*)$.

Proof Note that $\mu_t(x) \geq \sum_{i\in[n]} \frac{t^2}{g_t^{(i)}(x)(1+g_t^{(i)}(x))}$ yielding the first claim $\frac{t^2 \cdot w_t(x)}{\mu_t(x)} \leq \bar{g}_t(x)$. To obtain the second, we note that

$$\begin{aligned} \bar{g}_t(x) &\leq 1 + t \cdot \max_i \left\|x - a_i^{(i)}\right\| \\ &\leq 1 + t \cdot f(x). \end{aligned}$$

Now, we use Lemma 2.A.6 to show that $\frac{t^2 \cdot w_t(x)}{\mu_t(x)} \leq 1 + t \cdot f(x) \leq 1 + 3000nt \cdot f(x^*) \leq 3100nt \cdot f(x^*)$

■

Lemma 2.A.8 *Suppose that* $\left\|x - x_t\right\|_2 \leq \frac{1}{100t}$ *and* $t \geq \frac{1}{400f(x^*)}$*, then we have*

$$\frac{nt^2}{2}\left\|x - x_t\right\|_2^2 + f_t(x_t) \geq f_t(x) \geq f_t(x_t) + \frac{1}{10^8 n^2 \cdot f(x^*)^2}\left\|x - x_t\right\|_2^2.$$

Proof For the first inequality, note that $\bigtriangledown^2 f_t(x) \preceq \sum_{i\in[n]} \frac{t^2}{1+g_t^{(i)}(x)}\mathbf{I} \preceq n \cdot t^2\mathbf{I}$. Consequently, if we let $n \cdot t^2\mathbf{I} = \mathbf{H}$ in Lemma 2.D.5, we have that

$$f_t(x) - f_t(x_t) \leq \frac{1}{2}\left\|x - x_t\right\|_{\mathbf{H}}^2 \leq \frac{nt^2}{2}\left\|x - x_t\right\|_2^2$$

For the second inequality, Lemma 2.A.2 to show that

$$\nabla^2 f_t(x) \succeq \sum_{i\in[n]} \frac{t^2}{(1 + g_t^{(i)}(x))g_t^{(i)}(x)}\mathbf{I}.$$

Lemma 2.A.7 shows that $\bar{g}_t(x) \leq 3100nt \cdot f(x^*)$. Since $\bar{g}_t(x)$ is a weighted harmonic mean of $g_t^{(i)}(x)$, there is some i such that $g_t^{(i)}(x) \leq 3100nt \cdot f(x^*)$. Therefore, we have

$$\begin{aligned}\nabla^2 f_t(x) &\succeq \frac{1}{3100n \cdot f(x^*)3200n \cdot f(x^*)}\mathbf{I}\\ &\succeq \left(\frac{1}{5000n \cdot f(x^*)}\right)^2\mathbf{I}.\end{aligned}$$

Now, we apply Lemma 2.D.5 and get

$$f_t(x) \geq f_t(x_t) + \frac{1}{2}\left(\frac{1}{5000n \cdot f(x^*)}\right)^2\left\|x - x_t\right\|_2^2.$$

■

2.B Nearly Linear Time Geometric Median (Proofs)

Here we provide proofs, algorithms, and technical lemmas from Section 2.4.

2.B.1 Eigenvector Computation and Hessian Approximation

Lemma 2.B.1 (Power Method) *Let* $\mathbf{A} \in \mathbb{R}^{d\times d}$ *is a symmetric matrix with eigenvalues* $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d \geq 0$*, corresponding eigenvectors* $v_1, ..., v_d \in \mathbb{R}^d$*, and* $g \stackrel{\text{def}}{=} \frac{\lambda_1 - \lambda_2}{\lambda_1}$*. For any* $\epsilon > 0$ *and* $k \geq \frac{\alpha}{g}\log(\frac{d}{\epsilon})$ *for large enough constant* α*, in time* $O(\text{nnz}(\mathbf{A}) \cdot \log(\frac{d}{\epsilon}))$*, the algorithm* `PowerMethod`$(\mathbf{A}, k)$ *outputs a vector* u *such that*

$$\langle v_1, u\rangle^2 \geq 1 - \epsilon \text{ and } u^\top \mathbf{A} u \geq (1 - \epsilon)\lambda_1$$

Algorithm 7: `PowerMethod`$(\mathbf{A}, k)$

```
Input: a positive definite matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$, and the number of iterations $k$.
Let $x \sim \mathcal{N}(0,1)$ be drawn from a $d$ dimensional normal distribution.
Let $y = \mathbf{A}^k z$
Output: $u = y/\left\|y\right\|_2$
```

with high probability in d.

Proof We write $u = \sum_{i\in[d]} \alpha_i v_i$. Then, we have

$$\langle v_1, u\rangle^2 = \left\langle v_1, \frac{\sum_{i\in[d]} \alpha_i \lambda_i^k v_i}{\sqrt{\sum_{i\in[d]} \alpha_i^2 \lambda_i^{2k}}} \right\rangle^2 = \frac{\alpha_1^2}{\alpha_1^2 + \sum_{j\neq 1} \alpha_j^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}}$$
$$\geq \frac{\alpha_1^2/\left\|x\right\|_2^2}{\alpha_1^2/\left\|x\right\|_2^2 + e^{-2kg}}$$

where we used that $\frac{\lambda_2}{\lambda_1} = 1 - g \leq e^{-g}$. With high probability in $1 - \frac{1}{d^c}$, we have $\left\|x\right\|_2^2 \leq 2d$ and $|\alpha_1| \geq \frac{1}{d^{O(c)}}$. In this case, we have $\alpha_1^2/\left\|x\right\|_2^2 \geq 1/d^{O(c)}$.

Using $k = \Omega\left(\frac{1}{g}\log\left(\frac{d}{\epsilon}\right)\right)$, we have $\langle v_1, u\rangle^2 \geq 1 - \epsilon$. Furthermore, this implies that

$$u^\top \mathbf{A} u = u^\top \left(\sum_{i\in[d]} \lambda_i v_i v_i^\top\right) u \geq \lambda_1 \langle v_1, u\rangle^2 \geq (1-\epsilon)\lambda_1.$$

■

Lemma 2.4.1 (Hessian Eigenvector Computation and Approximation) *For any $x \in \mathbb{R}^d$, $t > 0$, and $\epsilon \in (0, \frac{1}{4})$, The algorithm* `ApproxMinEig`$(x, t, \epsilon)$ *outputs (λ, u) in $O(nd\log\frac{d}{\epsilon})$ time with high probability in d such that $\langle v_t(x), u\rangle^2 \geq 1 - \epsilon$ if $\mu_t(x) \leq \frac{1}{4}t^2 w_t(x)$. Furthermore, if $\epsilon \leq \frac{\mu_t(x)}{4t^2\cdot w_t(x)} \leq \frac{1}{10^9 n^2 t^2 \cdot \widetilde{f}_*^2}$ and $\mathbf{Q} \overset{\text{def}}{=} t^2 \cdot w_t(x) - (t^2 \cdot w_t(x) - \lambda)\, uu^\top$ then $\frac{1}{4}\mathbf{Q} \preceq \nabla^2 f_t(x) \preceq 4\mathbf{Q}$.*

Proof By Lemma 2.3.4 we have that $\frac{1}{2}\mathbf{Z} \preceq \nabla^2 f_t(x) \preceq \mathbf{Z}$ for

$$\mathbf{Z} = t^2 \cdot w_t(x) - \left(t^2 \cdot w_t(x) - \mu_t(x)\right) v_t(x) v_t(x)^\top.$$

Consequently, if $\mu_t(x) \leq \frac{1}{4}t^2 w_t(x)$, then for all unit vectors $w \perp v_t(x)$, we have that

$$w^\top \nabla^2 f_t(x) w \geq \frac{1}{2} w^\top \mathbf{Z} w \geq \frac{1}{2} t^2 w_t(x).$$

Since $\bigtriangledown^2 f_t(x) = t^2 \cdot w_t(x) - \mathbf{A}$, this implies that $v_t(x)^\top \mathbf{A} v_t(x) \geq \frac{3}{4}t^2 \cdot w_t(x)$ and that $w^\top \mathbf{A} w \leq \frac{1}{2}t^2 w_t(x)$. Therefore, in this case, $\mathbf{A}$ has a constant multiplicative gap between its top two eigenvectors and by Theorem 2.B.1 we have that $\langle v_t(x), u\rangle^2 \geq 1 - \epsilon$.

For the second claim, we note that

$$t^2 \cdot w_t(x) - \mu_t(x) \geq u^\top \mathbf{A} u \geq (1-\epsilon)\lambda_1(\mathbf{A}) = (1-\epsilon)(t^2 \cdot w_t(x) - \mu_t(x))$$

Therefore, since $\lambda = u^\top \bigtriangledown^2 f_t(x) u = t^2 \cdot w_t(x) - u^\top \mathbf{A} u$, we have

$$(1-\epsilon)\mu_t(x) - \epsilon \cdot t^2 w_t(x) \leq \lambda \leq \mu_t(x). \tag{2.12}$$

On the other hand, by Lemma 2.D.2, we have that

$$\sqrt{\epsilon}\mathbf{I} \preceq v_t(x) v_t(x)^\top - uu^\top \preceq \sqrt{\epsilon}\mathbf{I}. \tag{2.13}$$

Consequently, using (2.12), (2.13), we have that $\frac{1}{2}\mathbf{Z} \preceq \mathbf{Q} \preceq 2\mathbf{Z}$ if $\epsilon \leq \frac{\mu_t(x)}{4t^2 \cdot w_t(x)}$ and $\frac{1}{4}\mathbf{Q} \preceq \bigtriangledown^2 f_t(x) \preceq 4\mathbf{Q}$ follows.

All that remains is to consider the case where $\mu_t(x) > \frac{1}{4}t^2 w_t(x)$. However, in this case $\frac{1}{4}t^2 \cdot w_t(x)\mathbf{I} \preceq \bigtriangledown^2 f_t(x) \preceq t^2 \cdot w_t(x)\mathbf{I}$ and clearly $\frac{1}{4}t^2 \cdot w_t(x)\mathbf{I} \preceq \mathbf{Q} \preceq t^2 \cdot w_t(x)\mathbf{I}$ again yielding $\frac{1}{4}\mathbf{Q} \preceq \bigtriangledown^2 f_t(x) \preceq 4\mathbf{Q}$.

To simplify the term $\frac{\mu_t(x)}{4t^2 \cdot w_t(x)}$, we apply Lemma 2.A.7 to show that

$$\frac{\mu_t(x)}{4t^2 \cdot w_t(x)} \geq \frac{1}{15000nt \cdot f(x^*)} \geq \frac{1}{15000nt \cdot \widetilde{f}_*}.$$

Therefore, ApproxMinEig(x, t, ϵ) works if $\epsilon \leq \frac{1}{10^9 n^2 t^2 \cdot \widetilde{f}_*^2}$.

■

Lemma 2.4.2 *Let* $u = \texttt{ApproxMinEig}(x, t, \epsilon_v)$ *for some* x *such that* $\left\|x - x_t\right\|_2 \leq \frac{\epsilon_c}{t}$ *for* $\epsilon_c \leq \frac{1}{10^6}$. *Suppose that* $\mu_t \leq \frac{1}{4}t^2 \cdot w_t$. *Then, for all unit vectors* $y \perp u$, *we have* $\langle y, v_t\rangle^2 \leq \max\{500\epsilon_c^{2/3}, 10\epsilon_v\}$.

Proof First, by Lemma 2.4.1 we know that $\langle v_t(x), u\rangle^2 \geq 1 - \epsilon_v$. By assumption, we have that $\left\|x - x_t\right\|_2 \leq \frac{\epsilon_c}{t}$. Therefore, by Lemma 2.3.1 we have that

$$(1 - 6\epsilon_c^{2/3}) \bigtriangledown^2 f_t(x_t) \preceq \bigtriangledown^2 f_t(x) \preceq (1 + 6\epsilon_c^{2/3}) \bigtriangledown^2 f_t^{(i)}(x_t).$$

If $\mu_t \leq \frac{1}{4}t^2 \cdot w_t$, we know that the largest eigenvalue of $\mathbf{A}$ defined in ApproxMinEig(x, t, ϵ) is at least $\frac{3}{4}t^2 \cdot w_t$ while the second largest eigenvalue is at most $\frac{1}{2}t^2 \cdot w_t$. Hence, the eigengap g defined in Lemma 2.D.3 is at least $\frac{1}{3}$ and hence

$$\langle v_t(x), v_t\rangle^2 \geq 1 - 54\epsilon_c^{2/3}.$$

Consequently, by Lemma 2.D.4, we have that $\langle u, v_t\rangle^2 \geq 1 - \max\{216\epsilon_c^{2/3}, 4\epsilon_v\}$.

To prove the final claim, we write $u = \alpha v_t + \beta w$ for an unit vector $w \perp v_t$. Since $y \perp u$, we have that $0 = \alpha\langle v_t, y\rangle + \beta\langle w, y\rangle$. Then, either $\langle v_t, y\rangle = 0$ and the result follows or $\alpha^2\langle v_t, y\rangle^2 = \beta^2\langle w, y\rangle^2$. Since $\alpha^2 + \beta^2 = 1$, we have

$$\langle v_t, y\rangle^2 \leq \frac{\beta^2\langle w, y\rangle^2}{\alpha^2} \leq \frac{1}{\alpha^2} - 1.$$

Noting that $\alpha^2 \geq 1 - \max\left(216\epsilon_c^{2/3}, 4\epsilon_v\right)$ then yields the claim.

■

Lemma 2.4.3 *Suppose that $\mu_t \geq \frac{1}{4}t^2 \cdot w_t$. Then, we have $\left\|x_s - x_t\right\|_2 \leq \frac{1}{100t}$ for all $s \in [1, 1.001t]$.*

Proof Let t' be the supremum over all s such that $\left\|x_s - x_t\right\|_2 \leq \frac{1}{100t}$ for all $s \in [t, t']$. Hence, for all $s \in [t, t']$, Lemma 2.3.1 shows that

$$\nabla^2 f_s(x_s) \succeq \frac{1}{2}\nabla^2 f_t(x_t) \succeq \frac{1}{8}t^2 \cdot w_t\mathbf{I}.$$

Now, we use Lemma 2.A.1 to show that

$$\begin{aligned}
\left\|x_{t'} - x_t\right\|_2 &= \int_t^{t'} \left\|\frac{d}{ds}x_s\right\|_2 d_s \\
&\leq \int_t^{t'} \left\|\left(\triangledown^2 f_s(x_s)\right)^{-1}\sum_{i\in[n]}\frac{s}{(1+g_s^{(i)})g_s^{(i)}}(x_s - a^{(i)})\right\|_2 d_s \\
&\leq \frac{8}{t^2 w_t}\int_t^{t'} \left\|\sum_{i\in[n]}\frac{s}{(1+g_s^{(i)})g_s^{(i)}}(x_s - a^{(i)})\right\|_2 d_s.
\end{aligned}$$

Now since clearly $s\left\|x_s - a^{(i)}\right\|_2 \leq g_s^{(i)}$, invoking Lemma 2.3.2 yields that

$$\left\|\sum_{i\in[n]}\frac{s}{(1+g_s^{(i)})g_s^{(i)}}(x_s - a^{(i)})\right\|_2 \leq \sum_{i\in[n]}\frac{1}{1+g_s^{(i)}} = w_s \leq \left(\frac{s}{t}\right)^2 w_t.$$

Therefore, we have

$$\begin{aligned}
\left\|x_{t'} - x_t\right\|_2 &\leq \frac{8}{t^2 w_t}\int_t^{t'}\left(\frac{s}{t}\right)^2 w_t d_s \\
&\leq \frac{8}{3t^4}\left(t'^3 - t^3\right)
\end{aligned}$$

Hence, if $t' < 1.001t$, then we have $\left\|x_{t'} - x_t\right\|_2 \leq \frac{1}{101t}$, which is a contradiction because $\left\|x_{t'} - x_t\right\|_2 = \frac{1}{100t}$. Therefore, $t' \geq 1.001t$ and $\left\|x_s - x_t\right\|_2 \leq \frac{1}{100t}$ for all $s \leq 1.001t$.

■

2.B.2 Line Searching

Lemma 2.4.4 *Given some $y \in \mathbb{R}^d$, $t > 0$ and $\epsilon > 0$. In $O(nd \log(\frac{nt \cdot \widetilde{f}_*}{\epsilon}))$ time with high probability, `LocalCenter`(y, t, ϵ) computes $x^{(k)}$ such that*

$$f_t(x^{(k)}) - \min_{\left\|x-y\right\|_2 \le \frac{1}{100t}} f_t(x) \le \epsilon \left(f_t(y) - \min_{\left\|x-y\right\|_2 \le \frac{1}{100t}} f_t(x) \right).$$

Proof Note that by Lemma 2.4.1 we have that $\frac{1}{4}\mathbf{Q} \preceq \bigtriangledown^2 f_t(y) \preceq 4\mathbf{Q}$. For any x such that $\left\|x-y\right\|_2 \le \frac{1}{100t}$ and hence Lemma 2.3.1 shows that

$$\frac{1}{2} \bigtriangledown^2 f_t(x) \preceq \bigtriangledown^2 f_t(y) \preceq 2 \bigtriangledown^2 f_t(x).$$

Hence, we have that

$$\frac{1}{8}\mathbf{Q} \preceq \bigtriangledown^2 f_t(x) \preceq 8\mathbf{Q}$$

for all $\left\|x-y\right\|_2 \le \frac{1}{100t}$. Therefore, Lemma 2.D.5 shows that

$$f_t(x^{(k)}) - \min_{\left\|x-y\right\|_2 \le \frac{1}{100t}} f_t(x) \le \left(1 - \frac{1}{64}\right)^k \left(f_t(x^{(0)}) - \min_{\left\|x-y\right\|_2 \le \frac{1}{100t}} f_t(x) \right).$$

The guarantee follows from our choice of k.

For the running time, Lemma 2.4.1 showed the cost of `ApproxMinEig` is $O\left(nd \log\left(nt \cdot \widetilde{f}_*\right)\right)$. It is easy to show that the cost per iteration is $O(nd)$ and therefore, the total cost of the k iterations is $O(nd \log(1/\epsilon))$.

■

Lemma 2.4.5 *Given x such that $\left\|x - x_t\right\|_2 \le \frac{\epsilon_c}{t}$ with $t \ge \frac{1}{400 f(x^*)}$ and $\epsilon_c \le \frac{1}{10^{15} n^3 t^3 \cdot \widetilde{f}_*^3}$. Let $u =$ `ApproxMinEig`(x, t, ϵ_v) with $\epsilon_v \le \frac{1}{10^8 n^2 t^2 \cdot \widetilde{f}_*^2}$. Then, in $O(nd \log^2(\frac{nt \cdot \widetilde{f}_*}{\varepsilon}))$ time, `LineSearch`(x, t, t', u, ϵ) output y such that $\left\|y - x_{t'}\right\|_2 \le \frac{\epsilon}{t'}$.*

Proof To use `OneDimMinimizer`, we need to prove the function g is nt-Lipschitz convex function such that $\min g = \min f_{t'}$ and the minimizer of g lies in $[-12\widetilde{f}_*, 12\widetilde{f}_*]$

(Lipschitzness) Note that the function $\phi_t(s) = \sqrt{1 + t^2 s^2} - \ln\left[1 + \sqrt{1 + t^2 s^2}\right]$ is a t-Lipschitz function. Hence, f_t is a nt-Lipschitz function. Therefore, g is a nt-Lipschitz function.

(Convexity) Consider the function

$$h(\alpha, x) = f_t(x) + \infty 1_{\left\|x-(y+\alpha v)\right\|_2 > \frac{1}{100t}}.$$

Note that h is convex and

$$g(\alpha) = \min_{\left\|x-y\right\|_2 \leq \frac{1}{100t}} f_t(x) = \min_x h(\alpha, x).$$

Hence by Lemma 2.D.6, $g(\alpha)$ is convex.

(Validity) If $\mu_t \leq \frac{1}{4}t^2 \cdot w_t$, using Lemma 2.4.2, $\epsilon_c \leq \frac{1}{10^{15}n^3t^3 \cdot f^3(x^*)}$ and $\epsilon_v \leq \frac{1}{10^8 n^2 t^2 \cdot f^2(x^*)}$ and shows that

$$\begin{aligned}\langle y, v_t\rangle^2 &\leq \max\left(500\epsilon_c^{2/3}, 10\epsilon_v\right)\\ &\leq \frac{1}{10^7 n^2 t^2 \cdot f^2(x^*)}\end{aligned}$$

for all unit vectors $y \perp u$. Lemma 2.A.7 shows that $t^2\kappa \leq 3100nt \cdot f(x^*)$ and hence for such y, we have

$$\langle y, v_t\rangle^2 \leq \frac{1}{t^4 \cdot \kappa^2}.$$

Therefore, we can apply Lemma 2.3.5 to show that

$$y^\top(x_{t'} - x_t) \leq \frac{1}{100t}.$$

In particular, we let α_* be the minimizer of $\left\|x - \alpha^* u - x_{t'}\right\|_2$ and consider $y = x - x_{t'} - \alpha^* u$. Since $y \perp u$, this shows that $\left\|x - \alpha^* u - x_{t'}\right\|_2 \leq \frac{1}{100t}$.

If $\mu_t \geq \frac{1}{4}t^2 \cdot w_t$, then Lemma 2.4.3 shows that $\left\|x - \alpha^* u - x_{t'}\right\|_2 \leq \frac{1}{100t}$ with $\alpha^* = 0$.

Consequently, in both case, we have $\min_\alpha g(\alpha) \leq g(\alpha^*) = f_{t'}(x_{t'})$.

(Domain Size) Lemma 2.A.5 showed that $\left\|x_{t'} - x^*\right\| \leq f(x^*)$ and $\left\|x_t - x^*\right\| \leq f(x^*)$. Since $t \geq \frac{1}{400f(x^*)}$ and $\left\|x - x_t\right\|_2 \leq \frac{1}{100t}$, we have

$$\left\|x - x_{t'}\right\|_2 \leq 12f(x^*) \leq 12\widetilde{f}_*.$$

Therefore, we know the minimum $\alpha^* \in [-12\widetilde{f}_*, 12\widetilde{f}_*]$.

Now, we can use invoke Lemma 2.D.8 and shows that the function `OneDimMinimizer` outputs α such that $g(\alpha) \leq \min g(\alpha) + 4\epsilon_O$. By the guarantee of `LocalCenter`, we find a point z such that

$$f_{t'}(z) \leq f_{t'}(x_{t'}) + 5\epsilon_O\left(f_{t'}(x + \alpha v) - f_{t'}(x_{t'})\right).$$

Since f_t is a nt-Lipschitz function and $\left\|x - \alpha^* u - x_{t'}\right\|_2 \le \frac{1}{100t}$, we apply Lemma 2.A.8 and get that

$$\begin{aligned} f_{t'}(x + \alpha v) &\le f_{t'}(x + \alpha^* v) + 12nt' \cdot \widetilde{f}_* \\ &\le f_{t'}(x_{t'}) + \frac{n}{100^2} + 12nt' \cdot \widetilde{f}_* \\ &\le f_{t'}(x_{t'}) + 13nt' \cdot \widetilde{f}_*. \end{aligned}$$

Therefore, we have

$$f_{t'}(z) \le f_{t'}(x_{t'}) + 100nt \cdot \widetilde{f}_* \cdot \epsilon_O.$$

Lemma 2.A.8 shows that

$$f_{t'}(z) \ge f_{t'}(x_{t'}) + \frac{1}{10^8 n^2 \cdot \widetilde{f}_*^2}\left\|z - x_{t'}\right\|_2^2.$$

Therefore, we have that

$$\frac{1}{10^8 n^2 \cdot \widetilde{f}_*^2}\left\|z - x_{t'}\right\|_2^2 \le 100nt \cdot \widetilde{f}_* \cdot \epsilon_O.$$

Hence, we have $\left\|z - x_{t'}\right\|_2^2 \le 10^{10} n^3 t \cdot \widetilde{f}_*^3 \cdot \epsilon_O$. Since $\epsilon_O = \frac{\epsilon^2}{10^{10} t^3 n^3 \cdot \widetilde{f}_*^3}$, we have that `LocalCenter` works as desired.

All that remains is to bound the running time. We have by Lemma 2.D.8 that we call q only $O\left(\log \frac{tn\widetilde{f}_*}{\epsilon_O}\right) = O\left(\log \frac{tn \cdot \widetilde{f}_*}{\epsilon}\right)$ times. So all that is left is to bound the cost of q, i.e. each `LocalCenter` computation. By Lemma 2.4.4 the cost of each of these is $O\left(nd \log\left(\frac{nt \cdot \widetilde{f}_*}{\epsilon}\right)\right)$.Therefore, the total running time is $O\left(nd \log^2\left(\frac{nt \cdot \widetilde{f}_*}{\epsilon}\right)\right)$.

■

Lemma 2.4.6 *Given x such that $\left\|x - x_t\right\|_2 \le \frac{1}{100t}$ with $t \ge \frac{1}{400 f(x^*)}$. Then, in $O(nd \log^2(\frac{nt \cdot \widetilde{f}_*}{\varepsilon}))$ time, `LineSearch`(x, t, t, u, ϵ) output y such that $\left\|y - x_t\right\|_2 \le \frac{\epsilon}{t}$ for any vector u.*

Proof The proof is the same as that of Lemma 2.4.5, except for the fact that $\left\|x - \alpha^* u - x_t\right\|_2 \le \frac{1}{100t}$ is satisfied automatically for $\alpha^* = 0$. Note that this lemma places weaker conditions on the initial point.

■

2.B.3 Putting It All Together

Theorem 2.4.7 *In $O(nd \log^3(\frac{n}{\epsilon}))$ time, Algorithm 1 outputs an $(1 + \epsilon)$-approximate geometric median with constant probability.*

Proof Theorem 2.5.3 shows that $x^{(0)}$ is a 2 approximate median with constant probability. Now, Lemma 2.A.5 shows that

$$\left\|x^{(0)} - x_t\right\|_2 \leq f(x^{(0)}) = \widetilde{f}_*$$

for all $t > 0$. In particular for $t_1 = \frac{1}{400\widetilde{f}_*}$, we have that $\left\|x^{(0)} - x_{t_1}\right\|_2 \leq \frac{1}{100t_1}$. Hence, by Lemma 2.4.6, we have $\left\|x^{(1)} - x_{t_1}\right\|_2 \leq \frac{\epsilon_c}{t_1}$. Now, we can use Lemma 2.4.5 to prove $\left\|x^{(k)} - x_{t_k}\right\|_2 \leq \frac{\epsilon_c}{t_k}$ for all k.

Lemma 2.3.6 shows that

$$f(x_{t_k}) - f(x^*) \leq 100n\sqrt{\frac{f(x^*)}{t_k}}$$

and since $\left\|x^{(k)} - x_{t_k}\right\|_2 \leq \frac{1}{t}$, we have $f(x^{(k)}) - f(x^*) \leq 10Givenx0n\sqrt{\frac{f(x^*)}{t_k}} + \frac{n}{t_k}$. Since $t_k = \frac{1}{400\widetilde{f}_*}(1 + \frac{1}{600})^{k-1}$, we have

$$f(x^{(k)}) - f(x^*) \leq 3000n\widetilde{f}_*(1 + \frac{1}{600})^{-\frac{k-1}{2}}.$$

By our choice of k, we have $f(x^{(k)}) - f(x^*) \leq \frac{\varepsilon}{2}\widetilde{f}_* \leq \varepsilon f(x^*)$.

For the running time, Lemma 2.4.1 shows `ApproxMinEvec` takes $O(nd\log(n/\epsilon_v))$ per iteration and Lemma 2.4.5 shows `LineSearch` takes $O\left(nd\log^2\left(\frac{nt\cdot\widetilde{f}_*}{\epsilon_c}\right)\right)$ time per iteration. Since both ϵ_v and ϵ_c are $\Omega\left(\frac{1}{n^{O(1)}(t\cdot\widetilde{f}_*)^{O(1)}}\right)$. Since $t\widetilde{f}_* = O(\frac{n}{\varepsilon})$, we have that both subroutines take $O\left(nd\log^2\left(\frac{n}{\epsilon}\right)\right)$ time. Since there are $O(\log(\frac{n}{\varepsilon}))$ iterations, the total running time is $O(nd\log^3(\frac{n}{\epsilon}))$.

■

2.C Derivation of Penalty Function

Here we derive our penalized objective function. Consider the following optimization problem:

$$\min_{x\in\mathbb{R}^d,\alpha\geq 0\in\mathbb{R}^n} f_t(x,\alpha) \quad \text{where} \quad t\cdot 1^T\alpha + \sum_{i\in[n]} -\ln\left(\alpha_i^2 - \left\|x - a^{(i)}\right\|_2^2\right) \ .$$

As, $-\ln\left(\alpha_i^2 - \left\|x - a^{(i)}\right\|_2^2\right)$ is a natural barrier function for the set $\alpha_i^2 \geq \left\|x - a^{(i)}\right\|_2^2$ we see that as we minimize this function for increasing values of t the x values converge to a solution to the geometric median problem.

The penalty function we use $f_t(x)$ is simply this function after we solve for the optimal α_i explicitly. To see this fix and x and t. Note that for all $j \in [n]$ we have

$$\frac{\partial}{\partial\alpha_j} f(x,\alpha) = t - \left(\frac{1}{\alpha_j^2 - \left\|x - a^{(i)}\right\|_2^2}\right) 2\alpha_j$$

and therefore, as $f(x,\alpha)$ is convex in α, the minimum α_j^* must satisfy

$$t\left(\left(\alpha_j^*\right)^2 - \left\|x - a^{(i)}\right\|_2^2\right) - 2\alpha_j^* = 0 \ .$$

Solving for such α_j^* we get

$$\alpha_j^* = \frac{2 + \sqrt{4 + 4t^2\left\|x - a^{(i)}\right\|_2^2}}{2t} = \frac{1}{t}\left[1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}\right] \ .$$

and consequently

$$\begin{aligned}\left(\alpha_j^*\right)^2 &= \frac{1}{t^2}\left[1 + 2\sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2} + 1 + t^2\left\|x - a^{(i)}\right\|_2^2\right] \\ &= \frac{2}{t^2}\left[1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}\right] + \left\|x - a^{(i)}\right\|_2^2 \ .\end{aligned}$$

Thus we can write the problem as

$$\min_{x\in\mathbb{R}^d} \sum_{i\in[n]} \left[1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2} - \ln\left[\frac{2}{t^2}\left(1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}\right)\right]\right] \ .$$

If we drop the constants, we get our penalty function f_t :

$$\min_{x\in\mathbb{R}^d} \sum_{i\in[n]} \left[\sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2} - \ln\left(1 + \sqrt{1 + t^2\left\|x - a^{(i)}\right\|_2^2}\right)\right] \ .$$

2.D Technical Facts

Here we provide various technical lemmas we use through the chapter.

2.D.1 General Math

The following lemma shows that given any matrix obtained as a convex combination of the identity minus a rank 1 matrix less than the identity results in a matrix that is well approximation spectrally by the identity minus a rank 1 matrix. We use this Lemma to characterize the Hessian of our penalized objective function and thereby imply that it is possible to apply the inverse of the Hessian to a vector with high precision.

Lemma 2.D.1 *Given any matrix* $\mathbf{A} = \sum_i \left(\alpha_i \mathbf{I} - \beta_i a_i a_i^\top\right) \in \mathbb{R}^{d\times d}$ *where* a_i *are unit vectors and* $0 \le \beta_i \le \alpha_i$ *for all* i*. We have that*

$$\frac{1}{2}\left(\sum_i \alpha_i \mathbf{I} - \lambda v v^\top\right) \preceq \mathbf{A} \preceq \sum_i \alpha_i \mathbf{I} - \lambda v v^\top$$

where v is a unit vector that is the maximum eigenvector of $\sum_i \beta_i a_i a_i^\top$ and λ is the corresponding eigenvalue.

Proof Let $\lambda_1 \geq ... \geq \lambda_d$ denote the eigenvalues of $\sum_i \beta_i a_i a_i^\top$. Clearly $\mathrm{tr}(\mathbf{A}) = \sum_i \lambda_i = d\sum_i \alpha_i - \sum_i \beta_i \geq (d-1)\sum_i \alpha_i$. Also clearly, $v^\top \mathbf{A} v = v^\top \left(\sum_i \alpha_i \mathbf{I} - \lambda v v^\top\right) v$, and therefore it simply remains to show that $\lambda_j \in [\frac{1}{2}\sum_i \alpha_i, \sum_i \alpha_i]$ for all $j > 1$. Since, $\lambda_j \leq \sum_i \alpha_i$ for all j, we have that

$$(d-1)\sum_i \alpha_i \leq \mathrm{tr}(\mathbf{A}) = \sum_i \lambda_i < 2\lambda_2 + \sum_{j \geq 3} \lambda_i \leq 2\lambda_2 + (d-1)\sum_i \alpha_i.$$

Therefore, $\lambda_j \geq \lambda_2 \geq \frac{1}{2}\sum_i \alpha_i$ for all $j > 1$.

■

Next we show how to bound the difference between the outer product of two unit vectors by their inner product. We use this lemma to bound the amount of precision required in our eigenvector computations.

Lemma 2.D.2 *For unit vectors u_1 and u_2 we have*

$$\left\|u_1 u_1^\top - u_2 u_2^\top\right\|_2^2 = 1 - (u_1^\top u_2)^2 \tag{2.14}$$

Consequently if $\left(u_1^\top u_2\right)^2 \geq 1 - \epsilon$ for $\epsilon \leq 1$ we have that

$$-\sqrt{\epsilon}\mathbf{I} \preceq u_1 u_1^\top - u_2 u_2^\top \preceq \sqrt{\epsilon}\mathbf{I}$$

Proof Note that $u_1 u_1^\top - u_2 u_2^\top$ is a symmetric matrix and all eigenvectors are either orthogonal to both u_1 and u_2 (with eigenvalue 0) or are of the form $v = \alpha u_1 + \beta u_2$ where α and β are real numbers that are not both 0. Thus, if v is an eigenvector of eigenvalue λ it must be that

$$\begin{aligned} \gamma\left(\alpha u_1 + \beta u_2\right) &= \left(u_1 u_1^\top - u_2 u_2^\top\right)\left(\alpha u_1 + \beta u_2\right) \\ &= (\alpha + \beta(u_1^\top u_2))u_1 - (\alpha(u_1^\top u_2) + \beta)u_2 \end{aligned}$$

Consequently

$$\begin{pmatrix} (1-\gamma) & u_1^\top u_2 \\ -(u_1^\top u_2) & -(1+\gamma) \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

By computing the determinant we see this has a solution only when

$$-(1-\gamma^2) + (u_1^\top u_2)^2 = 0$$

Solving for γ then yields (2.14) and completes the proof.

■

Next we show how the top eigenvectors of two spectrally similar matrices are related. We use this to bound the amount of spectral approximation we need to obtain accurate eigenvector approximations.

Lemma 2.D.3 *Let* $\mathbf{A}$ *be a PSD matrix such that* $(1-\epsilon)\mathbf{A} \preceq \mathbf{B} \preceq (1+\epsilon)\mathbf{A}$. *Let* $g \overset{\text{def}}{=} \frac{\lambda_1(\mathbf{A})-\lambda_2(\mathbf{A})}{\lambda_1(\mathbf{A})}$ *where* $\lambda_1(\mathbf{A})$ *and* $\lambda_2(\mathbf{A})$ *are the largest and second largest eigenvalue of* $\mathbf{A}$. *Then, we have*

$$\left(v_1(\mathbf{A})^\top v_1(\mathbf{B})\right)^2 \geq 1 - 3\frac{\epsilon}{g}.$$

Proof Without loss of generality $v_1(\mathbf{B}) = \alpha v_1(\mathbf{A}) + \beta v$ for some unit vector $v \perp v_1(\mathbf{A})$ and $\alpha, \beta \in \mathbb{R}$ such that $\alpha^2 + \beta^2 = 1$. Now we know that

$$v_1(\mathbf{B})^\top \mathbf{B} v_1(\mathbf{B}) \leq (1+\epsilon) v_1(\mathbf{B})^\top \mathbf{A} v_1(\mathbf{B}) \leq (1+\epsilon)\left[\alpha^2 \lambda_1(\mathbf{A}) + \beta^2 \lambda_2(\mathbf{A})\right]$$

Furthermore, by the optimality of $v_1(\mathbf{B})$ we have that

$$v_1(\mathbf{B})^\top \mathbf{B} v_1(\mathbf{B}) \geq (1-\epsilon) v_1(\mathbf{A})^\top \mathbf{A} v_1(\mathbf{A}) \geq (1-\epsilon)\lambda_1(\mathbf{A}).$$

Now since $\beta^2 = 1 - \alpha^2$ we have that

$$(1-\epsilon)\lambda_1(\mathbf{A}) \leq (1+\epsilon)\alpha^2 \left(\lambda_1(\mathbf{A}) - \lambda_2(\mathbf{A}) + (1+\epsilon)\lambda_2(\mathbf{A})\right.$$

Thus we have that

$$\alpha^2 \geq \frac{\lambda_1(\mathbf{A}) - \lambda_2(\mathbf{A}) - \epsilon(\lambda_1(\mathbf{A}) + \lambda_2(\mathbf{A}))}{(1+\epsilon)(\lambda_1(\mathbf{A}) - \lambda_2(\mathbf{A}))}$$

Since $\frac{1}{1+\epsilon} \geq 1-\epsilon$, $\lambda_2(\mathbf{A}) \leq \lambda_1(\mathbf{A})$, and $g \leq 1$ we have

$$\alpha^2 \geq \left(1 - 2\epsilon\left(\frac{\lambda_1(\mathbf{A})}{\lambda_1(\mathbf{A}) - \lambda_2(\mathbf{A})}\right)\right)(1-\epsilon) = 1 - 2\frac{\epsilon}{g} - \epsilon + 2\frac{\epsilon^2}{g} \geq 1 - 3\frac{\epsilon}{g}.$$

■

Here we prove a an approximate transitivity lemma for inner products of vectors. We use this to bound the accuracy need for certain eigenvector computations.

Lemma 2.D.4 *Suppose that we have vectors* $v_1, v_2, v_3 \in \mathbb{R}^n$ *such that* $\langle v_1, v_2\rangle^2 \geq 1-\epsilon$ *and* $\langle v_2, v_3\rangle^2 \geq 1-\epsilon$ *for* $0 < \epsilon \leq \frac{1}{4}$ *then* $\langle v_1, v_3\rangle^2 \geq 1 - 4\epsilon$.

Proof Without loss of generality, we can write $v_1 = \alpha_1 v_2 + \beta_1 w_1$ for $\alpha_1^2 + \beta_1^2 = 1$ and unit vector $w_1 \perp v_2$. Similarly we can write $v_3 = \alpha_3 v_2 + \beta_3 w_3$ for $\alpha_3^2 + \beta_3^2 = 1$ and unit vector $w_3 \perp v_2$.

Now, by the inner products we know that $\alpha_1^2 \geq 1 - \epsilon$ and $\alpha_3^2 \geq 1 - \epsilon$ and therefore $|\beta_1| \leq \sqrt{\epsilon}$ and $|\beta_3| \leq \sqrt{\epsilon}$. Consequently, since $\epsilon \leq \frac{1}{4}$, we have that

$$\begin{aligned}\langle v_1, v_3 \rangle^2 &\geq \langle \alpha_1 v_2 + \beta_1 w_1, \alpha_3 v_2 + \beta_3 w_3 \rangle^2 \geq (\alpha_1 \alpha_3 - |\beta_1 \beta_3|)^2 \\ &\geq (1 - \epsilon - \epsilon)^2 \geq (1 - 2\epsilon)^2 \geq 1 - 4\epsilon.\end{aligned}$$

■

2.D.2 First Order Optimization

Here we provide a single general lemma about first order methods for convex optimization. We use this lemma for multiple purposes including bounding errors and quickly compute approximations to the central path.

Lemma 2.D.5 *[Nes03]Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function that obtains its minimum value at a point x_* contained in some convex set $B \subseteq \mathbb{R}$. Further suppose that for some symmetric positive definite matrix $\mathbf{H} \in \mathbb{R}^{n \times n}$ we have that for all $x \in B$*

$$\mu \mathbf{H} \preceq \bigtriangledown^2 f(y) \preceq L \mathbf{H} .$$

Then for all $x \in B$ we have

$$\frac{1}{2L} \| \bigtriangledown f(x) \|_{\mathbf{H}^{-1}}^2 \leq f(x) - f(x_*) \leq \frac{L}{2} \| x - x_* \|_{\mathbf{H}}^2$$

and

$$\frac{\mu}{2} \| x - x_* \|_{\mathbf{H}}^2 \leq f(x) - f(x_*) \leq \frac{1}{2\mu} \| \bigtriangledown f(x) \|_{\mathbf{H}^{-1}}^2$$

Furthermore, if

$$x_1 = \text{argmin}_{x \in B} \left[f(x_0) + \langle \bigtriangledown f(x_0), x - x_0 \rangle + \frac{L}{2} \| x_0 - x \|_{\mathbf{H}}^2 \right]$$

then

$$f(x_1) - f(x_*) \leq \left(1 - \frac{\mu}{L} \right) (f(x_0) - f(x_*)) \ . \tag{2.15}$$

Next we provide a short technical lemma about the convexity of functions that arises naturally in our line searching procedure.

Lemma 2.D.6 *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a convex function and and let $g(\alpha) \stackrel{\text{def}}{=} \min_{x \in S} f(x + \alpha d)$ for any convex set S and $d \in \mathbb{R}^n$. Then g is convex.*

Proof Let $\alpha, \beta \in \mathbb{R}$ and define $x_\alpha = \operatorname{argmin}_{x \in S} f(x + \alpha d)$ and $x_\beta = \operatorname{argmin}_{x \in S} f(x + \beta d)$. For any $t \in [0, 1]$ we have

$$\begin{aligned} g\left(t\alpha + (1-t)\beta\right) &= \min_{x \in S} f\left(x + (t\alpha + (1-t)\beta\right) \\ &\leq f(tx_\alpha + (1-t)x_\beta + (t\alpha + (1-t)\beta)d) && \text{(Convexity of } S\text{)} \\ &\leq t \cdot f(x_\alpha + \alpha d) + (1-t) \cdot f(x_\beta + \beta \cdot d) && \text{(Convexity of } f\text{)} \\ &= t \cdot g(\alpha) + (1-t) \cdot g(\beta) \end{aligned}$$

■

Lemma 2.D.7 *For any vectors $y, z, v \in \mathbb{R}^d$ and scalar α, we can minimize $\min_{\left\|x-y\right\|_2^2 \leq \alpha} \left\|x - z\right\|_{I-vv^\top}^2$ exactly in time $O(d)$.*

Proof Let x^* be the solution of this problem. If $\left\|x^* - y\right\|_2 < \alpha$, then $x^* = z$. Otherwise, there is $\lambda > 0$ such that x^* is the minimizer of

$$\min_x \left\|x - z\right\|_{I-vv^\top}^2 + \lambda \left\|x - y\right\|^2.$$

Let $Q = I - vv^\top$. Then, the optimality condition shows that

$$Q(x - z) + \lambda(x - y) = 0.$$

Therefore, we have

$$x = (Q + \lambda I)^{-1}(Qz + \lambda y). \tag{2.16}$$

Hence, we have

$$\alpha = \left\|x - y\right\|_2^2 = (z - y)^\top Q(Q + \lambda I)^{-2} Q(z - y).$$

Let $\eta = 1 + \lambda$, then we have $(Q + \lambda I) = \eta I - vv^\top$and hence Sherman–Morrison formula shows that

$$\begin{aligned} (Q + \lambda I)^{-1} &= \eta^{-1} I + \frac{\eta^{-2} vv^\top}{1 - \left\|v\right\|^2 \eta^{-1}} \\ &= \eta^{-1}\left(I + \frac{vv^\top}{\eta - \left\|v\right\|^2}\right). \end{aligned}$$

Hence, we have

$$\begin{aligned}(Q+\lambda I)^{-2} &= \eta^{-2}\left(I+\frac{2vv^{\top}}{\eta-\left\|v\right\|^{2}}+\frac{vv^{\top}\left\|v\right\|^{2}}{\left(\eta-\left\|v\right\|^{2}\right)^{2}}\right)\\ &= \eta^{-2}\left(I+\frac{2\eta-\left\|v\right\|^{2}}{\left(\eta-\left\|v\right\|^{2}\right)^{2}}vv^{\top}\right)\end{aligned}$$

Let $c_1=\left\|Q(z-y)\right\|_2^2$ and $c_2=\left(v^{\top}Q(z-y)\right)^2$, then we have

$$\alpha = \eta^{-2}\left(c_1+\frac{2\eta-\left\|v\right\|^{2}}{\left(\eta-\left\|v\right\|^{2}\right)^{2}}c_2\right).$$

Hence, we have

$$\alpha\eta^2\left(\eta-\left\|v\right\|^{2}\right)^{2}=c_1\left(\eta-\left\|v\right\|^{2}\right)^{2}+c_2\left(2\eta-\left\|v\right\|^{2}\right).$$

Note that this is a polynomial of degree 4 in η and all coefficients can be computed in $O(d)$ time. Solving this by explicit formula, one can test all 4 possible η's into the formula (2.16) of x. Together with trivial case $x^*=z$, we simply need to check among 5 cases to check which is the solution.

■

2.D.3 Noisy One Dimensional Convex Optimization

Here we show how to minimize an one dimensional convex function giving a noisy oracle for evaluating the function. While this could possibly be done using general results on convex optimization using a membership oracle, the proof in one dimension is much simpler and we include it here for completeness .

Lemma 2.D.8 *Given a L-Lipschitz convex function defined in $[\ell,u]$ interval. Suppose that we have an oracle g such that $|g(y)-f(y)|\leq\epsilon$ for all y. In $O(\log(\frac{L(u-\ell)}{\epsilon}))$ time and and with $O(\log(\frac{L(u-\ell)}{\epsilon}))$ calls to g, the algorithm $\texttt{OneDimMinimizer}(\ell,u,\epsilon,g,L)$ outputs a point x such that*

$$f(x)-\min_{y\in[\ell,u]}f(y)\leq 4\epsilon.$$

Proof First, note that for any y, y', if $f(y)<f(y')-2\epsilon$,$g(y)<g(y')$. This directly follows from the error guarantee on g.

Now, the output of the algorithm, x, is simply the point queried by the algorithm (i.e. ℓ and the z_ℓ^i and z_u^i) with the smallest value of g. This implies that $f(x)$ is within 2ϵ of the minimum value of

```
Algorithm 8: OneDimMinimizer($\ell, u, \epsilon, g, L$)
  Input: Interval $[\ell, u] \subseteq \mathbb{R}$, target additive error $\epsilon$, noisy additive evaluation oracle $g$, Lipschitz
  bound $L$
  Let $x^{(0)} = \ell$, $y_\ell^{(0)} = \ell$, $y_u^{(0)} = u$
  for $i = 1, ..., \left\lceil \log_{3/2}(\frac{L(u-\ell)}{\epsilon}) \right\rceil$ do
      Let $z_\ell^{(i)} = \frac{2y_\ell^{(i-1)} + y_u^{(i-1)}}{3}$, $z_u^{(i)} = \frac{y_\ell^{(i-1)} + 2y_u^{(i-1)}}{3}$...
      if $g(z_\ell^{(i)}) \leq g(z_u^{(i)})$ then
          Let $(y_\ell^{(i)}, y_u^{(i)}) = (y_\ell^{(i-1)}, z_u^{(i)})$.
          Let $x^{(i)} = z_\ell^{(i)}$ if $g(z_\ell^{(i)}) \leq g(x^{(i-1)})$ and $x^{(i-1)}$ otherwise.
      if $g(z_\ell^{(i)}) > g(z_u^{(i)})$ then
          Let $(y_\ell^{(i)}, y_u^{(i)}) = (z_\ell^{(i)}, y_u^{(i-1)})$.
          Let $x^{(i)} = z_u^{(i)}$ if $g(z_u^{(i)}) \leq g(x^{(i-1)})$ and $x^{(i-1)}$ otherwise.
  end
  Output: $x$
```

f among the points queried. It thus suffices to show that the algorithm queries some point within 2ϵ of optimal.

To do this, we break into two cases. One is where the intervals $[y_\ell^i, y_u^i]$ all contain an optimum of f. In this case, the final interval contains an optimum, and is of size at most $\frac{\epsilon}{L}$. Thus, by the Lipschitz property, all points in the interval are within $\epsilon \leq 2\epsilon$ of optimal, and at least one endpoint of the interval must have been queried by the algorithm.

For the other case, we consider the last i for which this interval does contain an optimum of f. This means that $g(z_\ell^{(i)}) \leq g(z_u^{(i)})$ while the optimum x^* is to the right of $z_u^{(i)}$, or the symmetric case with the optimum to the left of $g(z_\ell^{(i)})$. Without loss of generality, we will assume the former. We then have $z_\ell^{(i)} \leq z_u^{(i)} \leq x^*$, while $x^* - z_u^{(i)} \leq z_u^{(i)} - z_\ell^{(i)}$. By the convexity of f, we therefore have that $f(z_u^{(i)}) - f(x^*) \leq f(z_\ell^{(i)}) - f(z_u^{(i)})$. But $f(z_\ell^{(i)}) - f(z_u^{(i)}) \leq 2\epsilon$ since $g(z_\ell^{(i)}) \leq g(z_u^{(i)})$. Thus, $f(z_u^{(i)}) - f(x^*) \leq 2\epsilon$, and $z_u^{(i)}$ is queried by the algorithm, as desired.

■

Chapter 3

Online Row Sampling

Finding a small spectral approximation for a tall $n \times d$ matrix $\mathbf{A}$ is a fundamental numerical primitive. For a number of reasons, one often seeks an approximation whose rows are sampled from those of $\mathbf{A}$. Row sampling improves interpretability, saves space when $\mathbf{A}$ is sparse, and preserves row structure, which is especially important, for example, when $\mathbf{A}$ represents a graph.

However, correctly sampling rows from $\mathbf{A}$ can be costly when the matrix is large and cannot be stored and processed in memory. Hence, a number of recent publications focus on row sampling in the streaming setting, using little more space than what is required to store the outputted approximation [KL13, KLMS14].

Inspired by a growing body of work on online algorithms for machine learning and data analysis, we extend this work to a more restrictive *online* setting: we read rows of $\mathbf{A}$ one by one and immediately decide whether each row should be kept in the spectral approximation or discarded, without ever retracting these decisions. We present an extremely simple algorithm (Figure 3.1) that approximates $\mathbf{A}$ up to multiplicative error ϵ and additive error δ using $\mathcal{O}(d \log d \log(\epsilon \|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ online samples, with memory overhead proportional to the cost of storing the spectral approximation. We also present an algorithm that uses $\mathcal{O}(d^2)$ memory but only requires $\mathcal{O}(d \log(\epsilon \|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ samples, which we show is optimal.

Our methods are clean and intuitive, allow for lower memory usage than prior work, and expose new theoretical properties of leverage score based matrix approximation.

The results presented in this chapter are joint work with Michael Cohen and Cameron Musco [CMP15].

3.1 Introduction

3.1.1 Background

A spectral approximation to a tall $n \times d$ matrix $\mathbf{A}$ is a smaller, typically $\tilde{\mathcal{O}}(d) \times d$ matrix $\tilde{\mathbf{A}}$ such that $\|\tilde{\mathbf{A}}\mathbf{x}\|_2 \approx \|\mathbf{A}\mathbf{x}\|_2$ for all $\mathbf{x}$. Typically one asks for a multiplicative approximation, which guarantees that $(1-\epsilon)\|\mathbf{A}\mathbf{x}\|_2^2 \leq \|\tilde{\mathbf{A}}\mathbf{x}\|_2^2 \leq (1+\epsilon)\|\mathbf{A}\mathbf{x}\|_2^2$. In other notation, $(1-\epsilon)\mathbf{A} \preceq \tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}$.

Such approximations have many applications, most notably for solving least squares regression over $\mathbf{A}$. If $\mathbf{A}$ is the vertex edge incidence matrix of a graph, $\tilde{\mathbf{A}}$ is a *spectral sparsifier* [ST04b]. It can be used to approximate effective resistances, spectral clustering, mixing time and random walk properties, and many other computations.

A number of recent papers focus on fast algorithms for spectral approximation. Using sparse random subspace embeddings [CW13, NN13, MM13], it is possible to find $\tilde{\mathbf{A}}$ in input sparsity time, essentially by randomly recombining the rows of $\mathbf{A}$ into a smaller number of rows. In some cases these embeddings are not very useful, as it is desirable for the rows of $\tilde{\mathbf{A}}$ to be a subset of rows sampled from $\mathbf{A}$. If $\mathbf{A}$ is sparse, this ensures that $\tilde{\mathbf{A}}$ is also sparse. If $\mathbf{A}$ represents a graph, it ensures that $\tilde{\mathbf{A}}$ is also a graph, specifically a weighted subgraph of the original.

It is well known that sampling $\mathcal{O}(d \log d/\epsilon^2)$ rows of $\mathbf{A}$ with probabilities proportional to their *leverage scores* yields a $(1+\epsilon)$ multiplicative factor spectral approximation to $\mathbf{A}$. Further, this sampling can be done in input sparsity time, either using subspace embeddings to approximate leverage scores, or using iterative sampling techniques [LMP13], some that only work with subsampled versions of the original matrix [CLM$^+$15].

3.1.2 Streaming and Online Row Sampling

When $\mathbf{A}$ is very large, input sparsity runtimes are not enough – memory restrictions also become important. Hence, recent work has tackled row sampling in a streaming model of computation. [KL13] presents a simple algorithm for sampling rows from an insertion only stream, using space approximately proportional to the size of the final approximation. [KLMS14] gives a sparse-recovery based algorithm that works in dynamic streams with row insertions and deletions and also uses nearly optimal space. Unfortunately, in order to handle dynamic streams, the algorithm in [KLMS14] is complex, requires additional restrictions on the input matrix, and uses significantly suboptimal runtime to recover a spectral approximation from its low memory representation of the input stream.

In this work we initiate the study of row sampling in an *online setting*. As in an insertion only stream, we read rows of $\mathbf{A}$ one by one. However, upon seeing a row, we must immediately decide whether it should be kept in the spectral approximation or discarded, without ever retracting these decisions.

We present a similar algorithm to [KL13], however, since we never prune previously sampled rows, the probability of sampling a row only depends on whether previous rows in the stream were sampled. This limited dependency structure allows us to rigorously argue that a spectral approximation is obtained.

In addition to addressing gaps in the literature on streaming spectral approximation, our restricted model extends work on online algorithms for a variety of other machine learning and data analysis problems, including principal component analysis [BGKL15], clustering [LSS14], classification [BB05, CDK$^+$06], and regression [CDK$^+$06]. In practice, online algorithms are beneficial since they can be highly computationally and memory efficient. Further, they can be applied in scenarios in which data is produced in a continuous stream and intermediate results must be output as the stream is processed. Spectral approximation is a widely applicable primitive for approximate learning and computation, so studying its implementation through online row sampling is a natural direction.

3.1.3 Our Results

Our primary contribution is a very simple algorithm for leverage score sampling in an online manner. The main difficulty with row sampling using leverage scores is that leverage scores themselves are not easy to compute. They are given by $l_i = \mathbf{a}_i^T(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{a}_i$,[1] and so require solving systems in $\mathbf{A}^T\mathbf{A}$ if computed naively. This is not only expensive, but also impossible in an online setting, where we do not have access to all of $\mathbf{A}$.

A critical observation is that it always suffices to sample rows by overestimates of their true leverage scores. The number of rows that must be sampled is proportional to the sum of these overestimates. Since the leverage score of a row can only go up when we remove rows from the matrix, a simple way to obtain an overestimate is to compute leverage score using just a subset of the other rows of $\mathbf{A}$. That is, letting $\mathbf{A}_j$ contain just j of $\mathbf{A}$'s n rows, we can overestimate l_i by $\tilde{l}_i = \mathbf{a}_i^T(\mathbf{A}_j^T\mathbf{A}_j)^{-1}\mathbf{a}_i$

[CLM$^+$15] shows that if $\mathbf{A}_j$ is a subset of rows sampled uniformly at random, then the expected leverage score of $\mathbf{a}_i$ is d/j. This simple fact immediately gives a result for online sampling from a *randomly ordered stream*. If we compute the leverage score of the current row $\mathbf{a}_i$ against all previously seen rows (or some approximation to these rows), then the expected sum of our overestimates will be bounded by $d + d/2 + ... + ... + d/n = \mathcal{O}(d\log n)$. So, sampling $\mathcal{O}(d\log d\log n/\epsilon^2)$ rows will be enough obtain a $(1+\epsilon)$ multiplicative factor spectral approximation.

What if we cannot guarantee a randomly ordered input stream? Is there any hope of being able to compute good leverage score estimates in an online manner? Surprisingly the answer to this is yes - we can in fact run nearly the exact same algorithm and be guaranteed that the sum of estimated leverage scores is low, *regardless of stream order*. Roughly, each time we receive a row which has high leverage score with respect to the previous rows, it must compose a significant part of $\mathbf{A}$'s spectrum. If $\mathbf{A}$ does not continue to grow unboundedly, there simply cannot be too many of these significant rows.

Specifically, we show that if we sample by the *ridge leverage scores* [AM14] over all previously seen rows, which are the leverage scores computed over $\mathbf{A}_i^T\mathbf{A}_i + \lambda\mathbf{I}$ for some small regularizing factor λ, then with just $\mathcal{O}(d\log d\log(\epsilon\|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ samples we obtain a $(1+\epsilon)$ multiplicative, δ additive error spectral approximation. That is, with high probability we sample a matrix $\tilde{\mathbf{A}}$ with $(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I}$.

To gain intuition behind this bound, note that we can convert it into a multiplicative one by setting $\delta = \epsilon\sigma_{min}(\mathbf{A})^2$ (as long as we have some estimate of $\sigma_{min}(\mathbf{A})$). This setting of δ will require taking $\mathcal{O}(d\log d\log(\kappa(\mathbf{A}))/\epsilon^2)$ samples. If we have a polynomial bound on the condition number of $\mathbf{A}$, as we do, for instance, for graphs with polynomially bounded edges weights, this becomes $\mathcal{O}(d\log^2 d/\epsilon^2)$ – nearly matching the $\mathcal{O}(d\log d/\epsilon^2)$ achievable if sampling by true leverage scores.

[1]Throughout this chapter, $\mathbf{a}_i$ are the column vectors corresponding to the rows of $\mathbf{A}$.

Our online sampling algorithm is extremely simple. When each row comes in, we compute the online ridge leverage score, or an estimate of it, and then irrevocably either add the row to our approximation or remove it. As mentioned, it is similar in form to the streaming algorithm of [KL13], except that it does not require pruning previously sampled rows. This allows us to avoid difficult dependency issues. Additionally, without pruning, we do not even need to store all previously sampled rows. As long as we store a constant factor spectral approximation our previous samples, we can compute good approximations to the online ridge leverage scores. In this way, we can store just $\mathcal{O}(d \log d \log(\epsilon \|\mathbf{A}\|_2^2/\delta))$ rows in working memory ($\mathcal{O}(d \log^2 d)$ if we want a spectral graph sparsifier), filtering our input stream into an $\mathcal{O}(d \log d \log(\kappa(\mathbf{A}))/\epsilon^2)$ sized output stream. Note that this memory bound in fact *improves* as ϵ decreases, and regardless, can be significantly smaller than the output size of the algorithm.

In additional to our main sampling result, we use our bounds on online ridge leverage score approximations to show that an algorithm in the style of [BSS12] allows us to remove a $\log d$ factor and sample just $\mathcal{O}(d \log(\epsilon \|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ (Theorem 3.4.1). This algorithm is more complex and can require $\mathcal{O}(d^2)$ working memory. However, in Theorem 3.5.1 we show that it is asymptotically optimal. The $\log(\epsilon \|\mathbf{A}\|_2^2/\delta)$ factor is not an artifact of our analysis, but is truly the cost of restricting ourselves to online sampling. No algorithm can obtain a multiplicative $(1+\epsilon)$ additive δ spectral approximation taking fewer than $\Omega(d \log(\epsilon \|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ rows in an online manner.

3.2 Overview

Let $\mathbf{A}$ be an $n \times d$ matrix with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$. A natural approach to row sampling from $\mathbf{A}$ is picking an *a priori* probability with which each row is kept, and then deciding whether to keep each row independently. A common choice is for the sampling probabilities to be proportional to the *leverage scores* of the rows. The leverage score of the i-th row of $\mathbf{A}$ is defined to be

$$\mathbf{a}_i^T (\mathbf{A}^T \mathbf{A})^\dagger \mathbf{a}_i,$$

where the dagger symbol denotes the pseudoinverse. In this work, we will be interested in approximating $\mathbf{A}^T\mathbf{A}$ with some (very) small multiple of the identity added. Hence, we will be interested in the λ-*ridge leverage scores* [AM14]:

$$\mathbf{a}_i^T (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{a}_i,$$

for a parameter $\lambda > 0$.

In many applications, obtaining the (nearly) exact values of $\mathbf{a}_i^T (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{a}_i$ for sampling is difficult or outright impossible. A key idea is that as long as we have a sequence $l_1, \ldots, l_n$ of *overestimates* of the λ-ridge leverage scores, that is for $i = 1, \ldots, n$

$$l_i \geq \mathbf{a}_i^T (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{a}_i,$$

we can sample by these overestimates and obtain rigorous guarantees on the quality of the obtained spectral approximation. This notion is formalized in Theorem 3.2.1.

Theorem 3.2.1 *Let $\mathbf{A}$ be an $n \times d$ matrix with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$. Let $\epsilon \in (0,1), \delta > 0, \lambda := \delta/\epsilon, c := 8 \log d/\epsilon^2$. Assume we are given $l_1, \ldots, l_n$ such that for all $i = 1, \ldots, n$,*

$$l_i \geq \mathbf{a}_i^T(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{a}_i.$$

For $i = 1, \ldots, n$, let

$$p_i := \min(cl_i, 1).$$

Construct $\tilde{\mathbf{A}}$ by independently sampling each row $\mathbf{a}_i$ of $\mathbf{A}$ with probability p_i, and rescaling it by $1/\sqrt{p_i}$ if it is included in the sample. Then, with high probability,

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I},$$

and the number of rows in $\tilde{\mathbf{A}}$ is $\mathcal{O}\left(\left(\sum_{i=1}^n l_i\right) \log d/\epsilon^2\right)$.

Proof This sort of guarantee for leverage score sampling is well known. See for example Lemma 4 of [CLM+15]. If we sampled both the rows of $\mathbf{A}$ and the rows of $\sqrt{\lambda}\mathbf{I}$ with the leverage scores over $(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})$, we would have $(1-\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}) \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})$. However, we do not sample the rows of the identity. Since we could have sampled them each with probability 1, we can simply subtract $\lambda\mathbf{I} = (\delta/\epsilon)\mathbf{I}$ from the multiplicative bound and have: $(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I}$.

■

The idea of using overestimates of leverage scores to perform row sampling has been applied successfully to various problems (see e.g. [KMP10, CLM+15]). However, in these applications, access to the entire matrix is required beforehand. In the streaming and online settings, we have to rely on partial data to approximate the true leverage scores. The most natural idea is to just use the portion of the matrix seen thus far as an approximation to $\mathbf{A}$. This leads us to introduce the *online λ-ridge leverage scores*:

$$l_i := \min(\mathbf{a}_i^T(\mathbf{A}_{i-1}^T\mathbf{A}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i, 1),$$

where $\mathbf{A}_i$ $(i = 0, \ldots, n)$ is defined as the matrix consisting of the first i rows of $\mathbf{A}$[2].

Since clearly $\mathbf{A}_i^T\mathbf{A}_i \preceq \mathbf{A}^T\mathbf{A}$ for all i, it is not hard to see that l_i does overestimate the true λ-ridge leverage score for row $\mathbf{a}_i$. A more complex question, however, is establishing an upper bound on $\sum_{i=1}^n l_i$ so that we can bound the number of samples needed by Theorem 3.2.1.

[2]We note that the following, perhaps more natural, definition of online leverage scores would also be effective:

$$l_i' := \mathbf{a}_i^T(\mathbf{A}_i^T\mathbf{A}_i + \lambda\mathbf{I})^{-1}\mathbf{a}_i.$$

However, we opt to use the proposed scores l_i for simplicity.

A core result of this work, stated in Theorem 3.2.2, is establishing an upper bound on the sum of λ-ridge leverage scores; in fact, this bound is shown to be tight up to constants (Theorem 3.5.1) and is nearly-linear in most cases.

Theorem 3.2.2 *Let* $\mathbf{A}$ *be an* $n \times d$ *matrix with rows* $\mathbf{a}_1, \ldots, \mathbf{a}_n$*. Let* $\mathbf{A}_i$ *for* $i \in \{0, \ldots, n\}$ *be the matrix consisting of the first* i *rows of* $\mathbf{A}$*. For* $\lambda > 0$*, let*

$$l_i := \min(\mathbf{a}_i^T(\mathbf{A}_{i-1}^T\mathbf{A}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i, 1).$$

be the online λ*-ridge leverage score of the* i^{th} *row of* $\mathbf{A}$*. Then*

$$\sum_{i=1}^{n} l_i = \mathcal{O}(d \log(\|\mathbf{A}\|_2^2/\lambda)).$$

Theorems 3.2.1 and 3.2.2 suggest a simple algorithm for online row sampling: simply use the online λ-ridge leverage scores, for $\lambda := \delta/\epsilon$. This produces a spectral approximation with only $\mathcal{O}(d \log d \log(\epsilon\|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ rows in an online manner. Unfortunately, computing l_i exactly requires us to store *all* the rows we have seen in memory (or alternatively to store the sum of their outer products, $\mathbf{A}_i^T\mathbf{A}_i$). In many cases, such a requirement would defeat the purpose of streaming row sampling.

A natural idea is to use the sample we have kept thus far as an approximation to $\mathbf{A}_i$ when computing l_i. It turns out that the approximate online ridge leverage scores $\tilde{l}_i$ computed in this way will not necessarily be good approximations to l_i; however, we can still prove that they satisfy the requisite bounds and yield the same row sample size! We formalize these results in the algorithm ONLINE-SAMPLE (Figure 3.1) and Theorem 3.2.3.

Theorem 3.2.3 *Let* $\tilde{\mathbf{A}}$ *be the matrix returned by* ONLINE-SAMPLE$(\mathbf{A}, \epsilon, \delta)$*. Then, with high probability,*

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I},$$

and the number of rows in $\tilde{\mathbf{A}}$ *is* $\mathcal{O}(d \log d \log(\epsilon\|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$*.*

To save computation, we note that, with a small modification to our analysis, we can run ONLINE-SAMPLE with batch processing of rows. Specifically, say we start from the i^{th} position in the stream. we can store the next $b = \mathcal{O}(d)$ rows. We can then compute sampling probabilities for these rows all at once using a system solver for $(\tilde{\mathbf{A}}_{i+b}^T\tilde{\mathbf{A}}_{i+b} + \lambda\mathbf{I})$. Using a trick introduced in [SS11], by applying a Johnson-Lindenstrauss random projection to the rows whose scores we are computing, we need just $\mathcal{O}(\log(1/\delta))$ system solves to compute constant factor approximations to the ridge scores with probability $1-\delta$. If we set $\delta = 1/\text{poly}(n)$ then we can union bound over our whole stream, using this trick with each batch of $\mathcal{O}(d)$ input rows. The batch probabilities will only be closer to the true ridge leverage scores than the non-batch probabilities and we will enjoy the same guarantees as ONLINE-SAMPLE.

```
$\tilde{\mathbf{A}} = $ ONLINE-SAMPLE$(\mathbf{A}, \epsilon, \delta)$, where $\mathbf{A}$ is an $n \times d$ matrix with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$, $\epsilon \in (0,1)$, $\delta > 0$.

  1. Set $\lambda := \delta/\epsilon$, $c := 8 \log d/\epsilon^2$.
  2. Let $\tilde{\mathbf{A}}_0$ be a $0 \times d$ matrix.
  3. For $i = 1, \ldots, n$:
     (a) Let $\tilde{l}_i := \min((1+\epsilon)\mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i, 1)$.
     (b) Let $p_i := \min(c\tilde{l}_i, 1)$.
     (c) Set $\tilde{\mathbf{A}}_i := \begin{cases} \begin{bmatrix} \tilde{\mathbf{A}}_{i-1} \\ \mathbf{a}_i/\sqrt{p_i} \end{bmatrix} & \text{with probability } p_i, \\ \tilde{\mathbf{A}}_{i-1} & \text{otherwise.} \end{cases}$
  4. Return $\tilde{\mathbf{A}} := \tilde{\mathbf{A}}_n$.
```

Figure 3.1: The basic online sampling algorithm

Additionally, it turns out that with a simple trick, it is possible to reduce the memory usage of the algorithm by a factor of ϵ^{-2}, bringing it down to $\mathcal{O}(d \log d \log(\epsilon \|A\|_2^2/\delta))$ (assuming the sparsifier is output to an output stream). Note that this expression gets *smaller* with ϵ; hence we obtain a row sampling algorithm with memory complexity independent of desired multiplicative precision. The basic idea is that, instead of keeping all previously sampled rows in memory, we store a smaller set of edges that give a constant factor spectral approximation, still enough to give good estimates of the online ridge leverage scores.

This result is presented in the algorithm SLIM-SAMPLE (Figure 3.1) and Lemma 3.3.5. A particularly interesting consequence for graphs with polynomially bounded edge weights is stated in Corollary 3.2.4.

Corollary 3.2.4 *Let G be a simple graph on d vertices, and $\epsilon \in (0,1)$. We can construct a $(1+\epsilon)$-sparsifier of G of size $\mathcal{O}(d \log^2 d/\epsilon^2)$, using only $\mathcal{O}(d \log^2 d)$ working memory in the online model.*

Proof This follows simply from applying Theorem 3.2.3 with $\delta = \epsilon/\sigma_{min}^2(\mathbf{A})$ and noting that the condition number of a graph on d vertices whose edge weights are within a multiplicative poly(d) of each other is polynomial in d. So $\log(\epsilon\|\mathbf{A}\|_2^2/\delta) = \log(\kappa^2(\mathbf{A})) = \mathcal{O}(\log d)$.

■

We remark that the algorithm of Corollary 3.2.4 can be made to run in nearly linear time in the stream size. We combine SLIM-SAMPLE with the batch processing idea described above. Because

$\mathbf{A}$ is a graph, our matrix approximation is always a symmetric diagonally dominant matrix, with $\mathcal{O}(d)$ nonzero entries. We can solve systems in it in time $\tilde{\mathcal{O}}(d)$. Using the Johnson-Lindenstrauss random projection trick of [SS11], we can compute approximate ridge leverage scores for a batch of $\mathcal{O}(d)$ rows with failure probability polynomially small in n in $\tilde{\mathcal{O}}(d \log n)$ time. Union bounding over the whole stream, we obtain nearly linear runtime.

To complement the row sampling results discussed above, we explore the limits of the proposed online setting. In Section 3.4 we present the algorithm ONLINE-BSS, which obtains sparsifiers with $\mathcal{O}(d \log(\epsilon \|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$ rows in the online setting (with larger memory requirements than the simpler sampling algorithms). Its analysis is given in Theorem 3.4.1. In Section 6.4, we show that this number of samples is in fact the best achievable, up to constant factors (Theorem 3.5.1). The $\log(\epsilon \|\mathbf{A}\|_2^2/\delta)$ factor is truly the cost of requiring rows to be selected in an online manner.

3.3 Analysis of sampling schemes

We begin by establishing a bound on the sum of online λ-ridge leverage scores. The intuition behind the proof of Theorem 3.2.2 is that whenever we append a row with a large online leverage score to a matrix, we increase its determinant significantly, as follows from the matrix determinant lemma (Lemma 3.3.1). Thus we can reduce upper bounding the online leverage scores of rows of the matrix to bounding its determinant.

Lemma 3.3.1 (Matrix determinant lemma) *Assume $\mathbf{S}$ is an invertible square matrix and $\mathbf{u}$ is a vector. Then*

$$\det(\mathbf{S} + \mathbf{u}\mathbf{u}^T) = (\det \mathbf{S})(1 + \mathbf{u}^T \mathbf{S}^{-1} \mathbf{u}).$$

Proof of Theorem 3.2.2: By Lemma 3.3.1, we have

$$\begin{aligned}
\det(\mathbf{A}_{i+1}^T \mathbf{A}_{i+1} + \lambda \mathbf{I}) &= \det(\mathbf{A}_i^T \mathbf{A}_i + \lambda \mathbf{I}) \cdot \left(1 + \mathbf{a}_{i+1}^T (\mathbf{A}_i^T \mathbf{A}_i + \lambda \mathbf{I})^{-1} \mathbf{a}_{i+1}\right) \\
&\geq \det(\mathbf{A}_i^T \mathbf{A}_i + \lambda \mathbf{I}) \cdot (1 + l_{i+1}) \\
&\geq \det(\mathbf{A}_i^T \mathbf{A}_i + \lambda \mathbf{I}) \cdot e^{l_{i+1}/2}.
\end{aligned}$$

Hence,

$$\begin{aligned}
\det(\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}) &= \det(\mathbf{A}_n^T \mathbf{A}_n + \lambda \mathbf{I}) \\
&\geq \det(\lambda \mathbf{I}) \cdot e^{\sum l_i/2} \\
&= \lambda^d e^{\sum l_i/2}.
\end{aligned}$$

We have $\det(\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I}) \leq (\|\mathbf{A}\|_2^2 + \lambda)^d$. Therefore

$$(\|\mathbf{A}\|_2^2 + \lambda)^d \geq \lambda^d e^{\sum l_i/2}.$$

Taking logarithms of both sides, we obtain

$$d\log(\|\mathbf{A}\|_2^2+\lambda) \geq d\log\lambda + \sum l_i/2,$$
$$\sum l_i \leq 2d\log(1+\|\mathbf{A}\|_2^2/\lambda).$$

■

We now turn to analyzing the algorithm ONLINE-SAMPLE. Because the samples taken by the algorithm are *not* independent, we are not able to use a standard matrix Chernoff bound like the one in Theorem 3.2.1. However, we do know that whether we take row i does not depend on later rows; thus, we are able to analyze the process as a martingale. We will use a matrix version of the Freedman inequality given by Tropp.

Theorem 3.3.2 (Matrix Freedman inequality [Tro11]) *Let $\mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_n$ be a matrix martingale whose values are self-adjoint matrices with dimension d, and let $\mathbf{X}_1, \ldots, \mathbf{X}_n$ be the difference sequence. Assume that the difference sequence is uniformly bounded in the sense that*

$$\left\|\mathbf{X}_k\right\|_2 \leq R \text{ almost surely, for } k = 1, \ldots, n.$$

Define the predictable quadratic variation process of the martingale:

$$\mathbf{W}_k := \sum_{j=1}^{k} \mathbb{E}_{j-1}\left[\mathbf{X}_j^2\right], \text{ for } k = 1, \ldots, n.$$

Then, for all $\epsilon > 0$ and $\sigma^2 > 0$,

$$\mathbb{P}\left[\left\|\mathbf{Y}_n\right\|_2 \geq \epsilon \text{ and } \left\|\mathbf{W}_n\right\|_2 \leq \sigma^2\right] \leq d \cdot \exp\left(-\frac{-\epsilon^2/2}{\sigma^2 + R\epsilon/3}\right)$$

We begin by showing that the output of ONLINE-SAMPLE is in fact an approximation of $\mathbf{A}$, and that the approximate online leverage scores are lower bounded by the actual online leverage scores.

Lemma 3.3.3 *After running* ONLINE-SAMPLE*, it holds with high probability that*

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I},$$

and also

$$\tilde{l}_i \geq \mathbf{a}_i^T(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{a}_i$$

for $i = 1, \ldots, n$.

Proof Let

$$\mathbf{u}_i := (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1/2}\mathbf{a}_i.$$

We construct a matrix martingale $\mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_n \in \mathbb{R}^{d\times d}$ with the difference sequence $\mathbf{X}_1, \ldots, \mathbf{X}_n$. Set $\mathbf{Y}_0 = \mathbf{0}$. If $\left\|\mathbf{Y}_{i-1}\right\|_2 \geq \epsilon$, we set $\mathbf{X}_i := \mathbf{0}$. Otherwise, let

$$\mathbf{X}_i := \begin{cases} (1/p_i - 1)\mathbf{u}_i\mathbf{u}_i^T & \text{if } \mathbf{a}_i \text{ is sampled in } \tilde{\mathbf{A}}, \\ -\mathbf{u}_i\mathbf{u}_i^T & \text{otherwise.} \end{cases}$$

Note that in this case we have

$$\mathbf{Y}_{i-1} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1/2}(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} - \mathbf{A}_{i-1}^T\mathbf{A}_{i-1})(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1/2}.$$

Hence, since $\left\|\mathbf{Y}_{i-1}\right\|_2 < \epsilon$, we have

$$\begin{aligned}
\tilde{l}_i &= \min((1+\epsilon)\mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i, 1) \\
&\geq \min((1+\epsilon)\mathbf{a}_i^T(\mathbf{A}_{i-1}^T\mathbf{A}_{i-1} + \lambda\mathbf{I} + \epsilon(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}))^{-1}\mathbf{a}_i, 1) \\
&\geq \min((1+\epsilon)\mathbf{a}_i^T((1+\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}))^{-1}\mathbf{a}_i, 1) \\
&= \mathbf{a}_i^T(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{a}_i \\
&= \mathbf{u}_i^T\mathbf{u}_i,
\end{aligned}$$

and so

$$p_i \geq \min(c\mathbf{u}_i^T\mathbf{u}_i, 1).$$

Note that if $p_i = 1$, then $\mathbf{X}_i = 0$. Otherwise, we have $p_i \geq c\mathbf{u}_i^T\mathbf{u}_i$ and so

$$\|\mathbf{X}_i\|_2 \leq 1/c$$

and

$$\begin{aligned}
\mathbb{E}_{i-1}\left[\mathbf{X}_i^2\right] &\preceq p_i \cdot (1/p_i - 1)^2(\mathbf{u}_i\mathbf{u}_i^T)^2 + (1-p_i)\cdot(\mathbf{u}_i\mathbf{u}_i^T)^2 \\
&= (\mathbf{u}_i\mathbf{u}_i^T)^2/p_i \\
&\preceq \mathbf{u}_i\mathbf{u}_i^T/c.
\end{aligned}$$

And so, for the predictable quadratic variation process of the martingale $\{\mathbf{Y}_i\}$:

$$\mathbf{W}_i := \sum_{k=1}^{i}\mathbb{E}_{k-1}\left[\mathbf{X}_k^2\right],$$

we have

$$\|\mathbf{W}_i\|_2 \le \left\| \sum_{k=1}^{i} \mathbf{u}_i\mathbf{u}_i^T/c \right\|_2 \le 1/c.$$

Therefore by, Theorem 3.3.2, we have

$$\begin{aligned}\mathbb{P}\left[\|\mathbf{Y}_n\|_2 \ge \epsilon\right] &\le d \cdot \exp\left(\frac{-\epsilon^2/2}{1/c + \epsilon/(3c)}\right)\\ &\le d \cdot \exp(-c\epsilon^2/4)\\ &= 1/d.\end{aligned}$$

This implies that with high probability

$$\left\|(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1/2}(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I})(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1/2} - \mathbf{I}\right\|_2 \le \epsilon$$

and so

$$(1-\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}) \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I} \preceq (1+\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}).$$

Subtracting $\lambda\mathbf{I} = (\delta/\epsilon)\mathbf{I}$ from all sides, we get

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I}.$$

■

If we set c in ONLINE-SAMPLE to be proportional to $\log n$ rather than $\log d$, we would be able to take a union bound over all the rows and guarantee that with high probability all the approximate online leverage scores $\tilde{l}_i$ are close to true online leverage scores l_i. Thus Theorem 3.2.2 would imply that ONLINE-SAMPLE only selects $\mathcal{O}(d \log n \log(\|\mathbf{A}\|_2^2/\lambda)/\epsilon^2)$ rows with high probability.

In order to remove the dependency on n, we have to sacrifice achieving close approximations to l_i at every step. Instead, we show that the *sum* of the computed approximate online leverage scores is still small with high probability, using a custom Chernoff bound.

Lemma 3.3.4 *After running* ONLINE-SAMPLE*, it holds with high probability that*

$$\sum_{i=1}^{n} \tilde{l}_i = \mathcal{O}(d \log(\|\mathbf{A}\|_2^2/\lambda)).$$

Proof Define

$$\delta_i := \log\det(\tilde{\mathbf{A}}_i^T\tilde{\mathbf{A}}_i + \lambda\mathbf{I}) - \log\det(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I}).$$

The proof closely follows the idea from the proof of Theorem 3.2.2. We will aim to show that large values of $\tilde{l}_i$ correlate with large values of δ_i. However, the sum of δ_i can be bounded by the logarithm of the ratio of the determinants of $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I}$ and $\lambda\mathbf{I}$. First, we will show that $\mathbb{E}_{i-1}\left[\exp(\tilde{l}_i/8 - \delta_i)\right]$ is always at most 1. We begin by an application of Lemma 3.3.1.

$$\begin{aligned}\mathbb{E}_{i-1}\left[\exp(\tilde{l}_i/8 - \delta_i)\right] &= p_i \cdot e^{l_i/8}(1 + \mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i/p_i)^{-1} + (1-p_i)e^{l_i/8} \\ &\leq p_i \cdot (1 + l_i/4)(1 + \mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i/p_i)^{-1} + (1-p_i)(1 + l_i/4).\end{aligned}$$

If $c\tilde{l}_i < 1$, we have $p_i = c\tilde{l}_i$ and $\tilde{l}_i = (1+\epsilon)\mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i$, and so:

$$\begin{aligned}\mathbb{E}_{i-1}\left[\exp(\tilde{l}_i/8 - \delta_i)\right] &\leq c\tilde{l}_i \cdot (1 + l_i/4)(1 + 1/((1+\epsilon)c))^{-1} + (1 - c\tilde{l}_i)(1 + l_i/4) \\ &= (1 + l_i/4)(cl_i(1 + 1/((1+\epsilon)c))^{-1} + 1 - cl_i) \\ &\leq (1 + l_i/4)(1 + cl_i(1 - 1/(4c) - 1)) \\ &= (1 + l_i/4)(1 - l_i/4) \\ &\leq 1.\end{aligned}$$

Otherwise, we have $p_i = 1$ and so:

$$\begin{aligned}\mathbb{E}_{i-1}\left[\exp(\tilde{l}_i/8 - \delta_i)\right] &\leq (1 + l_i/4)(1 + \mathbf{a}_i^T(\tilde{\mathbf{A}}_{i-1}^T\tilde{\mathbf{A}}_{i-1} + \lambda\mathbf{I})^{-1}\mathbf{a}_i)^{-1} \\ &\leq (1 + l_i/4)(1 + l_i)^{-1} \\ &\leq 1.\end{aligned}$$

We will now analyze the expected product of $\exp(\tilde{l}_i/8 - \delta_i)$ over the first k steps. We group the expectation over the first k steps into one over the first $k-1$ steps, aggregating the expectation for the last step by using one-way independence. For $k \geq 1$ we have

$$\begin{aligned}\mathbb{E}\left[\exp\left(\sum_{i=1}^{k}\tilde{l}_i/8 - \delta_i\right)\right] &= \underset{\text{first } k-1 \text{ steps}}{\mathbb{E}}\left[\exp\left(\sum_{i=1}^{k-1}\tilde{l}_i/8 - \delta_i\right)\mathbb{E}_{k-1}\left[\exp(\tilde{l}_k/8 - \delta_k)\right]\right] \\ &\leq \mathbb{E}\left[\exp\left(\sum_{i=1}^{k-1}\tilde{l}_i/8 - \delta_i\right)\right],\end{aligned}$$

and so by induction on k

$$\mathbb{E}\left[\exp\left(\sum_{i=1}^{n}\tilde{l}_i/8 - \delta_i\right)\right] \leq 1.$$

Hence by Markov's inequality

$$\mathbb{P}\left[\sum_{i=1}^{n} \tilde{l}_i > 8d + 8\sum_{i=1}^{n} \delta_i\right] \leq e^{-d}.$$

By Lemma 3.3.3, with high probability we have

$$\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I} \preceq (1+\epsilon)(\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I}).$$

We also have with high probability

$$\begin{aligned}\det(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I}) &\leq (1+\epsilon)^d(\|A\|_2^2 + \lambda)^d, \\ \log\det(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I}) &\leq d(1 + \log(\|A\|_2^2 + \lambda)).\end{aligned}$$

Hence, with high probability it holds that

$$\begin{aligned}\sum_{i=1}^{n} \delta_i &= \log\det(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} + \lambda\mathbf{I}) - d\log(\lambda) \\ &\leq d(1 + \log(\|A\|_2^2 + \lambda) - \log(\lambda)) \\ &= d(1 + \log(1 + \|A\|_2^2/\lambda)).\end{aligned}$$

And so, with high probability,

$$\begin{aligned}\sum_{i=1}^{n} \tilde{l}_i &\leq 8d + 8\sum_{i=1}^{n} \delta_i \\ &\leq 9d + 8d\log(1 + \|A\|_2^2/\lambda) \\ &= \mathcal{O}(d\log(\|A\|_2^2/\lambda)).\end{aligned}$$

■

Proof of Theorem 3.2.3: The thesis follows immediately from Lemmas 3.3.3 and 3.3.4.

■

We now consider a simple modification of ONLINE-SAMPLE that removes dependency on ϵ from the working memory usage with no additional cost.

Lemma 3.3.5 *Let $\tilde{\mathbf{A}}$ be the matrix returned by* SLIM-SAMPLE$(\mathbf{A}, \epsilon, \delta)$*. Then, with high probability,*

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \preceq \tilde{\mathbf{A}}^T\tilde{\mathbf{A}} \preceq (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I},$$

```
$\tilde{\mathbf{A}} = \textsc{Slim-Sample}(\mathbf{A}, \epsilon, \delta)$, where $\mathbf{A}$ is an $n \times d$ matrix with rows $\mathbf{a}_1, \dots, \mathbf{a}_n$, $\epsilon \in (0,1)$, $\delta > 0$.

  1. Set $\lambda := \delta/\epsilon$, $c := 8\log d/\epsilon^2$.
  2. Let $\tilde{\mathbf{A}}_0$ be a $0 \times d$ matrix.
  3. Let $\tilde{l}_1, \dots, \tilde{l}_n$ be the approximate online leverage scores computed by an independent
     instance of $\textsc{Online-Sample}(\mathbf{A}, 1/2, \delta/(2\epsilon))$.
  4. For $i = 1, \dots, n$:
       (a) Let $p_i := \min(c\tilde{l}_i, 1)$.
       (b) Set $\tilde{\mathbf{A}}_i := \begin{cases} \begin{bmatrix} \tilde{\mathbf{A}}_{i-1} \\ \mathbf{a}_i/\sqrt{p_i} \end{bmatrix} & \text{with probability } p_i, \\ \tilde{\mathbf{A}}_{i-1} & \text{otherwise.} \end{cases}$
  5. Return $\tilde{\mathbf{A}} := \tilde{\mathbf{A}}_n$.
```

Figure 3.1: The low-memory online sampling algorithm

and the number of rows in $\tilde{\mathbf{A}}$ *is* $\mathcal{O}(d \log d \log(\epsilon\|\mathbf{A}\|_2^2/\delta)/\epsilon^2)$.

Moreover, with high probability the algorithm SLIM-SAMPLE*'s memory requirement is dominated by storing* $\mathcal{O}(d \log d \log(\epsilon\|\mathbf{A}\|_2^2/\delta))$ *rows of* $\mathbf{A}$.

Proof As the samples are independent, the thesis follows from Theorem 3.2.1 and Lemmas 3.3.3 and 3.3.4.

■

3.4 Asymptotically optimal algorithm

In addition to sampling by online leverage scores, there is also a variant of the "BSS" method [BSS12] that applies in our setting. Like the original [BSS12], this approach removes the $\log d$ factor from the row count of the output sparsifier, matching the lower bound for online sampling given in Theorem 3.5.1.

Unlike [BSS12] itself, our algorithm is randomized – it is similar to, and inspired by, the randomized version of BSS from [LS15] (which unfortunately is not yet available online), especially the simple version contained in the appendix of that paper (the main difference from that is considering each row separately). In fact, this algorithm is of the same form as the basic sampling algorithm, in that when each row comes in, a probability p_i is assigned to it, and it is kept (and rescaled) with probability p_i and rejected otherwise. The key difference is the definition of the p_i.

There are also some differences in the nature of the algorithm and its guarantees. Notably, the p_i cannot be computed solely based on the row sample output so far–it is necessary to "remember" the entire matrix given so far. This means that the BSS method is not very memory efficient, using $O(d^2)$ space. Additionally, the online leverage score method gives bounds on both the size of the sparsifier and its accuracy with high probability. In contrast, this method only gives an *expected* bound on the sparsifier size, while it *never* fails to output a correct sparsifier. Note that these guarantees are essentially the same as those in the appendix of [LS15].

One may, however, improve the memory dependence in some cases simply by running it on the output stream of the online leverage score sampling method. This reduces the storage cost to the size of that sparsifier. The BSS method still does not produce an actual space *savings* (in particular, there is a still a $\log d$ factor in space), but it does reduce the number of rows in the output stream while only blowing up the space usage by $O(1/\epsilon^2)$ (due to requiring the storage of an ϵ-quality sparsifier rather than only $O(1)$).

The BSS method maintains two matrices, $\mathbf{B}_i^U$ and $\mathbf{B}_i^L$, acting as upper and lower "barriers". The current sparsifier will always fall between them:

$$\mathbf{B}_i^L \prec \tilde{\mathbf{A}}_i^T \tilde{\mathbf{A}}_i^T \prec \mathbf{B}_i^U.$$

This guarantee, at the end of the algorithm, will ensure that $\tilde{\mathbf{A}}$ is a valid approximation.

Below, we give the actual BSS algorithm.

```
$\tilde{\mathbf{A}} = \text{ONLINE-BSS}(\mathbf{A}, \epsilon, \delta)$, where $\mathbf{A}$ is an $n \times d$ matrix with rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$, $\epsilon \in (0,1)$, $\delta > 0$.

    1. Set $c_U = \frac{2}{\epsilon} + 1$ and $c_L = \frac{2}{\epsilon} - 1$.
    2. Let $\tilde{\mathbf{A}}_0$ be a $0 \times d$ matrix, $\mathbf{B}_0^U = \delta\mathbf{I}$, $\mathbf{B}_0^L = -\delta\mathbf{I}$.
    3. For $i = 1, \ldots, n$:
        (a) Let $\mathbf{X}_{i-1}^U = (\mathbf{B}_{i-1}^U - \tilde{\mathbf{A}}_{i-1}^T \tilde{\mathbf{A}}_{i-1})$, $\mathbf{X}_{i-1}^L = (\tilde{\mathbf{A}}_{i-1}^T \tilde{\mathbf{A}}_{i-1} - \mathbf{B}_{i-1}^L)$.
        (b) Let $p_i := \min(c_U \mathbf{a}_i^T (\mathbf{X}_{i-1}^U)^{-1} \mathbf{a}_i + c_L \mathbf{a}_i^T (\mathbf{X}_{i-1}^L)^{-1} \mathbf{a}_i, 1)$.
        (c) Set $\tilde{\mathbf{A}}_i := \begin{cases} \begin{bmatrix} \tilde{\mathbf{A}}_{i-1} \\ \mathbf{a}_i/\sqrt{p_i} \end{bmatrix} & \text{with probability } p_i, \\ \tilde{\mathbf{A}}_{i-1} & \text{otherwise.} \end{cases}$
        (d) Set $\mathbf{B}_i^U = \mathbf{B}_{i-1}^U + (1+\epsilon)\mathbf{a}_i \mathbf{a}_i^T$, $\mathbf{B}_i^L = \mathbf{B}_{i-1}^L + (1-\epsilon)\mathbf{a}_i \mathbf{a}_i^T$.
    4. Return $\tilde{\mathbf{A}} := \tilde{\mathbf{A}}_n$.
```

Figure 3.1: The Online BSS Algorithm

Theorem 3.4.1

1. *The online BSS algorithm always outputs $\tilde{A}$ such that*

$$(1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I} \prec \tilde{\mathbf{A}}^T\tilde{\mathbf{A}}^T \prec (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I}$$

2. *The probability that a row $\mathbf{a}_i$ is included in $\tilde{\mathbf{A}}$ is at most $\frac{8}{\epsilon^2}l_i$, where l_i is the online $\frac{2\delta}{\epsilon}$-ridge leverage score of $\mathbf{a}_i$. That is $l_i = \min(\mathbf{a}_i^T\left(\mathbf{A}_i^T\mathbf{A}_i + \frac{2\delta}{\epsilon}I\right)^{-1}\mathbf{a}_i, 1)$. The expected number of rows in $\tilde{\mathbf{A}}$ is thus at most $\frac{8}{\epsilon^2}\sum_{i=1}^n l_i = \mathcal{O}(d\log(\epsilon\left\|\mathbf{A}\right\|_2^2/\delta)/\epsilon^2)$.*

Proof of Theorem 3.4.1 part 1: We first note the basic invariant that $\mathbf{X}_i^U$ and $\mathbf{X}_i^L$ always remain positive definite–or equivalently,

$$\mathbf{B}_i^L \prec \tilde{\mathbf{A}}_i^T\tilde{\mathbf{A}}_i^T \prec \mathbf{B}_i^U.$$

We may prove this by induction on i. The base case follows from the initialization of $\tilde{\mathbf{A}}_0$, $\mathbf{B}_0^U$ and $\mathbf{B}_0^L$. For each successive step, we consider two possibilities.

The first is that $p_i = 1$. In that case, $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ always increases by exactly $\mathbf{a}_i\mathbf{a}_i^T$, $\mathbf{B}^U$ by $(1+\epsilon)\mathbf{a}_i\mathbf{a}_i^T$ and $\mathbf{B}^L$ by $(1-\epsilon)\mathbf{a}_i\mathbf{a}_i^T$. Thus $\mathbf{X}^U$ and $\mathbf{X}^L$ increase by exactly $\epsilon\mathbf{a}_i\mathbf{a}_i^T$, which is positive semidefinite, and so remain positive definite.

In the other case, $p_i < 1$. Now, $\mathbf{X}^U$ decreases by at most the increase in $\tilde{\mathbf{A}}_i^T\tilde{\mathbf{A}}_i^T$, or

$$\mathbf{M}_i = \frac{\mathbf{a}_i\mathbf{a}_i^T}{p}.$$

Since $c_U > 1$, $p > \mathbf{a}_i^T(\mathbf{X}_{i-1}^U)^{-1}\mathbf{a}_i$, so $\mathbf{a}_i\mathbf{a}_i^T \prec p\mathbf{X}_{i-1}^U$ and $\mathbf{M}_i \prec \mathbf{X}_{i-1}^U$. Subtracting this then must leave $\mathbf{X}^U$ positive definite. Similarly, $\mathbf{X}^L$ decreases by at most the increase in $\mathbf{B}^L$, which is $(1-\epsilon)\mathbf{a}_i\mathbf{a}_i^T \prec \mathbf{a}_i\mathbf{a}_i^T$. Since $c_L > 1$ and $p < 1$, $\mathbf{a}_i^T(\mathbf{X}_{i-1}^L)^{-1}\mathbf{a}_i < 1$, and $\mathbf{a}_i\mathbf{a}_i^T \prec \mathbf{X}_{i-1}^L$. Subtracting this similarly leaves $\mathbf{X}^L$ positive definite.

Finally, we note that

$$\mathbf{B}_n^U = (1+\epsilon)\mathbf{A}^T\mathbf{A} + \delta\mathbf{I}$$
$$\mathbf{B}_n^L = (1-\epsilon)\mathbf{A}^T\mathbf{A} - \delta\mathbf{I}.$$

This gives the desired result.

■

To prove part 2, we will use quantities of the form $\mathbf{v}^T\mathbf{X}^{-1}\mathbf{v}$. We will need a lemma describing how this behaves under a random rank-1 update:

Lemma 3.4.2 *Given a positive definite matrix $\mathbf{X}$, two vectors $\mathbf{u}$ and $\mathbf{v}$, two multipliers a and b and a probability p, define the random variable $\mathbf{X}'$ to be $X - a\mathbf{u}\mathbf{u}^T$ with probability p and $X - b\mathbf{u}\mathbf{u}^T$ otherwise. Then if $\mathbf{u}^T\mathbf{X}^{-1}\mathbf{u} = 1$,*

$$\mathbb{E}\left[\mathbf{v}^T\mathbf{X}'^{-1}\mathbf{v} - \mathbf{v}^T\mathbf{X}^{-1}\mathbf{v}\right] = (\mathbf{v}^\mathbf{T}\mathbf{X}^{-1}\mathbf{u})^2\frac{pa + (1-p)b - ab}{(1-a)(1-b)}.$$

Proof We apply the Sherman-Morrison formula to each of the two possibilities (subtracting $a\mathbf{u}\mathbf{u}^T$ and $b\mathbf{u}\mathbf{u}^T$ respectively). These give $\mathbf{X}'$ values of respectively

$$\mathbf{X}^{-1} + a\frac{\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}}{1 - a\mathbf{u}^T\mathbf{X}^{-1}\mathbf{u}} = \mathbf{X}^{-1} + \frac{a}{1-a}\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}$$

and

$$\mathbf{X}^{-1} + b\frac{\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}}{1 - b\mathbf{u}^T\mathbf{X}^{-1}\mathbf{u}} = \mathbf{X}^{-1} + \frac{b}{1-b}\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}.$$

The values of $\mathbf{v}^T\mathbf{X}'^{-1}\mathbf{v} - \mathbf{v}^T\mathbf{X}^{-1}\mathbf{v}$ are then respectively

$$\frac{a}{1-a}\mathbf{v}^T\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}\mathbf{v} = (\mathbf{v}^\mathbf{T}\mathbf{X}^{-1}\mathbf{u})^2\frac{a}{1-a}$$

and

$$\frac{b}{1-b}\mathbf{v}^T\mathbf{X}^{-1}\mathbf{u}\mathbf{u}^T\mathbf{X}^{-1}\mathbf{v} = (\mathbf{v}^\mathbf{T}\mathbf{X}^{-1}\mathbf{u})^2\frac{b}{1-b}.$$

Combining these gives the stated result.

■

Proof of Theorem 3.4.1 part 2: First, we introduce some new matrices to help in the analysis:

$$\begin{aligned}\mathbf{C}^U_{i,j} &= \delta\mathbf{I} + \frac{\epsilon}{2}\mathbf{A}_i^T\mathbf{A}_i + \left(1 + \frac{\epsilon}{2}\right)\mathbf{A}_j^T\mathbf{A}_j\\ \mathbf{C}^L_{i,j} &= -\delta\mathbf{I} - \frac{\epsilon}{2}\mathbf{A}_i^T\mathbf{A}_i + \left(1 - \frac{\epsilon}{2}\right)\mathbf{A}_j^T\mathbf{A}_j.\end{aligned}$$

Note that $\mathbf{C}^U_{i,i} = \mathbf{B}^U_i$, $\mathbf{C}^L_{i,i} = \mathbf{B}^L_i$, and for $j \le i$, $\mathbf{C}^U_{i,j} \succeq \mathbf{B}^U_j$ and $\mathbf{C}^L_{i,j} \preceq \mathbf{B}^L_j$.

We can then define

$$\begin{aligned}\mathbf{Y}^U_{i,j} &= \mathbf{C}^U_{i,j} - \tilde{\mathbf{A}}_j^T\tilde{\mathbf{A}}_j\\ \mathbf{Y}^L_{i,j} &= \tilde{\mathbf{A}}_j^T\tilde{\mathbf{A}}_j - \mathbf{C}^L_{i,j}.\end{aligned}$$

We then have, similarly, $\mathbf{Y}^U_{i,i} = \mathbf{X}^U_i$, $\mathbf{Y}^L_{i,i} = \mathbf{X}^L_i$, and for $j \le i$, $\mathbf{Y}^U_{i,j} \succeq \mathbf{X}^U_j$ and $\mathbf{Y}^L_{i,j} \succeq \mathbf{X}^L_j$.

We will assume that $l_i < 1$, since otherwise the claim is immediate (as probabilities cannot exceed 1). Now, note that

$$\begin{aligned}
\mathbf{a}_i^T(\mathbf{Y}_{i,0}^U)^{-1}\mathbf{a}_i &= \mathbf{a}_i^T(\mathbf{Y}_{i,0}^L)^{-1}\mathbf{a}_i \\
&= \mathbf{a}_i^T\left(\frac{\epsilon}{2}\mathbf{A}_i^T\mathbf{A}_i + \delta I\right)^{-1}\mathbf{a}_i \\
&= \frac{2}{\epsilon}\left(\mathbf{A}_i^T\mathbf{A}_i + \frac{2\delta}{\epsilon}I\right)^{-1}\mathbf{a}_i \\
&= \frac{2}{\epsilon}l_i.
\end{aligned}$$

Next, we will aim to show that for $j < i-1$,

$$\begin{aligned}
\mathbb{E}\left[\mathbf{a}_i^T\mathbf{Y}_{i-1,j+1}^U\mathbf{a}_i\right] &\le \mathbb{E}\left[\mathbf{a}_i^T\mathbf{Y}_{i-1,j}^U\mathbf{a}_i\right] \\
\mathbb{E}\left[\mathbf{a}_i^T\mathbf{Y}_{i-1,j+1}^L\mathbf{a}_i\right] &\le \mathbb{E}\left[\mathbf{a}_i^T\mathbf{Y}_{i-1,j}^L\mathbf{a}_i\right]
\end{aligned}$$

In particular, we will simply show that conditioned on any choices for the first j edges, the expected value of $\mathbf{a}_i^T\mathbf{Y}_{i-1,j+1}^U\mathbf{a}_i$ is no larger than $\mathbf{a}_i^T\mathbf{Y}_{i-1,j}^U\mathbf{a}_i$, and analogously for $\mathbf{Y}^L$.

Similar to the proof of part 1, we separately consider the case where $p_{j+1} = 1$. In that case, the positive semidefinite matrix $\frac{\epsilon}{2}\mathbf{a}_j\mathbf{a}_j^T$ is simply added to $\mathbf{Y}^U$ and $\mathbf{Y}^L$. Adding this can only decrease the values of $\mathbf{a}_i^T\mathbf{Y}^U\mathbf{a}_i$ and $\mathbf{a}_i^T\mathbf{Y}^L\mathbf{a}_i$.

The $p_{j+1} < 1$ case is more tricky. Here, we define the vector $\mathbf{w}_{j+1} = \frac{\mathbf{a}_{j+1}}{\sqrt{p_{j+1}}}$. Importantly

$$\begin{aligned}
p_{j+1} &\ge c_U\mathbf{a}_{j+1}^T(\mathbf{X}_j^U)^{-1}\mathbf{a}_{j+1} \ge c_U\mathbf{a}_{j+1}^T(\mathbf{Y}_{i-1,j}^U)^{-1}\mathbf{a}_{j+1} \\
p_{j+1} &\ge c_L\mathbf{a}_{j+1}^T(\mathbf{X}_j^L)^{-1}\mathbf{a}_{j+1} \ge c_L\mathbf{a}_{j+1}^T(\mathbf{Y}_{i-1,j}^L)^{-1}\mathbf{a}_{j+1}.
\end{aligned}$$

This means that

$$\begin{aligned}
\mathbf{w}_{j+1}^T(\mathbf{Y}_{i-1,j}^U)^{-1}\mathbf{w}_{j+1}^T &\le \frac{1}{c_U} \\
\mathbf{w}_{j+1}^T(\mathbf{Y}_{i-1,j}^L)^{-1}\mathbf{w}_{j+1}^T &\le \frac{1}{c_L}.
\end{aligned}$$

Now, we additionally define

$$\begin{aligned}
s_{j+1}^U &= \mathbf{w}_{j+1}^T(\mathbf{Y}_{i-1,j}^U)^{-1}\mathbf{w}_{j+1}^T \\
s_{j+1}^L &= \mathbf{w}_{j+1}^T(\mathbf{Y}_{i-1,j}^L)^{-1}\mathbf{w}_{j+1}^T \\
\mathbf{u}_{j+1}^U &= \frac{\mathbf{w}_{j+1}}{\sqrt{s_{j+1}^U}} \\
\mathbf{u}_{j+1}^L &= \frac{\mathbf{w}_{j+1}}{\sqrt{s_{j+1}^L}}.
\end{aligned}$$

We then deploy Lemma 3.4.2 to compute the expectations. For the contribution from the upper barrier, we use $\mathbf{X} = \mathbf{Y}_{i-1,j}^U$, $\mathbf{u} = \mathbf{u}_{j+1}^U$, $\mathbf{v} = \mathbf{a}_i^T$, $a = -s_{j+1}^U(1 - p_{j+1}(1 + \epsilon/2))$, $b = s_{j+1}^U p_{j+1}(1 + \epsilon/2)$, $p = p_{j+1}$. For the lower barrier, we use $\mathbf{X} = \mathbf{Y}_{i-1,j}^L$, $\mathbf{u} = \mathbf{u}_{j+1}^L$, $\mathbf{v} = \mathbf{a}_i^T$, $a = s_{j+1}^L(1 - p_{j+1}(1 - \epsilon/2))$, $b = -s_{j+1}^L p_{j+1}(1 - \epsilon/2)$, $p = p_{j+1}$. In both cases we can see that the numerator of the expected change is nonpositive.

Finally, this implies that the probability that row i is sampled is

$$\begin{aligned}
\mathbb{E}\left[p_i\right] &= c_U\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{X}_{i-1}^U)^{-1}\mathbf{a}_i\right] + c_L\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{X}_{i-1}^L)^{-1}\mathbf{a}_i\right] \\
&= c_U\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{Y}_{i-1,i-1}^U)^{-1}\mathbf{a}_i\right] + c_L\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{Y}_{i-1,i-1}^L)^{-1}\mathbf{a}_i\right] \\
&\leq c_U\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{Y}_{i-1,0}^U)^{-1}\mathbf{a}_i\right] + c_L\,\mathbb{E}\left[\mathbf{a}_i^T(\mathbf{Y}_{i-1,0}^L)^{-1}\mathbf{a}_i\right] \\
&= \frac{2}{\epsilon}(c_U + c_L)l_i \\
&= \frac{8}{\epsilon^2}l_i
\end{aligned}$$

as desired.

■

3.5 Matching Lower Bound

Here we show that the row count obtained by Theorem 3.4.1 is in fact optimal. While it is possible to obtain a spectral approximation with $\mathcal{O}(d/\epsilon^2)$ rows in the offline setting, online sampling always incurs a loss of $\Omega\left(\log(\epsilon\|\mathbf{A}\|_2^2/\delta)\right)$ and must sample $\Omega\left(\frac{d\log(\epsilon\|\mathbf{A}\|_2^2/\delta)}{\epsilon^2}\right)$ rows.

Theorem 3.5.1 *Assume that $\epsilon\|\mathbf{A}\|_2^2 \geq c_1\delta$ and $\epsilon \geq c_2/\sqrt{d}$, for fixed constants c_1 and c_2. Then any algorithm that selects rows in an online manner and outputs a spectral approximation to $\mathbf{A}^T\mathbf{A}$ with $(1+\epsilon)$ multiplicative error and δ additive error with probability at least $1/2$ must sample $\Omega\left(\frac{d\log(\epsilon\|\mathbf{A}\|_2^2/\delta)}{\epsilon^2}\right)$ rows of $\mathbf{A}$ in expectation.*

Note that the assumptions we make lower bounding $\epsilon\|\mathbf{A}\|_2^2$ and ϵ are very minor. They are just ensuring that $\log(\epsilon\|\mathbf{A}\|_2^2/\delta) \geq 1$ and that ϵ is not so small that we can essentially just sample all rows of $\mathbf{A}$.

Proof We apply Yao's minimax principle, constructing, for any large enough M, a distribution on inputs $\mathbf{A}$ with $\|\mathbf{A}\|_2^2 \leq M$ for which any deterministic online row selection algorithm that succeeds with probability at least $1/2$ must output $\Omega\left(\frac{d\log(\epsilon M/\delta)}{\epsilon^2}\right)$ rows in expectation. The best randomized algorithm that works with probability $1/2$ on any input matrix with $\|\mathbf{A}\|_2^2 \leq M$ therefore must select at least $\Omega\left(\frac{d\log(\epsilon M/\delta)}{\epsilon^2}\right)$ rows in expectation on the worst case input, giving us the lower bound.

Our distribution is as follows. We select an integer N uniformly at random from $[1, \log(M\epsilon/\delta)]$. We then stream in the vertex edge incidence matrices of N complete graphs on d vertices. We double the weight of each successive graph. Intuitively, spectrally approximating a complete graph requires selecting $\Omega(d/\epsilon^2)$ edges [BSS12] (as long as $\epsilon \geq c_2/\sqrt{d}$ for some fixed constant c_2). Each time we stream in a new graph with double the weight, we force the algorithm to add $\Omega(d/\epsilon^2)$ more edges to its output, eventually forcing it to output $\Omega(d/\epsilon^2 \cdot N)$ edges – $\Omega(d\log(M\epsilon/\delta)/\epsilon^2)$ in expectation.

Specifically, let $\mathbf{K}_d$ be the $\binom{d}{2} \times d$ vertex edge incidence matrix of the complete graph on d vertices. $\mathbf{K}_d^T\mathbf{K}_d$ is the Laplacian matrix of the complete graph on d vertices. We weight the first graph so that its Laplacian has all its nonzero eigenvalues equal to δ/ϵ. (That is, each edge has weight $\frac{\delta}{d\epsilon}$). In this way, even if we select $N = \lfloor\log(M\epsilon/\delta)\rfloor$ we will have overall $\|\mathbf{A}\|_2^2 \leq \delta/\epsilon + 2\delta/\epsilon + \ldots 2^{\lfloor\log(M\epsilon/\delta)\rfloor-1}\delta/\epsilon \leq M$.

Even if $N = 1$, all nonzero eigenvalues of $\mathbf{A}^T\mathbf{A}$ are at least δ/ϵ, so achieving $(1+\epsilon)$ multiplicative error and $\delta\mathbf{I}$ additive error is equivalent to achieving $(1+2\epsilon)$ multiplicative error. $\mathbf{A}^T\mathbf{A}$ is a graph Laplacian so has a null space. However, as all rows are orthogonal to the null space, achieving additive error $\delta\mathbf{I}$ is equivalent to achieving additive error $\delta\mathbf{I}_r$ where $\mathbf{I}_r$ is the identity projected to the span of $\mathbf{A}^T\mathbf{A}$. $\delta\mathbf{I}_r \preceq \epsilon\mathbf{A}^T\mathbf{A}$ which is why we must achieve $(1+2\epsilon)$ multiplicative error.

In order for a deterministic algorithm to be correct with probability $1/2$ on our distribution, it must be correct for at least $1/2$ of our $\lfloor\log(M\epsilon/\delta)\rfloor$ possible choices of N.

Let i be the lowest choice of N for which the algorithm is correct. By the lower bound of [BSS12], the algorithm must output $\Omega(d/\epsilon^2)$ rows of $\mathbf{A}_i$ to achieve a $(1+2\epsilon)$ multiplicative factor spectral approximation. Here $\mathbf{A}_i$ is the input consisting of the vertex edge incidence matrices of i increasingly weighted complete graphs. Call the output on this input $\tilde{\mathbf{A}}_i$. Now let j be the second lowest choice of N on which the algorithm is correct. Since the algorithm was correct on $\mathbf{A}_i$ to within a multiplicative $(1+2\epsilon)$, to be correct on $\mathbf{A}_j$, it must output a set of edges $\tilde{\mathbf{A}}_j$ such that

$$(\mathbf{A}_j^T\mathbf{A}_j - \mathbf{A}_i^T\mathbf{A}_i) - 4\epsilon\mathbf{A}_j^T\mathbf{A}_j \preceq \tilde{\mathbf{A}}_j^T\tilde{\mathbf{A}}_j - \tilde{\mathbf{A}}_i^T\tilde{\mathbf{A}}_i \preceq (\mathbf{A}_j^T\mathbf{A}_j - \mathbf{A}_i^T\mathbf{A}_i) + 4\epsilon\mathbf{A}_j^T\mathbf{A}_j.$$

Since we double each successive copy of the complete graph, $\mathbf{A}_j^T\mathbf{A}_j \preceq 2(\mathbf{A}_j^T\mathbf{A}_j - \mathbf{A}_i^T\mathbf{A}_i)$. So, $\tilde{\mathbf{A}}_j^T\tilde{\mathbf{A}}_j - \tilde{\mathbf{A}}_i^T\tilde{\mathbf{A}}_i$ must be a $1+8\epsilon$ spectral approximation to the true difference $\mathbf{A}_j^T\mathbf{A}_j - \mathbf{A}_i^T\mathbf{A}_i$.

Noting that this difference is itself just a weighting of the complete graph, by the lower bound in [BSS12] the algorithm must select $\Omega(d/\epsilon^2)$ additional edges between the i^{th} and j^{th} input graphs. Iterating this argument over all $\lfloor \log(M\epsilon/\delta) \rfloor/2$ inputs on which the algorithm must be correct, it must select a total of $\Omega(d \log(M\epsilon/\delta)/\epsilon^2)$ edges in expectation over all inputs.

■

Chapter 4

Faster SDD Solvers I: Graph Clustering Algorithms

In the next two chapters, we present the fastest sequential algorithm for solving linear systems on graphs, and more generally symmetric diagonally dominant systems.

The Laplacian of a graph is defined to be $B^T B$, where B is the signed edge-vertex incidence matrix of the graph (cf. Figure 1.1). For an n-vertex graph, it is an $n \times n$ matrix containing the degrees of the vertices on the diagonal, and -1 in any row u and column v such that u and v are connected by an edge.

Spielman and Teng [ST04a] were the first to demonstrate that Laplacian systems can be solved in nearly linear time. After that, many approaches were developed in an attempt to bring the running time of the algorithms closer to linear. The approach we base on, of combinatorial preconditioning, was introduced by Koutis, Miller and Peng. The idea is to approximate the graph by a tree, and recursively make it sparser and more tree-like, while utilizing numerical preconditioning between levels of recursion. There are two bottlenecks that keep this algorithm from achieving runtime better than $\mathcal{O}(m \log n)$:

- Finding low stretch trees. The algorithm given by Abraham and Neiman [AN12] runs in $\mathcal{O}(m \log n)$ time.
- The iterative method in between recursive levels. The condition number between the graph and subsampled approximation is $\mathcal{O}(\log^2 n)$: $\mathcal{O}(\log n)$ factor comes from the stretch of the tree used, and one comes from the necessity to over-sample to handle the coupon collector problem.

In this and the following chapter, we show how to overcome both obstacles, leading to Theorem 5.1.1 [CKM$^+$14].

Theorem 5.1.1 *Given a graph G with m edges, a vector $\boldsymbol{b} = \boldsymbol{L}_G\boldsymbol{x}$, and any error $\epsilon > 0$, we can find w.h.p. a vector $\boldsymbol{x}$ such that*

$$\|\bar{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G} \leq \epsilon \|\bar{\boldsymbol{x}}\|_{\boldsymbol{L}_G},$$

in expected $\mathcal{O}(m \log^{1/2} n \log\log^{3+\delta} n \log(\frac{1}{\epsilon}))$ time for any constant $\delta > 0$.

4.0.1 Graph Clustering Algorithms

Low stretch trees are a fundamental construct for many spectral graph theoretic algorithms. The total stretch of a graph $G = (V, E)$ with respect to a tree T is defined as

$$\sum_{(u,v)\in E} d_T(u, v),$$

where $d_T(u, v)$ is the distance in T between vertices u and v. The average stretch is the total stretch divided by $m = |E|$. We can think of the average stretch as a measure of the quality of approximation of G by T.

The first algorithm for constructing low stretch trees was given by Alon, Karp, Peleg and West [AKPW95]. Another approach was proposed by Bartal [Bar96]. The output of that algorithm is not necessarily a subtree of the original graph, but is still dominated by it in a metric sense.

It turns out that every graph contains a spanning subtree with average stretch $\mathcal{O}(\log n \log\log n)$, as shown by Abraham and Neiman.

All of the aforementioned algorithms can be thought of as hierarchical clustering schemes, with various clustering algorithms being the main tool of the approach. The most prevalent one is low-diameter clustering. An extremely elegant randomized low-diameter clustering algorithm was given by Miller, Peng and Xu [MPX13]. In Chapter 6 we discuss generalizing it to a class of directed graphs.

In order to obtain a running time closer to linear, we will output trees that are *embeddable* into the original graph G; that is, the Steiner vertices of the tree can be mapped to vertices of G so that the result is a subgraph of the G. This is a slightly stronger condition that that provided by Bartal's algorithm. We also provide a slightly weaker guarantee on the stretch, namely, we produce trees for which the average *ℓ_p-stretch* is $\mathcal{O}(\log^p n)$, for a parameter $p \in (0, 1)$. The ℓ_p-stretch of a graph with respect to a tree is defined as:

$$\sum_{(u,v)\in E} d_T(u, v)^p.$$

Despite these relaxations, the produced trees are suitable for use in the iterative methods described in the next section.

We base our approach toward obtaining a faster low stretch tree construction algorithm on the top-down scheme proposed by Bartal. In order to increase the efficiency of our algorithm, we do not treat all the vertices as separate entities for each stage of the clustering; instead, we first find a coarse clustering using the bottom-up AKPW algorithm, which is then subject to refinement. The scheme can be summarized as follows:

1. Let $S_0 = V$. Compute the partitions $S_1, \ldots, S_k$, $|S_k| = 1$. S_i is computed by clustering the elements of S_{i-1} using a low-diameter decomposition algorithm.

2. Let d be the diameter of the graph. Cluster V into clusters of diameter $d/2$ using a low-diameter decomposition scheme. To improve runtime, use S_i as an initial partition, where i is chosen so that its clusters have diameter $\leq d/c$ ($c = \tilde{\mathcal{O}}(1)$). Recurse on the obtained clusters.

3. To compute the output tree, embed parts of the tree given by $S_0, \ldots, S_k$ into the decomposition computed in the previous step.

The guarantees of the algorithm are formalized in Theorem 4.1.1 below [CMP+14]:

Theorem 4.1.1 *Let $G = (V, E, d)$ be a weighted graph with n vertices and m edges. For any parameter p strictly between 0 and 1, we can construct a distribution over trees embeddable in G such that for any edge e its expected ℓ_p-stretch in a tree picked from this distribution is $\mathcal{O}((\frac{1}{1-p})^2 \log^p n)$. Furthermore, a tree from this distribution can be picked in expected $\mathcal{O}(\frac{1}{1-p} m \log\log n)$ time in the RAM model.*

The results of this section are joint work with Michael Cohen, Gary Miller, Richard Peng and Shen Chen Xu [CMP+14, CKM+14].

4.1 Introduction

Over the last few years substantial progress has been made on a large class of graph theoretic optimization problems. We now know substantially better asymptotic running time bounds and parallelizations for approximate undirected maximum flow/minimum cut [Mad10, CKM+11, LRS13, KLOS14, She13], bipartite matching [Mad13], minimum cost maximum flow [DS08], minimum energy flows [ST04a, KMP11, KOSZ13, CFM+14], and graph partitioning [She09, OSV12]. One commonality of all these new algorithms is that they either explicitly find low-stretch spanning trees or call an algorithm that at least at present uses these trees.

The fastest known algorithm for generating these trees, due to Abraham and Neiman runs in $\mathcal{O}(m \log n \log\log n)$ time [AN12]. Among the problems listed above, this running time is only the bottleneck for the minimum energy flow problem and its dual, solving symmetric diagonally dominant linear systems. However, there is optimism that all of the above problems can be solved in $o(m \log n)$ time, in which case finding these trees becomes a bottleneck as well. The main question we address in this chapter is finding algorithms for constructing even better trees in $\mathcal{O}(m)$ time. Unfortunately, this remains an open question.

The results presented remove the tree construction obstacle from $o(m \log n)$ time algorithms for solving SDD systems, as well as other graph optimization problems. We give two modifications to the definition of low stretch spanning trees that can simplify and speed up their construction. Firstly, we allow additional vertices in the tree, leading to a Steiner tree. This avoids the need for the complex graph decomposition scheme of [AN12]. Secondly, we discount the cost of high-stretch edges in ways that more accurately reflect how these trees are used. This allows the algorithm to be more "forgetful," and is crucial to our speedup.

Throughout this chapter we let $G = (V, E, l)$ be a graph with edge lengths $l(e)$, and $T = (V_T, E_T, l_T)$ denote the trees that we consider. In previous works on low stretch spanning trees, T was required to be a subgraph of G in the weighted sense. In other words, $E_T \subseteq E$, and $l_T(e) = l(e)$ for all $e \in E_T$. We relax this condition by only requiring edge lengths in T to be not too short with respect to G through the notion of embeddability, which we formalize in Section 4.2.

For a tree $T = (V_T, E_T, l_T)$, the stretch of an edge $e = uv$ with respect to T is

$$\mathbf{STR}_T(e) \stackrel{\text{def}}{=} \frac{l_T(u, v)}{l(e)},$$

where $l_T(u, v)$ is the length of the unique path between u and v in T. Previous tree embedding algorithms aim to pick a T such that the total stretch of all edges e in G is small [AKPW95, AN12]. A popular alternate goal is to show that the expected stretch of any edge is small, and these two definitions are closely related [AKPW95, CCG$^+$98] . Our other crucial definition is the discounting of high stretches by adopting the notion of ℓ_p-stretch:

$$\mathbf{STR}_T^p(e) \stackrel{\text{def}}{=} (\mathbf{STR}_T(e))^p\,.$$

These two definitional changes greatly simplify the construction of low stretch embeddings. It also allows the combination of existing algorithms in a robust manner. Our algorithm is based on the bottom-up clustering algorithm used to generate AKPW low-stretch spanning trees [AKPW95], combined with the top-down decompositions common in recent algorithms [Bar96, EEST08, ABN08, AN12]. Its guarantees can be stated as follows:

Theorem 4.1.1 *Let $G = (V, E, d)$ be a weighted graph with n vertices and m edges. For any parameter p strictly between 0 and 1, we can construct a distribution over trees embeddable in G such that for any edge e its expected ℓ_p-stretch in a tree picked from this distribution is $\mathcal{O}((\frac{1}{1-p})^2 \log^p n)$. Furthermore, a tree from this distribution can be picked in expected $\mathcal{O}(\frac{1}{1-p} m \log\log n)$ time in the RAM model.*

We will formally define embeddability, as well as other notations, in Section 4.2. An overview of our algorithm for generating low ℓ_p-stretch embeddable trees is in Section 6.2. We expand on it using existing low-stretch embedding algorithms in mostly black-box manners in Section 4.4. Then in Section 4.5 we show a two-stage algorithm that combines bottom-up and top-down routines that gives our main result.

Although our algorithm runs in $\mathcal{O}(m \log\log n)$ time, the running time is in the RAM model, and our algorithm calls a sorting subroutine. As sorting is used to approximately bucket the edge weights, this dependency is rather mild. If all edge lengths are between 1 and Δ, this process can be done in $\mathcal{O}(m \log(\log \Delta))$ time in the pointer machine model, which is $\mathcal{O}(m \log\log m)$ when $\Delta \leq m^{\text{poly}(\log m)}$. We suspect that there are pointer machine algorithms without even this mild dependence on Δ, and perhaps even algorithms that improve on the runtime of $\mathcal{O}(m \log\log n)$. Less speculatively, we also believe that our two-stage approach of combining bottom-up and top-down schemes can be applied with the decomposition scheme of [AN12] to generate actual spanning trees (as opposed to merely embeddable Steiner trees) with low ℓ_p-stretch. However, we do not have a rigorous analysis of this approach, which would presumably require a careful interplay with the radius-bounding arguments in that paper.

4.1.1 Related Works

Alon et al. [AKPW95] first proposed the notion of low stretch embeddings and gave a routine for constructing such trees. They showed that for any graph, there is a distribution over spanning trees such that the expected stretch of an edge is $\exp(\mathcal{O}(\sqrt{\log n \log\log n}))$. Subsequently, results with improved expected stretch were obtained by returning an arbitrary tree metric instead of a spanning tree. The only requirement on requirement on these tree metrics is that they don't shorten distances from the original graph, and they may also include extra vertices. However, in contrast to the objects constructed in this chapter, they do not necessarily fulfill the embeddability property. Bartal gave trees with expected stretch of $\mathcal{O}(\log^2 n)$ [Bar96], and $\mathcal{O}(\log n \log\log n)$ [Bar98]. Optimal trees with $\mathcal{O}(\log n)$ stretches are given by Fakcharoenphol et al. [FRT04], and are known as the FRT trees. This guarantee can be written formally as

$$\mathbb{E}_T\left[\mathbf{STR}_T(e)\right] \leq \mathcal{O}(\log n).$$

Recent applications to SDD linear system solvers has led to renewed interest in finding spanning trees with improved stretch over AKPW trees. The first LSSTs with $\text{poly}(\log n)$ stretch were given by Elkin et al. [EEST08]. Their algorithm returns a tree such that the expected stretch of an edge is $\mathcal{O}(\log^2 n \log\log n)$, which has subsequently been improved to $\mathcal{O}(\log n \log\log n(\log\log\log n)^3)$ by Abraham et al. [ABN08] and to $\mathcal{O}(\log n \log\log n)$ by Abraham and Neiman [AN12].

Notationally our guarantee is almost identical to the expected stretch above when p is a constant strictly less than 1:

$$\mathbb{E}_T\left[\mathbf{STR}_T^p(e)\right] \leq \mathcal{O}(\log^p n).$$

The power mean inequality implies that our embedding is weaker than those with ℓ_1-stretch bounds. However, at present, $O(\log n)$ guarantees for ℓ_1-stretch are *not known*–the closest is the result by Abraham and Neiman [AN12], which is off by a factor of $\log\log n$.

Structurally, the AKPW low-stretch spanning trees are constructed in a bottom-up manner based on repeated clusterings [AKPW95]. Subsequent methods are based on top down decompositions starting with the entire graph [Bar96]. Although clusterings are used implicitly in these algorithms, our result is the first that combines these bottom-up and top-down schemes.

4.1.2 Applications

The ℓ_p-stretch embeddable trees constructed in this chapter can be used in all existing frameworks that reduce the size of graphs using low-stretch spanning trees. In Appendix 4.A, we check that the larger graph with Steiner trees can lead to linear operators close to the graph Laplacian of the original graph. It allows us to use these trees in algorithms for solving linear systems in graph Laplacians, and in turn SDD linear systems. This analysis also generalizes to other convex norms, which means that our trees can be used in approximate flow [LS13, She13] and minimum cut [Mad10] algorithms.

Combining our algorithm with the recursive preconditioning framework by Koutis et al. [KMP11] leads to an algorithm that runs solves such a system to constant accuracy in $\mathcal{O}(m \log n)$ time. They are also crucial for the recent faster solver by Cohen et al. [CKP$^+$14], which runs in about $m \log^{1/2} n$ time. Parallelizations of it can be used can also lead to work-efficient parallel algorithms for solving SDD linear systems with depth of about $m^{1/3}$ [BGK$^+$13], and in turn for spectral sparsification [SS08, KLP12]. For these parallel applications, ignoring a suitable fraction of the edges leads to a simpler algorithm with lower depth. This variant is discussed in Section 4.5.3. On the other hand, these applications can be further improved by incorporating the recent polylog depth, nearly-linear work parallel solver by Peng and Spielman [PS13]. Consequently, we omit discussing the best bounds possible with the hope of a more refined parallel algorithm.

4.2 Background

Before we describe our algorithm, we need to formally specify the simple embeddability property that our trees satisfy. The notion used here is the same as the congestion/dilation definition widely used in routing [Lei92, LMR94]. It was used explicitly in earlier works on combinatorial preconditioning [Vai91, Gre96], and is implicit in the more recent algorithms.

Informally, an embedding generalizes the notion of a weighted subgraph in two ways. First, in an embedding of H into G, edges in H may correspond to paths in G, rather than just edges. Second, H may contain Steiner vertices that can be seen as "shadow copies" of vertices in G. Edges in G can be apportioned between different paths and connect to different Steiner vertices, but their weight must be reduced proportionally.

Formally, an embedding can be viewed as a weighted mapping from one graph to another. Splitting an edge will make it lighter, and therefore easier to embed. However, it will also make it harder to traverse, and therefore longer. As a result, for embeddings it is convenient to view an edge e by both its length $l(e)$ and weight $w(e)$, which is the reciprocal of its length:

$$w(e) \stackrel{\text{def}}{=} \frac{1}{l(e)}.$$

A path embedding is then a weighted mapping from the edges of a graph to paths in another. Such a mapping from a graph $H = (V_H, E_H, l_H)$ to a graph $G = (V_G, E_G, l_G)$ is given by the following three functions:

1. A mapping from vertices of H to those in G, $\pi : V_H \to V_G$.

2. A function from each edge $e_H \in E_H$ to a weighted path of G, denoted by $Path(e_H = x_G y_G)$ that goes from $\pi(x_G)$ to $\pi(y_G)$.

3. We let $W_{Path}(e_H, e_G)$ denote the weight of the edge e_G on path $Path(e_H)$. This value is zero if $e_G \notin Path(e_H)$.

The congestion-dilation notion of embeddability can then be formalized as follows:

Definition 4.2.1 *A graph H is* path embeddable*, or simply* embeddable*, into a graph G, if there exists a path embedding $(\pi, Path)$ of H into G such that:*

- *for all edges $e \in E_G$, $\sum_{e_H \in E_H} W_{Path}(e_H, e_G) \leq w_G(e_G)$: congestion is at most one, and*
- *for all edges $e_H \in E_H$, $\sum_{e_G \in Path(e_H)} \frac{1}{W_{Path}(e_H, e_G)} \leq l_H(e) = \frac{1}{w_H(e)}$: dilation is at most one.*

Note that since G has no self-loops, the definition precludes mapping both endpoints of an edge in H to the same point in G. Also note that if H is a subgraph of G such that $l_H(e) \geq l_G(e)$, setting π to be the identity function and $Path(e) = e$ and $W_{Path}(e, e) = w_H(e)$ is one way to certify embeddability.

4.3 Overview

We now give an overview of our main results. Our algorithm follows the decomposition scheme taken by Bartal for generating low stretch embeddings [Bar96]. This scheme partitions the graph repeatedly to form a laminar decomposition, and then constructs a tree from the laminar decomposition. However, our algorithm also makes use of spanning trees of the decomposition itself. As a result we start with the following alternate definition of Bartal decompositions where these trees are clearly indicated.

Definition 4.3.1 *Let $G = (V, E, l)$ be a connected multigraph. We say that a sequence of forests **B**, where*

$$\boldsymbol{B} = (B_0, B_1, \ldots, B_t),$$

is a Bartal decomposition *of G if all of the following conditions are satisfied:*

1. *B_0 is a spanning tree of G and B_t is an empty graph.*
2. *For any $i \leq t$, B_i is a subgraph of G in the weighted sense.*
3. *For any pair of vertices u, v and level $i < t$, if u and v are in the same connected component of B_{i+1}, then they are in the same connected component of B_i.*

Condition 2 implies that each of the B_is is embeddable into G. A strengthening of this condition would require the union of all the B_is to be embeddable into G. We will term such decompositions *embeddable Bartal decompositions*.

Bartal decompositions correspond to laminar decompositions of the graphs: if any two vertices u and v are separated by the decomposition in level i, then they are also separated in all levels $j > i$. If u and v are in the same partition in some level i, but are separated in level $i+1$, we say that u and v are first cut at level i. This definition is useful because if the diameters are decreasing, the stretch of an edge can be bounded using only information related to level at which it is first cut.

We will work with bounds on diameters, $\boldsymbol{d} = (d_0, \dots d_t)$. We say that such a sequence is geometrically decreasing if there exists some constant $0 < c < 1$ such that $d_{i+1} \leq c d_i$. Below we formalize a condition when such sequences can be used as diameter bounds for a Bartal decomposition.

Definition 4.3.2 *A geometrically decreasing sequence $\boldsymbol{d} = (d_0 \dots d_t)$ bounds the diameter of a Bartal decomposition $\mathcal{B}$ if for all $0 \leq i \leq t$,*

1. *The diameter of any connected component of B_i is at most d_i, and*
2. *any edge $e \in B_i$ has length $l(e) \leq \frac{d_i}{\log n}$.*

Given such a sequence, the bound d_i for the level where an edge is first cut dominates its final stretch. This motivates us to define the ℓ_p-stretch of an edge w.r.t. a Bartal decomposition as follows:

Definition 4.3.3 *Let $\boldsymbol{B}$ be a Bartal decomposition with diameter bounds bounds $\boldsymbol{d}$, and p a parameter such that $p > 0$. The ℓ_p-*stretch *with respect to $\boldsymbol{B}$, $\boldsymbol{d}$ of an edge e with length $l(e)$ that is first cut at level i is*

$$\boldsymbol{STR}^{p}_{\boldsymbol{B},\boldsymbol{d}}(e) \stackrel{\text{def}}{=} \left(\frac{d_i}{l(e)}\right)^p.$$

In Section 4.4, we will check rigorously that it suffices to generate (not necessarily embeddable) Bartal decompositions for which edges are expected to have small ℓ_p-stretch. We will give more details on these transformations later in the overview as well.

The decomposition itself will be generated using repeated calls to variants of probabilistic low-diameter decomposition routines [Bar96]. Such routines allow one to partition a graph into pieces of diameter d such that the probability of an edge being cut is at most $\mathcal{O}(\log n/d)$. At a high level, our algorithm first fixes a geometrically decreasing sequence of diameter bounds, then repeatedly decomposes all the pieces of the graph. With regular (ℓ_1) stretch, such routines can be shown to give expected stretch of about $\log^2 n$ per edge [Bar96], and most of the follow-up works focused on reducing this factor. With ℓ_p-stretch on the other hand, such a trade-off is sufficient for the optimum bounds when p is a constant bounded away from 1.

Lemma 4.3.4 *Let $\mathcal{B}$ be a distribution over Bartal decompositions. If $\boldsymbol{d}$ is a geometrically decreasing sequence that bounds the diameter of any $\boldsymbol{B} \in \mathcal{B}$, and the probability of an edge with length $l(e)$ being cut on level i of some $\boldsymbol{B} \in \mathcal{B}$ is*

$$\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right)$$

for some $0 < q < 1$. Then for any p such that $0 < p < q$, we have

$$\mathbb{E}_{\boldsymbol{B}\in\mathcal{B}}\left[\boldsymbol{STR}^{p}_{\boldsymbol{B},\boldsymbol{d}}(e)\right] \le \mathcal{O}\left(\frac{1}{q-p}\log^p n\right)$$

Its proof relies on the following fact about geometric series, which plays a crucial role in all of our analyses.

Fact 4.3.5 *There is an absolute constant c_{geo} such that if c and ϵ are parameters such that $c \in [e, e^2]$ and $\epsilon > 0$*

$$\sum_{i=0}^{\infty} c^{-i\epsilon} = c_{geo}\epsilon^{-1}.$$

Proof Since $0 < c^{-1} < 1$, the sum converges, and equals

$$\frac{1}{1-c^{-\epsilon}} = \frac{1}{1-\exp(-\epsilon \ln c)}.$$

Therefore it remains to lower bound the denominator. If $\epsilon \ge 1/4$, then the denominator can be bounded by a constant. Otherwise, $\epsilon \ln c \le 1/2$, and we can invoke the fact that $\exp(-t) \le 1 - t/2$ when $t \le 1/2$ to obtain

$$1 - \exp(-\epsilon \ln c) \ge \epsilon \ln c.$$

Substituting in the bound on c and this lower bound into the denominator then gives the result.

■

Proof of Lemma 4.3.4: If an edge is cut at a level with $d_i \le l(e)\log n$, its stretch is at most $\log n$, giving an ℓ_p-stretch of at most $\log^p n$. It remains only to consider the levels with $d_i \ge l(e)\log n$. Substituting the bounds of an edge cut on level i and the probability of it being cut into the definition of ℓ_p-stretch gives:

$$\begin{aligned}\mathbb{E}_{\mathbf{B}}\left[\mathbf{STR}^{p}_{\mathbf{B},\boldsymbol{d}}(e)\right] &\le \sum_{i, d_i \ge \log n l(e)} \left(\frac{d_i}{l(e)}\right)^p \mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right)\\ &= \mathcal{O}\left(\log^p n \sum_{i, d_i \ge \log n l(e)} \left(\frac{l(e)\log n}{d_i}\right)^{q-p}\right).\end{aligned}$$

Since an edge e is only cut in levels where $d_i \ge l(e)\log n$ and the d_is are geometrically increasing, this can be bounded by

$$\mathcal{O}\left(\log^p n \sum_{i=0} c^{-i(q-p)}\right)$$

Invoking Fact 4.3.5 then gives a bound of $\mathcal{O}\left(\frac{1}{q-p}\log^p n\right)$.

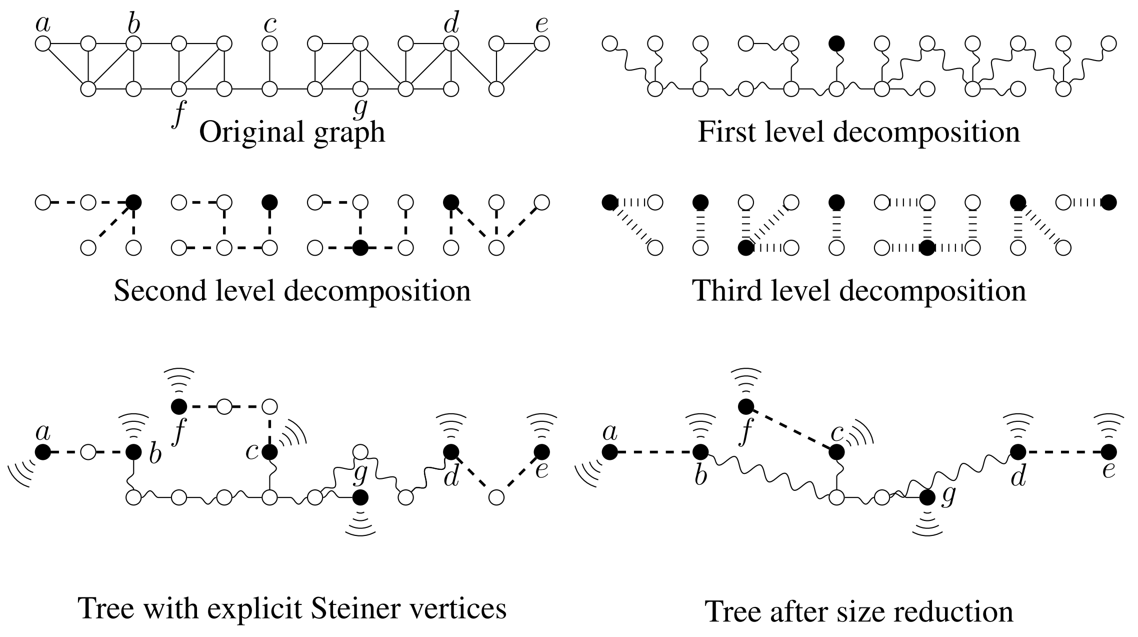

a
b
c
d
e
f
g
Original graph
First level decomposition
Second level decomposition
Third level decomposition
Tree with explicit Steiner vertices
Tree after size reduction

Figure 4.1: Bartal decomposition and final tree produced

■

This is our approach for showing that a Bartal decomposition has small ℓ_p-stretch, and it remains to convert them into embeddable trees. This conversion is done in two steps: we first show how to obtain a decomposition such that all of the B_is are embeddable into G, and then we give an algorithm for converting such a decomposition into a Steiner tree. To accomplish the former, we first ensure that each B_i is embeddable by choosing them to be subgraphs. Then we present pre-processing and post-processing procedures that converts such a guarantee into embeddability of all the B_is simultaneously.

In order to obtain a tree from the decomposition, we treat each cluster in the laminar decomposition as a Steiner vertex, and join them using parts of B_is. This step is similar to Bartal trees in that it identifies centers for each of the B_is, and connects the centers between one level and the next. However, the need for the final tree to be embeddable means that we cannot use the star-topology from Bartal trees [Bar96]. Instead, we must use part of the B_is between the centers. As each B_i is a forest with up to n edges, a tree obtained as such may have a much larger number of Steiner vertices. As a result, the final step involves reducing the size of this tree by contracting the paths connecting the centers. This process is illustrated in Figure 4.1.

In Section 4.4, we give the details on these steps that converts Bartal decompositions to embeddable trees. Furthermore, we check that Bartal's algorithm for generating such trees meets the good cutting probability requirements of Lemma 4.3.4. This then gives the following result:

Lemma 4.3.6 *Given a graph G with weights are between $[1, \Delta]$, for the diameter sequence $\mathbf{d}$ where $d_0 = 2n\Delta, d_1 = 2^{-1}n\Delta, \ldots d_t < 1$, we can create a distribution over Bartal decompositions with diameters bounded by $\mathbf{d}$ such that for any edge e and any parameter $0 < p < 1$,*

$$\mathbb{E}_{\boldsymbol{B}}\left[\boldsymbol{STR}^{p}_{\boldsymbol{B},\mathbf{d}}\left(e\right)\right] \leq \mathcal{O}\left(\frac{1}{1-p}\log^{p} n\right).$$

Furthermore, a random decomposition from this distribution can be sampled with high probability in $\mathcal{O}(m\log(n\Delta)\log n)$ time in the RAM model.

This routine plus the transformations gives a simple algorithm for constructing low ℓ_p-stretch embeddable trees. with expected stretch matching the bound stated our main result, Theorem 4.1.1. However, the running time of $\mathcal{O}(m\log(n\Delta)\log n)$ is more than the current best for finding low-stretch spanning trees [AN12], as well as the $\mathcal{O}(m\log^2 n)$ running time for finding Bartal trees.

Our starting point towards a faster algorithm is the difference between our simplified routine and Bartal's algorithm. Bartal's algorithm, as well as subsequent algorithms [EEST08] ensure that an edge participates in only $\mathcal{O}(\log n)$ partitions. At each step, they work on a graph obtained by contracting all edges whose lengths are less than $d_i/\text{poly}(n)$. This coupled with the upper bound of edge lengths from Definition 4.3.2, Part 2 and the geometric decrease in diameter bounds gives that each edge is involved in $\mathcal{O}(\log(\text{poly}(n))) = \mathcal{O}(\log n)$ steps of the partition.

As a path in the tree has at most n edges, the additive increase in stretch caused by these shrunken edges is negligible. Furthermore, the fact that the diameter that we partition upon decreases means that once we uncontract an edge, it remains uncontracted in all future steps. Therefore, these algorithms can start from the initial contraction for d_0, and maintain all contractions in work proportional to their total sizes.

When viewed by itself, this contraction scheme is almost identical to Kruskal's algorithm for building minimum spanning trees (MSTs). This suggests that the contraction sequence can be viewed as another tree underlying the top-down decomposition algorithm. This view also leads to the question of whether other trees can be used in place of the MST. In Section 4.5, we show that if the AKPW low-stretch spanning tree is used instead, each edge is expected to participate in $\mathcal{O}(\log\log n)$ levels of the top-down decomposition scheme. Combining this with a $\mathcal{O}(m\log\log n)$ time routine in the RAM model for finding the AKPW low-stretch spanning tree and a faster decomposition routine then leads to our faster algorithm.

Using these spanning trees to contract parts of the graph leads to additional difficulties in the post-processing steps where we return embeddable Steiner trees. A single vertex in the contracted graph may correspond to a large cluster in the original graph. As a result, edges incident to it in the decomposition may need to be connected by long paths. Furthermore, the total size of these paths may be large, which means that they need to be treated implicitly. In Section 4.5.5, we leverage the tree structure of the contraction to implicitly compute the reduced tree. Combining it with the faster algorithm for generating Bartal decompositions leads to our final result as stated in Theorem 4.1.1.

4.4 From Bartal Decompositions to Embeddable Trees

In this section, we show that embeddable trees can be obtained from Bartal decompositions using the process illustrated in Figure 4.1. We do this in three steps: exhibiting Bartal's algorithm in Section 4.4.1, showing that a decomposition routine that makes each B_i embeddable leads to a routine that generates embeddable decompositions in Section 4.4.2, and giving an algorithm for finding a tree from the decomposition in Section 4.4.3. We start by formally describing Bartal's algorithm for decomposing the graph.

4.4.1 Bartal's Algorithm

Bartal's algorithm in its simplest form can be viewed as repeatedly decomposing the graph so the pieces have the diameter guarantees specified by $\boldsymbol{d}$. At each step, it calls a low-diameter probabilistic decomposition routine with the following guarantees.

Lemma 4.4.1 (Probabilistic Decomposition) *There is an algorithm* PARTITION *that given a graph G with n vertices and m edges, and a diameter parameter d, returns a partition of V into $V_1 \uplus V_2 \uplus \ldots \uplus V_k$ such that:*

1. *The diameter of the subgraph induced on each V_i is at most d with high probability, certified by a shortest path tree on V_i with diameter d, and*
2. *for any edge $e = uv$ with length $l(e)$, the probability that u and v belong to different pieces is at most $\mathcal{O}(\frac{l(e)\log n}{d})$.*

Furthermore, PARTITION *can be implemented using one call to finding a single source shortest path tree on the same graph with all vertices connected to a super-source by edges of length between 0 and d.*

This routine was first introduced by Bartal to construct these decompositions. It and the low diameter decompositions that it's based on constructed each V_i in an iterative fashion. Miller et al. [MPX13] showed that a similar procedure can be viewed globally, leading to the implementation-independent view described above. Dijkstra's algorithm (Chapter 24 of [CSRL01]) then allows one to obtain a running time of $\mathcal{O}((m+n)\log n)$. It can be further sped up to $\mathcal{O}(m + n\log n)$ using Fibonacci heaps due to Fredman and Tarjan [FT87], and to $\mathcal{O}(m)$ in the RAM model by Thorup [Tho00]. In this setting where approximate answers suffice, a running time of $\mathcal{O}(m + n\log\log\Delta)$ was also obtained by Koutis et al. [KMP11]. As our faster algorithm only relies on the shortest paths algorithm in a more restricted setting, we will use the most basic $\mathcal{O}(m\log n)$ bound for simplicity.

We can then obtain Bartal decompositions by invoking this routine recursively. Pseudocode of the algorithm is given in Figure 4.1. The output of this algorithm for a suitable diameter sequence gives us the decomposition stated in Lemma 4.3.6.

```
$\mathbf{B} = \textsc{DecomposeSimple}(G, \boldsymbol{d})$, where $G$ is a multigraph, and $\boldsymbol{d}$ are diameter bounds.

  1. Initialize B by setting $B_0$ to a shortest path tree from an arbitrary vertex in $V_G$.
  2. For $i = 1 \dots t$ do
     (a) Initialize $B_i$ to empty.
     (b) Remove all edges $e$ with $l(e) \geq \frac{d_i}{\log n}$ from $G$.
     (c) For each subgraph $H$ of $G$ induced by a connected component of $B_{i-1}$ do
         i. $G_1 \dots G_k \leftarrow \textsc{Partition}(H, d_i)$.
         ii. Add shortest path trees in each $G_j$ to $B_i$.
  3. Return B.
```

Figure 4.1: Bartal's Decomposition Algorithm

Proof of Lemma 4.3.6: Consider the distribution produced by $\textsc{DecomposeSimple}(G, \boldsymbol{d})$. With high probability it returns a decomposition **B**. We first check that **B** is a Bartal decomposition. Each tree in B_is is a shortest path tree on a cluster of vertices formed by the partition. As these clusters are disjoint, B_i is a subgraph in the weighted sense. Since the algorithm only refines partitions, once two vertices are separated, they remain separated for any further partitions. Also, the fact that B_0 is spanning follows from the initialization step, and B_t cannot contain any edge since any edge has length at least 1 and $d_t < 1$.

We now show that $\boldsymbol{d}$ are valid diameter bounds for any decomposition produced with high probability. The diameter bounds on d_i follow from the guarantees of PARTITION and the initialization step. The initialization of $d_0 = 2n\Delta$ also ensures that no edge's length is more than $\frac{d}{\log n}$, and this invariant is kept by discarding all edges longer than $\frac{d}{c \log n}$ before each call to PARTITION.

The running time of the algorithm follows from $t \leq \mathcal{O}(\log(n\Delta))$ and the cost of the shortest path computations at all the steps. It remains to bound the expected ℓ_p-stretch of an edge e w.r.t. the decomposition. When $l(e) \geq \frac{d_i}{c \log n}$, a suitable choice of constants allows us to bound the probability of e being cut by 1. Otherwise, e will not be removed unless it is already cut. In case that it is in the graph passed onto PARTITION, the probability then follows from Lemma 4.4.1. Hence the cutting probability of edges satisfies Lemma 4.3.4, which gives us the bound on stretch.

■

4.4.2 Embeddability by Switching Moments

We now describe how to construct *embeddable* Bartal decomposition by using a routine that returns Bartal decompositions. This is done in three steps: pre-processing the graph to transform the edge lengths of G to form G', running the decomposition routine on G' for a different parameter q, and post-processing its output.

Pseudocode of this conversation procedure is given in Figure 4.2. Both the pre-processing and post-processing steps are deterministic, linear mappings. As a result, we can focus on bounding the expected stretch of an edge in the decomposition given by DECOMPOSE.

```
B, d = EMBEDDABLEDECOMPOSE($G, p, q$, DECOMPOSE$_q$) where $G$ is a graph, $p$, $q$ are
exponents for stretch, and DECOMPOSE is a routine that generates a decomposition $\mathbf{B}'$ along
with diameter bounds $\boldsymbol{d}'$.

    1. Create graph $G'$ with edge lengths $l'(e) = l(e)^{\frac{p}{q}}$.
    2. $\mathbf{B}', \boldsymbol{d}' \leftarrow$ DECOMPOSE$_q(G')$.
    3. Create decomposition $\mathbf{B}$ and diameter bounds $\boldsymbol{d}$ scaling lengths in $B'_i$ and $d_i$ by
           $\frac{c_{geo}}{q-p}\left(\frac{d'_i}{\log n}\right)^{\frac{q-p}{p}}$
       where $c_{geo}$ is the constant given by Fact 4.3.5.
    4. Return $\mathbf{B}, \boldsymbol{d}$.
```

Figure 4.2: Using a generic decomposition routine to generate an embeddable decomposition

We first verify that $\boldsymbol{d}$ is a geometrically decreasing sequence bounding the diameters of $\mathbf{B}$.

Lemma 4.4.2 *If $\mathbf{B}'$ is a Bartal decomposition of G' whose diameters are bounded by $\mathbf{d}'$, then $\mathbf{d}$ is geometrically decreasing sequence that bound the diameter of $\mathbf{B}$.*

Proof

The post-processing step scales the difference between adjacent d'_is by an exponent of $\frac{q}{p}$, which is at least 1 since $q > p$. Therefore $\boldsymbol{d}$ is also a geometrically decreasing sequence. As the lengths in B'_i and d'_i are scaled by the same factor, d_i remains an upper bound for the diameter of B_i. Also, since $d'_i \geq l'(e) \log n = l(e)^{\frac{p}{q}} \log n$, we have

$$d_i = \frac{c_{geo}}{q-p}\left(\frac{d'_i}{\log n}\right)^{\frac{q-p}{p}} d'_i \geq l(e) \log n.$$

Therefore $\boldsymbol{d}$ upper bounds the diameters of $\mathbf{B}$ as well.

■

We now check that $\mathbf{B}$ is a subgraph in the weighted case, which makes it an embeddable Bartal decomposition.

Lemma 4.4.3 *For any edge e we have*

$$\sum_i w_{B_i}(e) \leq w(e).$$

Proof Combining the pre-processing and post-processing steps gives that the total weight of e in all the layers is:

$$\sum_i w_{B_i}(e) = \sum_{i, e \in B_i} \frac{1}{l_{B_i}(e)} = \frac{p-q}{c_{geo}} \sum_{i, e \in B_i} \left(\frac{\log n}{d'_i} \right)^{\frac{q-p}{p}} w(e)^{\frac{p}{q}}.$$

Showing that this is at most $w(e)$ is therefore equivalent to showing

$$\frac{p-q}{c_{geo}} \sum_{i, e \in B_i} \left(\frac{\log n}{d'_i w(e)^{\frac{p}{q}}} \right)^{\frac{q-p}{p}} \leq 1.$$

Here we make use of the condition that the levels in which edge e appears have $d'_i \geq l'(e) \log n$. Substituting in $l'(e) = w(e)^{-\frac{p}{q}}$ into this bound on d'_i gives:

$$d'_i \geq w(e)^{-\frac{p}{q}} \log n$$
$$d'_i w(e)^{\frac{p}{q}} \geq \log n.$$

As d'_is are decreasing geometrically, this means that these terms are a geometrically decreasing sequence whose first term can be bounded by 1. Fact 4.3.5 then gives that the summation is bounded by $\frac{c_{geo} p}{q-p} \leq \frac{c_{geo}}{q-p}$, which cancels with the coefficient in front of it.

■

We can also check that the stretch of an edge e w.r.t. $\mathbf{B}$, $\boldsymbol{d}$ is comparable to its stretch in $\mathbf{B}'$, $\boldsymbol{d}'$.

Lemma 4.4.4 *For parameters $0 < p < q < 1$, the ℓ_p-stretch of an edge e in G w.r.t. $\boldsymbol{B}$, $\mathbf{d}$ and its ℓ_q-stretch in G' w.r.t. $\boldsymbol{B}'$, $\mathbf{d}'$ are related by*

$$\boldsymbol{STR}^p_{\boldsymbol{B}, \boldsymbol{d}}(e) = \mathcal{O}\left(\frac{1}{q-p} \log^{p-q} n \cdot \boldsymbol{STR}^q_{\boldsymbol{B}', \boldsymbol{d}'}(e) \right).$$

Proof Rearranging scaling on d'_i used to obtain d_i gives

$$d_i = \frac{c_{geo}}{q-p} \left(\frac{d'_i}{\log n} \right)^{\frac{q-p}{p}} d'_i = \frac{c_{geo}}{q-p} \log^{\frac{p-q}{p}} n \cdot d'^{\frac{q}{p}}_i.$$

We can then relate the stretch of an edge in the new decomposition with that of its ℓ_q-stretch in G'. For an edge cut at level i, we have

$$\mathbf{STR}_{\mathbf{B},\boldsymbol{d}}(e) = \frac{d_i}{l(e)} = \frac{c_{geo}}{q-p} \log^{\frac{p-q}{p}} n \frac{{d'_i}^{\frac{q}{p}}}{l'(e)^{\frac{q}{p}}}. = \frac{c_{geo}}{q-p} \log^{\frac{p-q}{p}} n \left(\mathbf{STR}^q_{\mathbf{B}',\boldsymbol{d}'}(e)\right)^{\frac{1}{p}}.$$

Taking both sides to the p-th power, and using the fact that $p < 1$, then gives the desired bound.

■

It's worth noting that when p and q are bounded away from 1 by constants, this procedure is likely optimal up to constants. This is because the best ℓ_p-stretch that one could obtain in these settings are $\mathcal{O}(\log^p n)$ and $\mathcal{O}(\log^q n)$ respectively.

4.4.3 From Decompositions to Trees

It remains to show that an embeddable decomposition can be converted into an embeddable tree. Our conversion routine is based on the laminar-decomposition view of the decomposition. From the bottommost level upwards, we iteratively reduce the interaction of each cluster with other clusters to a single vertex in it, which we term the centers. Centers can be picked arbitrarily, but to enforce the laminar decomposition view, we require that if a vertex u is a center on level i, it is also a center on level $i+1$ and therefore all levels $j > i$. Once the centers are picked, we can connect the clusters starting at the bottom level, by connecting all centers of level $i+1$ to the center of the connected component they belong to at level i. This is done by taking the part of B_i involving these centers. We first show that the tree needed to connect them has size at most twice the number of centers.

Fact 4.4.5 *Given a tree T and a set of k vertices S, there is a tree T_S on $2k-1$ vertices including these k leaves such that:*

- *The distances between vertices in S are the same in T and T_S.*
- *T_S is embeddable into T.*

Proof The proof is by induction on the number of vertices in T. The base case is when T has fewer than $2k-1$ vertices, where it suffices to set $T_S = T$. For the inductive case suppose the result is true for all trees with n vertices, and T has $n+1$ vertices. We will show that there is a tree T' on n vertices that preserves all distances between vertices in S, and is embeddable into T.

If T has a leaf that's not in S, removing it and the edge incident to it does not affect the distances between the vertices in S, and the resulting tree T' is a subgraph and therefore embeddable into T. Otherwise, we can check via a counting argument that there is a vertex u of degree 2 that's not in S. Let this vertex and its two neighbors be u and v_1, v_2 respectively. Removing u and adding an edge between v_1v_2 with weight $l(uv_1) + l(uv_2)$ preserves distances. This new tree T' is embeddable in T by mapping v_1v_2 to the path $v_1 - u - v_2$ with weights equaling the weights of the two edges.

Since T' has n vertices, the inductive hypothesis gives the existence of a tree T_S meeting the requirements. As T' is embeddable into T, T_S is embeddable into T as well.

■

Invoking this routine repeatedly on the clusters then leads to the overall tree. Pseudocode of this tree construction algorithm is given in Figure 4.3.

```
$T = $ BUILDTREE$(G, \mathbf{B})$ where $\mathbf{B}$ is a Bartal decomposition of $G$.

  1. Designate a center vertex for each connected component of each level of $\mathbf{B}$ such that
     if $u$ is a center vertex on level $i$, it is also a center vertex on level $i + 1$.

  2. For each connected component on level $i$.

     (a) Find all center vertices in level $i + 1$ contained in this piece.
     (b) Connect these vertices using the small sized equivalent of $B_i$ given by Fact 4.4.5.

  3. Return $T$.
```

Figure 4.3: Constructing a Steiner tree from a Bartal decomposition

Lemma 4.4.6 *Given a graph G and an embeddable Bartal decomposition **B**,* BUILDTREE *gives an embeddable tree T with $\mathcal{O}(n)$ vertices containing V such that for any geometrically decreasing sequence $\mathbf{d}$ that bounds the diameters of **B** and any edge e we have*

$$\boldsymbol{STR}_T(e) = \mathcal{O}(\boldsymbol{STR}_{\boldsymbol{B},\boldsymbol{d}}(e)).$$

Proof We first bound the total size of T. Note that the number of vertices added is proportional to the decrease in number of components. Since the initial number of clusters is n, T has at most $2n - 1$ vertices.

Fact 4.4.5 gives that the trees used to connect the level $i + 1$ clusters are embeddable into the corresponding connected component of B_i. Since the vertices in these clusters are disjoint and $\cup_i B_i$ is embeddable into G, T is also embeddable into G.

It remains to bound the stretch of edges w.r.t. T. For an edge $e = uv$ that's cut at level i, consider the the path from u to the centers of the clusters levels $t, t - 1, \ldots i$. The diameter bounds give that the distance traversed on level j is bounded by d_j. As $\boldsymbol{d}$ is a geometrically decreasing sequence, the total length of this path is bounded by $\mathcal{O}(d_i)$. A similar argument can be applied to v, and since u and v is cut at level i, the centers on level i are the same. Therefore, the distance between u and v in the tree can be bounded by $\mathcal{O}(d_i)$, giving the bound on stretch.

■

Combining these pieces leads to an algorithm generating low ℓ_p-stretch embeddable trees.

Lemma 4.4.7 *Let $G = (V, E, w)$ be a weighted graph with n vertices and m edges and weights $w : E \rightarrow [1, \Delta]$, and p be any parameter strictly between 0 and 1. We can construct a distribution* *over Bartal decompositions such that for any edge e, its expected ℓ_p-stretch in a decomposition picked from this distribution is $\mathcal{O}((\frac{1}{1-p})^2 \log^p n)$.*

Proof

Consider running EMBEDDABLEDECOMPOSE with $q = \frac{1+p}{2}$, and DECOMPOSESIMPLE with the parameters given by Lemma 4.3.6 as the decomposition procedure. By Lemmas 4.3.6 and 4.4.4, the expected stretch of an edge e in the post-processed decomposition **B** w.r.t. diameter bounds $\boldsymbol{d}$ is:

$$\mathcal{O}\left(\frac{1}{q-p}\log^{p-q}\frac{1}{1-q}\log^{q} n\right) = \mathcal{O}\left(\left(\frac{1}{1-p}\right)^2 \log^p n\right).$$

Running BUILDTREE on this decomposition then gives a tree where the expected stretch of edges are the same. The embeddability of this tree also follows from the embeddability of **B** given by Lemma 4.4.3.

To bound the running time, note that as $0 \leq \frac{p}{q} < 1$, the lengths of edges in the pre-processed graph G' are also between 1 and Δ. Both the pre and post processing steps consist of only rescaling edge weights, and therefore take linear time. The total running time then follows from Lemma 4.3.6.

■

4.5 Two-Stage Tree Construction

We now give a faster algorithm for constructing Bartal decompositions. The algorithm proceeds in two stages. We first quickly build a lower quality decomposition using the same scheme as the AKPW low stretch spanning tree [AKPW95]. Then we proceed in the same way as Bartal's algorithm and refine the decompositions in a top-down manner. However, with the first stage decomposition, we are able to construct a Bartal decomposition much faster.

Both the AKPW decomposition and the way that our Bartal decomposition routine uses it relies on repeated clustering of vertices. Of course, in an implementation, such clusterings will be represented using various linked-list structures. However, from an analysis perspective, it is helpful to view them as quotient graphs. For a graph G and a subset of edges A, we let the quotient graph G/A be the graph formed by the connected components of A. Each of these components corresponding to subsets of vertices becomes a single vertex in G/A, and the edges have their vertices relabeled accordingly. For our algorithms, it is essential for us to keep multi-edges as separate copies. As a result, all the graphs that we deal with in this section are potentially multi-graphs, and we will omit this distinction for simplicity.

The main advantages offered by the AKPW decomposition are

- it is a bottom-up routine that can be performed in linear time, and
- each edge only participates in $\mathcal{O}(\log\log n)$ steps of the refinement process in expectation, and

- all partition routines are done on graphs with diameter poly$(\log n)$.

The interaction between the bottom-up AKPW decomposition scheme and the top-down Bartal decomposition leads to some distortions. The rest of this section can be viewed as analyzing this distortion, and the algorithmic gains from having it. We will show that for an appropriately constructed AKPW decomposition, the probability of an edge being cut can be related to a quantity in the ℓ_q norm for some $p < q < 1$. The difference between these two norms then allows us to absorb distortions of size up to poly$(\log n)$, and therefore not affecting the quality of the resulting tree. Thus we will work mostly with a different exponent q in this section, and only bring things back to an exponent in p at the very end.

Both the AKPW and the top-down routines will issue multiple calls to PARTITION. In both cases the granularity of the edge weights will be poly$(\log n)$. As stated in Section 3, PARTITION can be implemented in linear time in the RAM model, using the rather involved algorithm presented in [Tho00]. In practice, it is also possible to use the low granularity of edge weights and use Dial's algorithm [Dia69], worsening the total running time of our algorithm to $\mathcal{O}(m \log\log n + \log \Delta \text{ poly}(\log n))$ when all edge lengths are in the range $[1, \Delta]$. Alternatively, we can use the weight-sensitive shortest path algorithm from [KMP11], which works in the pointer machine model, but would be slower by a factor of $\mathcal{O}(\log\log\log n)$.

4.5.1 The AKPW Decomposition Routine

We first describe the AKPW algorithm for generating decomposition. The decomposition produced is similar to Bartal decompositions, although we will not impose the strict conditions on diameters in our definition.

Definition 4.5.1 *Let $G = (V, E, l)$ be a connected multigraph. We say that a sequence of forests $\boldsymbol{A}$, where*

$$\boldsymbol{A} = (A_0, A_1, \dots, A_s),$$

is an AKPW decomposition *of G with parameter δ if:*

1. *A_s is a spanning tree of G.*
2. *For any $i < t$, $A_i \subseteq A_{i+1}$.*
3. *The diameter of each connected component in A_i is at most δ^{i+1}.*

Pseudocode for generating this decomposition is given in Figure 4.1. We first bound the diameters of each piece, and the probability of an edge being cut in A_i.

Lemma 4.5.2 AKPW(G, δ) *generates with high probability an AKPW decomposition $\boldsymbol{A}$ such that for an edge $e = uv$ with $l(e) \in [\delta^i, \delta^{i+1})$ and some $j \geq i$, the probability that u and v are not connected in A_j is at most*

$$\left(\frac{c_{Partition} \log n}{\delta}\right)^{j-i},$$

```
$\boldsymbol{A} = \mathrm{AKPW}(G, \delta)$, where $G$ is a connected multigraph.
  1. Bucket the edges by length into $E_0, E_1, \ldots$, where $E_i$ contains all edges of length in $[\delta^i, \delta^{i+1})$
  2. Initialize $A_0 := \emptyset$, $s := 0$.
  3. While $A_s$ is not a spanning tree of $G$:
     (a) Let $E'$ be the set of all edges from $E_0, \ldots, E_s$ that connect different components of $A_s$.
     (b) Set $G_s := (V, E', \vec{1})/A_s$, where $\vec{1}$ is a constant function that assigns all edges length 1.
     (c) Decompose $G$ by calling $\textsc{Partition}(G_s, \delta/3)$; let $T_1, T_2, \ldots, T_k$ be the edge sets of the corresponding low diameter spanning trees.
     (d) Set $A_{s+1} := A_s \cup T_1 \cup \ldots \cup T_k$
     (e) Set $s := s + 1$.
  4. Return $\boldsymbol{A} := (A_0, \ldots, A_s)$.
```

Figure 4.1: The routine for generating AKPW decompositions

where $c_{Partition}$ is a constant associated with the partition routine. Furthermore, if $\delta \geq 2c_{Partition} \log n$, it runs in expected $\mathcal{O}(m \log\log^{1/2} n)$ time in the RAM model,

Proof The termination condition on Line 3 implies that A_s is a spanning tree, and the fact that we generate A_{i+1} by adding edges to A_i gives $A_i \subseteq A_{i+1}$. The bound on diameter can be proven inductively on i.

The base case of $i = 0$ follows from the vertices being singletons, and as a result having diameter 0. For the inductive case, suppose the result is true for i. Then with high probability each connected component in A_{i+1} corresponds to a tree with diameter $\delta/3$ connecting connected components in A_i. The definition of E_i gives that each of these edges have length at most δ^{i+1}, and the inductive hypothesis gives that the diameter of each connected component in A_i is also at most δ^{i+1}. This allows us to bound the diameter of A_{i+1} by $(\delta/3) \cdot \delta^{i+1} + (\delta/3 + 1)\delta^{i+1} \leq \delta^{i+2}$. Hence the inductive hypothesis holds for $i + 1$ as well.

The guarantees of the probabilistic decomposition routine from Lemma 4.4.1 gives that on any level, an edge has its two endpoints separated with probability $\frac{c_P \log n}{\delta}$. The assumption of the length of e means that it is in E_i. So by the time A_j is formed, it has gone through $j - i$ rounds of partition, and is present iff its endpoints are separated in each of these steps. Multiplying the probabilities then gives the bound.

If $\delta \geq 2c_P \log n$, then the probability of an edge in E_i appearing in subsequent levels decrease geometrically. This means that the total expected sizes of G_t processed is $\mathcal{O}(m)$. Combining this with the linear running time of PARTITION gives the expected running time once we have the buckets E_0, E_1, etc. Under the RAM model of computation, these buckets can be formed in $\mathcal{O}(m \log \log^{1/2} n)$ time using the sorting algorithm by Han and Thorup [Han04]. Incorporating this cost gives the overall running time.

■

Combining the bound on diameter and probability of an edge being cut leads to the bound on the expected ℓ_1-stretch of an edge shown by Alon et al. [AKPW95]. For an edge on the i^{th} level, the ratio between its length and the diameter of the j^{th} level can be bounded by δ^{j-i+1}. As j increases, the expected stretch of e then increases by factors of

$$\delta \cdot \mathcal{O}\left(\frac{\log n}{\delta}\right) = \mathcal{O}\left(\log n\right),$$

which leads to the more than logarithmic bound on the expected ℓ_1-stretch. With ℓ_p-stretch however, the p^{th} power of the diameter-length ratio only increases by factors of δ^p. This means that, as long as the probabilities of an edge being cut increases by factors of less than δ^p, a better bound can be obtained.

4.5.2 AKPW meets Bartal

In this section, we describe how we combine the AKPW decomposition and Bartal's scheme into a two-pass algorithm. At a high level, Bartal's scheme repeatedly partitions the graph in a top-down fashion, and the choice of having geometrically decreasing diameters translates to a $\mathcal{O}(m \log n)$ running time. The way our algorithm achieves a speedup is by contracting vertices that are close to each other, in a way that does not affect the top-down partition scheme. More specifically, we precompute an appropriate AKPW decomposition, and only expose a limited number of layers while running the top-down partition. This way we ensure that each edge only appears in $\mathcal{O}(\log \log n)$ calls to the partition routine.

Let $\boldsymbol{A} = (A_0, A_1, \cdots, A_s)$ be an AKPW decomposition with parameter δ, so that G/A_i is the quotient graph where each vertex corresponds to a cluster of diameter at most δ^{i+1} in the original graph. While trying to partition the graph G into pieces of diameter d, where under some notion d is relatively large compared to δ^{i+1}, we observe that the partition can be done on the quotient graph G/A_i instead. As the complexity of our partition routine is linear in the number of edges, there might be some potential gain. We use the term *scope* to denote the point at which lower levels of the AKPW decomposition are handled at a coarser granularity. When the top-down algorithm is reaches diameter d_i in the diameter sequence $\boldsymbol{d}$, this cutoff point in the AKPW decomposition is denoted by $scope(i)$. The algorithm is formalized in Figure 4.2.

We first show that the increase in edge lengths to $\delta^{scope(i)+1}$ still allows us to bound the diameter of the connected components of B_i.

Lemma 4.5.3 *The diameter of each connected component in B_i is bounded by d_i with high probability.*

```
$\mathbf{B}$ = DecomposeTwoStage($G, \boldsymbol{d}, \boldsymbol{A}$), where $G$ is a graph, $\boldsymbol{d} = d_0, d_1, \dots, d_t$ is a
decreasing diameter sequence and $\boldsymbol{A} = (A_0, A_1, \dots A_s)$ is a fixed AKPW decomposition.

  1. Initialize $\mathbf{B}$ by setting $B_0$ to $A_s$
  2. For $i = 1 \dots t$ do
      (a) If necessary, increase $i$ so that $G' = B_{i-1}/A_{scope(i)}$ is not singletons.
      (b) Initialize $B_i$ to empty.
      (c) Increase all edge lengths to at least $\delta^{scope(i)+1}$ and remove all edges $e$ with
          $l(e) \geq \frac{d_i}{\log n}$ from $G'$.
      (d) For each connected component $H$ of $G'$ do
           i. $G_1 \dots G_k \leftarrow$ Partition($H, d_i/3$).
          ii. Add the edges in the shortest path tree in each $G_j$, plus the intermediate
              edges from $A_{scope(i)}$, to $B_i$.
  3. Return $\mathbf{B}$.
```

Figure 4.2: Pseudocode of two pass algorithm for finding a Bartal decomposition

Proof By the guarantee of the partition routine, the diameter of each G_i is at most $\frac{d_i}{3}$ with high probability. However, since we are measuring diameter of the components in G, we also need to account for the diameter of the components that were shrunken into vertices when forming G'. These components corresponds to connected pieces in $A_{scope(i)}$, therefore the diameters of the corresponding trees are bounded by $\delta^{scope(i)+1}$ with high probability. Our increase of edge lengths in G', on the other hand, ensures that the length of any edge is more than the diameter of its endpoints. Hence the total increase in diameter from these pieces is at most twice the length of a path in G', and the diameter of these components in G can be bounded by d_i.

■

Once we established that the diameters of our decomposition is indeed geometrically decreasing, it remains to bound the probability of an edge being cut at each level of the decomposition. In the subsequent sections, we give two different analyses of the algorithm DecomposeTwoStage with different choices of scope. We first present a simple version of our algorithm which ignores a $1/\text{poly}(\log n)$ fraction of the edges, but guarantees an expected ℓ_1-stretch close to $\mathcal{O}(\log n)$ for rest of the edges. Then we present a more involved analysis with a careful choice of scope which leads to a tree with small ℓ_p-stretch.

4.5.3 Decompositions that Ignore $\frac{1}{k}$ of the Edges

In this section, we give a simplified algorithm that ignores $\mathcal{O}(\frac{1}{k})$ fraction of the edges, but guarantees for other edges an expected ℓ_1-stretch of close to $\mathcal{O}(\log n)$. We also discuss how this relates to the problem of generating low-stretch subgraphs in parallel and its application to parallel SDD linear system solvers.

In this simplified algorithm, we use a naive choice of scope, reaching a small power of $k \log n$ into the AKPW decomposition.

Let $\boldsymbol{d} = (d_0, d_1, \dots, d_t)$ be a diameter sequence and let $\boldsymbol{A} = (A_0, A_1, \dots, A_s)$ be an AKPW decomposition constructed with parameter $\delta = k \log n$. We let $scope(i) = \max\{j \mid \delta^{j+3} \leq d_i\}$. Note that $\delta^{scope(i)}$ is always between $\frac{d_i}{\delta^4}$ and $\frac{d_i}{\delta^3}$. We say an edge $e \in E_i$ is ***AKPW-cut*** if e is cut in A_{i+1}. Furthermore, we say an edge e is ***floating in level*** i if it exists in $B_{i-1}/A_{scope(i)}$ and has length less than $\delta^{scope(i)+1}$. Note that the floating edges are precisely the edges whose length is increased before running the Bartal decomposition. We say that an edge is ***floating-cut*** if it is not AKPW-cut, but is cut by the Bartal decomposition at any level in which it is floating.

The simplification of our analysis over bounding overall ℓ_p stretch is that we can ignore all AKPW-cut or floating-cut edges. We start by bounding the expected number of edges ignored in these two ways separately.

Lemma 4.5.4 *Let $\boldsymbol{A} = AKPW(G, \delta)$ where $\delta = k \log n$. The expected number of AKPW-cut edges in $\boldsymbol{A}$ is at most $\mathcal{O}(\frac{m}{k})$.*

Proof For an edge $e \in E_i$, the probability that e is cut in A_{i+1} is at most

$$\frac{c_{Partition} \log n}{\delta} = \frac{c_{Partition}}{k}$$

by Lemma 4.5.2, where $c_{Partition}$ is the constant associated with the partition routine. Linearity of expectation then gives that the expected number of AKPW-cut edges is at most $\mathcal{O}(\frac{m}{k})$.

■

We now bound the total number of floating-cut edges:

Lemma 4.5.5 *The expected number of floating-cut edges is $\mathcal{O}(\frac{m}{k})$.*

Proof First, we note that only edges whose length is at least $\frac{d_i}{\delta^4}$ may be floating-cut at level i: any edge smaller than that length that is not AKPW-cut will not be contained in $B_{i-1}/A_{scope(i)}$. Furthermore, by the definition of floating, only edges of lengths at most $\frac{d_i}{\delta^2}$ may be floating. Therefore, each edge may only be floating-cut for levels with d_i between δ^2 and δ^4 times the length of the edge. Since the d_i increase geometrically, there are at most $\log(\delta)$ such levels.

Furthermore, at any given level, the probability that a given edge is floating-cut at the level is at most $\mathcal{O}(\frac{\log n}{\delta^2})$, since any floating edge is passed to the decomposition with length $\frac{d_i}{\delta^2}$. Taking a union bound over all levels with d_i between δ^2 and δ^4 times the length of the edge, each edge has at most a $\mathcal{O}(\frac{\log n \log \delta}{\delta^2})$ probability of being cut. Since $\frac{\log \delta}{\delta} = \mathcal{O}(1)$, this is $\mathcal{O}(\frac{\log n}{\delta}) = \mathcal{O}(\frac{1}{k})$.

Again, applying linearity of expectation implies that the expected number of floating-cut edges is $O(\frac{m}{k})$.

■

Combining these two bounds gives that the expected number of ignored edges so far is bounded by $\mathcal{O}(\frac{m}{k})$. We can also check that conditioned on an edge being not ignored, its probability of being cut on some level is the same as before.

Lemma 4.5.6 *Assume $\boldsymbol{A} = AKPW(G, \delta)$. We may associate with the output of the algorithm a set of edges S, with expected size $\mathcal{O}(\frac{m}{k})$, such that for any edge e with length $l(e)$, conditioned on $e \notin S$, is cut on the i^{th} level of the Bartal decomposition $\boldsymbol{B}$ with probability at most*

$$\mathcal{O}\left(\frac{l(e)\log n}{d_i}\right).$$

Proof We set S to the union of the sets of AKPW-cut and floating-cut edges.

Fix a level i of the Bartal decomposition: if an edge e that is not AKPW-cut or floating-cut appears in $B_{i-1}/A_{scope(i)}$, then its length is unchanged. If e is removed from G' due to $l(e) \geq d_i/\log n$, the bound becomes trivial. Otherwise, the guarantees of PARTITION then give the cut probability.

■

Lemma 4.5.7 *The simplified algorithm produces with high probability an embeddable Bartal decomposition with diameters bounded by $\boldsymbol{d}$ where all but (in expectation) $\mathcal{O}(\frac{m}{k})$ edges satisfy $\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{STR}_{\boldsymbol{B},\boldsymbol{d}}(e)] \leq \mathcal{O}(\log n(\log(k\log n))^2)$.*

Proof Let $p = 1 - 1/\log(k\log n)$ and $q = (1+p)/2$. Applying Lemma 4.5.6 and Lemma 4.3.4 we get that for edges not in S, $\mathbb{E}_{\mathbf{B}}[\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e)] = \mathcal{O}(\log^q n\log(k\log n))$. Then using EMBEDDABLEDECOMPOSE as a black box we obtain an embeddable decomposition with expected l_p-stretches of $\mathcal{O}(\log^p n(\log(k\log n))^2)$ for non-removed edges.

By repeatedly running this algorithm, in an expected constant number of iterations, we obtain an embeddable decomposition $\mathbf{B}$ with diameters bounded by $\boldsymbol{d}$ such that for a set of edges $E' \subseteq E$ and $|E'| \geq m - \mathcal{O}(\frac{m}{k})$ we have:

$$\sum_{e\in E'} \mathbb{E}_{\mathbf{B}}[\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e)] = \mathcal{O}(m\log^q n(\log(k\log n))^2).$$

By Markov's inequality, no more than a $1/k$ fraction of the edges in E' can have $\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e) \geq \mathcal{O}(k\log^q n(\log(k\log n))^2)$. This gives a set of edges E'' with size at least $m - \mathcal{O}(\frac{m}{k})$ such that any edge $e \in E''$ satisfies $\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e) \leq \mathcal{O}(k\log^q n(\log(k\log n))^2) \leq \mathcal{O}((k\log n)^2)$.

But for each of these edges

$$\begin{aligned}\mathbf{STR}_{\mathbf{B},\boldsymbol{d}}(e) &= (\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e))^{1/q}\\ &\leq (\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e))^{1+2/\log(k\log n)}\\ &\leq \mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e)\cdot \mathcal{O}\left((k\log n)^{4/\log(k\log n)}\right)\\ &= \mathcal{O}(\mathbf{STR}^q_{\mathbf{B},\boldsymbol{d}}(e)).\end{aligned}$$

Excluding these high-stretch edges, the ℓ_1 stretch is thus at most a constant factor worse than the ℓ_q stretch, and can be bounded by $\mathcal{O}(\log n(\log(k\log n))^2)$.

■

The total running time of DECOMPOSETWOSTAGE is dominated by the calls to PARTITION. The total cost of these calls can be bounded by the expected number of calls that an edge participates in.

Lemma 4.5.8 *For any edge e, the expected number of iterations in which e appears is bounded by $\mathcal{O}(\log(k\log n))$.*

Proof As pointed out in the proof of 4.5.5, an edge that is not AKPW-cut only appears in level i of the Bartal decomposition if $l(e) \in [\frac{d_i}{\delta^5}, \frac{d_i}{\log n})$. Since the diameters decrease geometrically, there are at most $\mathcal{O}(\log(k\log n))$ such levels. AKPW-cut edges can appear sooner than other edges from the same weight bucket, but using an argument similar to the proof of Lemma 4.5.4 we observe that the edge propagates up j levels in the AKPW decomposition with probability at most $(\frac{1}{k})^j$. Therefore the expected number of such appearances by an APKW-cut edge is at most $\sum_i (\frac{1}{k})^i = \mathcal{O}(1)$.

■

Combining all of the above we obtain the following result about our simplified algorithm. The complete analysis of its running time is deferred to Section 4.5.5.

Lemma 4.5.9 *For any k, given an AKPW decomposition $\boldsymbol{A}$ with $\delta = k\log n$, we can find in $\mathcal{O}(m\log(k\log n))$ time an embeddable Bartal decomposition such that all but expected $\mathcal{O}(\frac{m}{k})$ edges have expected total ℓ_1-stretch of at most $\mathcal{O}(m\log n(\log(k\log n))^2)$.*

Parallelization

If we relax the requirement of asking for a tree, the above analysis shows that we can obtain low stretch subgraphs edges and total stretch of $\mathcal{O}(\log n(\log(k\log n))^2)$ for all but $\mathcal{O}(\frac{m}{k})$ edges. As our algorithmic primitive PARTITION admits parallelization [MPX13], we also obtain a parallel algorithm for constructing low stretch subgraphs. These subgraphs are used in the parallel SDD linear system solver by [BGK$^+$13]. By observing that PARTITION is run on graphs with edge weights within δ of each other and hop diameter at most polynomial in $\delta = k\log n$, and invoking tree-contraction routines to extract the final tree [MR89], we can obtain the following result.

Lemma 4.5.10 *For any graph G with polynomially bounded edge weights and $k \leq poly(\log n)$, in $\mathcal{O}(k \log^2 n \log\log n)$ depth and $\mathcal{O}(m \log n)$ work we can generate an embeddable tree of size $\mathcal{O}(n)$ such that the total ℓ_1-stretch of all but $\mathcal{O}(\frac{m}{k})$ edges of G is $\mathcal{O}(m \log n (\log(k \log n))^2)$.*

4.5.4 Bounding Expected ℓ_p-Stretch of Any Edge

In this section we present our full algorithm and bound the expected ℓ_p-stretch of all edges Since we can no longer ignore edges whose lengths we increase while performing the top-down partition, we need to choose the scope carefully in order to control their probability of being cut during the second stage of the algorithm. We start off by choosing a different δ when computing the AKPW decomposition.

Lemma 4.5.11 *If A is generated by a call to* $\text{AKPW}(G, \delta)$ *with $\delta \geq (c_P \log n)^{\frac{1}{1-q}}$, then the probability of an edge $e \in E_i$ being cut in level j is at most $\delta^{-q(j-i)}$.*

Proof Manipulating the condition gives $c_P \log n \leq \delta^{1-q}$, and therefore using Lemma 4.5.2 we can bound the probability by

$$\left(\frac{c_P \log n}{\delta}\right)^{j-i} \leq \left(\frac{\delta^{1-q}}{\delta}\right)^{j-i} = \delta^{-q(j-i)}.$$

■

Since δ is $\text{poly}(\log n)$, we can use this bound to show that expected ℓ_p-stretch of an edge in an AKPW-decomposition can be bounded by $\text{poly}(\log n)$. The exponent here can be optimized by taking into account the trade-offs given in Lemma 4.3.4.

This extra factor of δ can also be absorbed into the analysis of Bartal decompositions. When $l(e)$ is significantly less than d, the difference between $\frac{l(e)\log n}{d}$ and $\left(\frac{l(e)\log n}{d}\right)^q$ is more than δ. This means that for an floating edge that originated much lower in the bucket of the AKPW decomposition, we can afford to increase its probability of being cut by a factor of δ.

From the perspective of the low-diameter decomposition routine, this step corresponds to increasing the length of an edge. This increase in length can then be used to bound the diameter of a cluster in the Bartal decomposition, and also ensures that all edges that we consider have lengths close to the diameter that we partition into. On the other hand, in order to control this increase in lengths, and in turn to control the increase in the cut probabilities, we need to use a different scope when performing the top-down decomposition.

Definition 4.5.12 *For an exponent q and a parameter $\delta \geq \log n$, we let the scope of a diameter d be*

$$scope(i) := \max_i \left\{ \delta^{i + \frac{1}{1-q} + 1} \leq d_i \right\}.$$

Note that for small d, $scope(i)$ may be negative. As we will refer to $A_{scope(i)}$, we assume that $A_i = \emptyset$ for $i < 0$. Our full algorithm can then be viewed as only processing the edges within the scope using Bartal's top-down algorithm. Its pseudocode is given in Figure 4.2.

Note that it is not necessary to perform explicit contraction and expansion of the AKPW clusters in every recursive call. In an effective implementation, they can be expanded gradually, as $scope(i)$ is monotonic in d_i.

The increase in edge lengths leads to increases in the probabilities of edges being cut. We next show that because the AKPW decomposition is computed using a higher norm, this increase can be absorbed, giving a probability that is still closely related to the p^{th} power of the ratio between the current diameter and the length of the edge.

Lemma 4.5.13 *Assume* $\boldsymbol{A} = \text{AKPW}(G, \delta)$ *with parameter specified as above. For any edge* e *with length* $l(e)$ *and any level* i*, the probability that* e *is cut at level* i *of* $\boldsymbol{B} = \text{DECOMPOSETWOSTAGE}(G, \boldsymbol{d}, \boldsymbol{A})$ *is*

$$\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right).$$

Proof There are two cases to consider based whether the length of the edge is more than $\delta^{scope(i)+1}$. If it is and it appears in G', then its length is retained. The guarantees of PARTITION then gives that it is cut with probability

$$\mathcal{O}\left(\frac{l(e)\log n}{d_i}\right) \leq \mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right),$$

where the inequality follows from $l(e)\log n \leq d_i$.

Otherwise, since we contracted the connected components in $A_{scope(i)}$, the edge is only cut at level i if it is both cut in $A_{scope(i)}$ and cut by the partition routine. Lemma 4.5.11 gives that if the edge is from E_j, its probability of being cut in $A_{scope(i)}$ can be bounded by $\delta^{-q(scope(i)-j)}$. Combining with the fact that $\delta^j \leq l(e)$ allows us to bound this probability by

$$\left(\frac{l(e)}{\delta^{scope(i)}}\right)^q.$$

Also, since the weight of the edge is set to $\delta^{scope(i)+1}$ in G', its probability of being cut by PARTITION is

$$\mathcal{O}\left(\frac{\delta^{scope(i)+1}\log n}{d_i}\right).$$

As the partition routine is independent of the AKPW decomposition routine, the overall probability can be bounded by

$$\mathcal{O}\left(\frac{\delta^{scope(i)+1}\log n}{d_i}\cdot\left(\frac{l(e)}{\delta^{scope(i)}}\right)^q\right)=\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\cdot\delta\log^{1-q}n\cdot\left(\frac{\delta^{scope(i)}}{d_i}\right)^{1-q}\right).$$

Recall from Definition 4.5.12 that $scope(i)$ is chosen to satisfy $\delta^{scope(i)+\frac{1}{1-q}+1}\le d_i$. This along with the assumption that $\delta\ge\log n$ gives

$$\delta\log^{1-q}n\cdot\left(\frac{\delta^{scope(i)}}{d_i}\right)^{1-q}\le\delta^{2-q}\left(\delta^{-\frac{2-q}{1-q}}\right)^{1-q}\le 1.$$

Therefore, in this case the probability of e being cut can also be bounded by $\mathcal{O}\left(\left(\frac{l(e)\log n}{d_i}\right)^q\right)$.

■

Combining this bound with Lemma 4.3.4 and setting $q=\frac{1+p}{2}$ gives the bound on ℓ_p-stretch.

Corollary 4.5.14 *If q is set to $\frac{1+p}{2}$, we have for any edge e $\mathbb{E}_{\boldsymbol{B}}[\boldsymbol{STR_{B,d}}(e)]\le\mathcal{O}(\frac{1}{1-p}\log^p n)$.*

Therefore, we can still obtain the properties of a good Bartal decomposition by only considering edges in the scope during the top-down partition process. On the other hand, this shrinking drastically improves the performance of our algorithm.

Lemma 4.5.15 *Assume $\boldsymbol{A}=\mathrm{AKPW}(G,\delta)$. For any edge e, the expected number of iterations of* DECOMPOSETWOSTAGE *in which e is included in the graph given to* PARTITION *can be bounded by $\mathcal{O}(\frac{1}{1-p}\log\log n)$.*

Proof Note that for any level i it holds that

$$\delta^{scope(i)}\ge d_i\delta^{-\frac{1}{1-q}-2}.$$

Since the diameters of the levels decrease geometrically, there are at most $\mathcal{O}(\frac{1}{1-q}\log\log n)$ levels i such that $l(e)\in[d_i\delta^{-\frac{1}{1-q}-2},\frac{d_i}{\log n})$.

The expected number of occurrences of e in lower levels can be bounded using Lemma 4.5.11 in a way similar to the proof of the above Lemma. Summing over all the levels i where e is in a lower level gives:

$$\sum_{i:l(e)<d_i\delta^{-\frac{1}{1-q}-2}}\left(\frac{l(e)}{\delta^{scope(i)}}\right)^q$$

Substituting in the bound on $\delta^{scope(i)}$ from above and rearranging then gives:

$$\leq \sum_{i: l(e) \leq d_i \delta^{-\frac{1}{1-q}-2}} \left(\frac{l(e)}{d_i} \delta^{\frac{1}{1-q}+2} \right)^q .$$

As d_i increase geometrically, this is a geometric sum with the first term at most 1. Therefore the expected number of times that e appears on some level i while being out of scope is $\mathcal{O}(1)$.

■

Recall that each call to PARTITION runs in time linear in the number of edges. This then implies a total cost of $\mathcal{O}(m \log\log n)$ for all the partition steps. We can now proceed to extract a tree from this decomposition, and analyze the overall running time.

4.5.5 Returning a Tree

We now give the overall algorithm and analyze its performance. Introducing the notion of scope in the recursive algorithm limits each edge to appear in at most $\mathcal{O}(\log\log n)$ levels. Each of these calls partitions G' in time linear in its size, which should give a total of $\mathcal{O}(m \log\log n)$. However, the goal of the algorithm as stated is to produce a Bartal decomposition, which has a spanning tree at each level. Explicitly generating this gives a total size of $\Omega(nt)$, where t is the number of recursive calls. As a result, we will circumvent this by storing only an implicit representation of the Bartal decomposition to find the final tree.

This smaller implicit representation stems from the observation that large parts of the B_is are trees from the AKPW decomposition, A_i. As a result, such succinct representations are possible if we have pointers to the connected components of A_i. We first analyze the quality and size of this implicit decomposition, and the running time for producing it.

```
$\mathbf{B}$, $\boldsymbol{d}$ = DECOMPOSE$(G, p)$, where $G$ is a graph, $p$ is an exponent

  1. Set $q = \frac{1+p}{2}$, $\delta = (c \log n)^{\frac{1}{q-p}}$.

  2. Compute an AKPW decomposition of $G$, $\boldsymbol{A} = \mathrm{AKPW}(G, \delta)$.

  3. Let $\boldsymbol{d} = (d_0, d_1, \cdots, d_t)$ be a geometrically decreasing sequence diameters where $d_0$
     is the diameter of $A_s$.

  4. Set $\mathbf{B}$ := DECOMPOSETWOSTAGE$(G, \boldsymbol{d}, \boldsymbol{A})$.

  5. Set $B_0$ to $A_s$.

  6. Return $\mathbf{B}$, $\boldsymbol{d}$.
```

Figure 4.3: Overall decomposition algorithm

Lemma 4.5.16 *There is a routine that for any graph G and and parameter $p < 1$, produces in expected $\mathcal{O}(\frac{1}{1-p} m \log\log n)$ time an implicit representation of a Bartal decomposition **B** with expected size $\mathcal{O}(\frac{1}{1-p} m \log\log n)$ and diameter bounds $\mathbf{d}$ such that with high probability:*

- ***B** is embeddable into G, and*
- *for any edge e, $\mathbb{E}_{\boldsymbol{B}}(\boldsymbol{STR}^p_{\boldsymbol{B},\boldsymbol{d}}(e)) \leq \mathcal{O}((\frac{1}{1-p})^2 \log^p n)$.*
- ***B** consist of edges and weighted connected components of an AKPW decomposition*

Proof Consider calling EMBEDDABLEDECOMPOSE from Section 4.4.2 with the routine given in Figure 4.3. The properties of **B** and the bounds on stretch follows from Lemma 4.4.4 and Corollary 4.5.14.

Since the number of AKPW components implicitly referred to at each level of the recursive call is bounded by the total number of vertices, and in turn the number of edges, the total number of such references is bounded by the size of the G's as well. This gives the bound on the size of the implicit representation.

We now bound the runnign time. In the RAM model, bucketing the edges and computing the AKPW decomposition can be done in $\mathcal{O}(m \log\log n)$ time. The resulting tree can be viewed as a laminar decomposition of the graph. This is crucial for making the adjustment in DECOMPOSETWOSTAGE in $\mathcal{O}(1)$ time to ensure that $A_{scope(i)}$ is disconnected. As we set q to $\frac{1+p}{2}$, by Lemma 4.5.15, each edge is expected to participate in $\mathcal{O}(\frac{1}{1-p} \log\log n)$ recursive calls, which gives a bound on the expected total.

The transformation of the edge weights consists of a linear-time pre-processing, and scaling each level by a fixed parameter in the post-post processing step. This process affects the implicit decomposition by changing the weights of the AKPW pieces, which is can be done implicitly in $\mathcal{O}(1)$ time by attaching extra 'flags' to the clusters.

■

It remains to show that an embeddable tree can be generated efficiently from this implicit representation. To do this, we define the notion of a contracted tree with respect to a subset of vertices, obtained by repeating the two combinatorial steps that preserve embeddability described in Section 4.2.

Definition 4.5.17 *We define the* contraction *of a tree T to a subset of its vertices S as the unique tree arising from repeating the following operations while possible:*

- *removal of a degree 1 vertex not in S, and*
- *contraction of a degree 2 vertex not in S.*

We note that it is enough to find contractions of the trees from the AKPW decomposition to the corresponding sets of connecting endpoints in the implicit representation. Here we use the fact that the AKPW decomposition is in fact a single tree.

Fact 4.5.18 *Let $\boldsymbol{A} = A_0, \ldots, A_s$ be an AKPW decomposition of G. Let S be a subset of vertices of G. For any i in $\{0, \ldots, s\}$, if S is contained in a single connected component of A_i, then the contraction of A_i to S is equal to the contraction of A_s to S.*

This allows us to use data structures to find the contractions of the AKPW trees to the respective vertex sets more efficiently.

Lemma 4.5.19 *Given a tree A_s on the vertex set V (with $|V| = n$) and subsets $S_1, \ldots, S_k$ of V of total size $\mathcal{O}(n)$, we can generate the contractions of A_s to each of the sets S_i in time $\mathcal{O}(n)$ in the RAM model and $\mathcal{O}(n\alpha(n))$ in the pointer machine model.*

Proof Root A_s arbitrarily. Note that the only explicit vertices required in the contraction of A_s to a set $S \subseteq V$ are

$$\Gamma(S) \stackrel{\text{def}}{=} S \cup \{LCA(u, v) : u, v \in S\}$$

where $LCA(u, v)$ denotes the lowest common ancestor of u and v in A_s. Moreover, it is easily verified that if we sort the vertices $v_1, \ldots, v_{|S|}$ of S according to the depth first search pre-ordering, then

$$\Gamma(S) = S \cup \{LCA(v_i, v_{i+1}) : 1 \leq i < |S|\}.$$

We can therefore find $\Gamma(S_i)$ for each i simultaneously in the following steps:

1. Sort the elements of each S_i according to the pre-ordering, using a single depth-first search traversal of A_s.
2. Prepare a list of lowest common ancestor queries for each pair of vertices adjacent in the sorted order in each set S_i.
3. Answer all the queries simultaneously using an off-line lowest common ancestor finding algorithm.

Since the total number of queries in the last step is $\mathcal{O}(n)$, its running time is $\mathcal{O}(n\alpha(n))$ in the pointer machine model using disjoint union [Tar79], and $\mathcal{O}(n)$ in the RAM model [GT83].

Once we find the sets $\Gamma(S_i)$ for each i, we can reconstruct the contractions of A_s as follows:

1. Find the full traversal of the vertices in $\Gamma(S_i)$ for each i, using a single depth first search traversal of A_s.
2. Use this information to reconstruct the trees [Vui80].

■

Applying this procedure to the implicit decomposition then leads to the final embeddable tree.

Proof of Theorem 4.1.1: Consider the distribution over Bartal decompositions given by Lemma 4.5.16. We will apply the construction given in Lemma 4.4.6, albeit in a highly efficient manner.

For the parts of the decomposition that are explicitly given, the routine runs in linear time. The more intricate part is to extract the smaller contractions from the AKPW components that are referenced to implicitly. Since all levels of the AKPW decomposition are subtrees of A_s, these are equivalent to finding contractions of A_s for several sets of vertices, as stated in Fact $4.5.18$. The algorithm given in Lemma 4.5.19 performs this operation in linear time. Concatenating these trees with the one generated from the explicit part of the decomposition gives the final result.

■

Appendix

4.A Sufficiency of Embeddability

In the construction of our trees, we made a crucial relaxation of only requiring embeddability, rather than restricting to subgraphs. In this section, we show that linear operators on the resulting graph can be related to linear operators on the original graph. Our analysis is applicable to ℓ_∞ flows as well.

The spectral approximation of two graphs can be defined in terms of their Laplacians. As we will interpret these objects combinatorially, we omit their definition and refer the reader to Doyle and Snell [DS84]. For matrices, we can define a partial ordering $\preceq$ where $A \preceq B$ if $B - A$ is positive semidefinite. That is, for any vector **x** we have

$$\mathbf{x}^T A \mathbf{x} \leq \mathbf{x}^T B \mathbf{b}.$$

If we let the graph formed by adding the tree to G be H, then our goal is to bound $\mathbf{L}_G$ and $\mathbf{L}_H$ with each other. Instead of doing this directly, it is easier to relate their pseudoinverses. This will be done by interpreting $\mathbf{x}^T \mathbf{L}^\dagger \mathbf{x}$ in terms of the energy of electrical flows. The energy of an electrical flow is defined as the sum of squares of the flows on the edges multiplied by their resistances, which in our case are equal to the lengths of the edges. Given a flow $f \in \Re^E$, we will denote its electrical energy using

$$\mathcal{E}_G(f) \stackrel{\text{def}}{=} \sum_e l_e f(e)^2.$$

The residue of a flow f is the net in/out flow at each vertex. This give a vector on all vertices, and finding the minimum energy of flows that meet a given residue is equivalent to computing $\mathbf{x}^T \mathbf{L}^\dagger \mathbf{x}$. The following fact plays a central role in the monograph by Doyle and Snell [DS84]:

Fact 4.A.1 *Let G be a connected graph. For any vector $\boldsymbol{x}$ orthogonal to the all ones vector, $\boldsymbol{x}^T \boldsymbol{L}_G^\dagger \boldsymbol{x}$ equals the minimum electrical energy of a flow with residue $\boldsymbol{x}$.*

Lemma 4.A.2 *Let $G = (V_G, E_G, w_G)$ and $H = (V_H, E_H, w_H)$ be graphs such that G is a subgraph of H in the weighted sense and $H \setminus G$ is embeddable in G. Furthermore, let the graph Laplacians of G and H be $\boldsymbol{L}_G$ and $\boldsymbol{L}_H$ respectively. Also, let Π be the $|V_G| \times |V_H|$ matrix with one 1 in each* *row at the position that vertex corresponds to in H and 0 everywhere else, and Π_1 the orthogonal projection operator onto the part of $\Re^{V_G}$ that's orthogonal to the all-ones vector. Then we have:*

$$\frac{1}{2}\boldsymbol{L}_G^{\dagger} \preceq \Pi_1 \Pi \boldsymbol{L}_H^{\dagger} \Pi^T \Pi_1^T \preceq \boldsymbol{L}_G^{\dagger}.$$

Proof Since $\Pi_1^T = \Pi_1$ projects out any part space spanned by the all ones vector, and is this precisely the null space of $\mathbf{L}_G$, it suffices to show the result for all vectors $\mathbf{x}_G$ orthogonal to the all-1s vector. These vectors are in turn valid demand vectors for electrical flows. Therefore, the statement is equivalent to relating the minimum energies of electrical flows routing $\mathbf{x}_G$ on G and $\Pi^T\mathbf{x}_G$ on H.

We first show that flows on H take less energy than the ones in G. Let $\mathbf{x}_G$ be any vector orthogonal to the all ones vector, and f_G^* be the flow of minimum energy in G that meets demand $\mathbf{x}_G$. Setting the same flow on the edges of $E(G)$ in H and 0 on all other edges yields a flow f_H. The residue of this flow is the same residue in V_G, and 0 everywhere else, and therefore equal to $\Pi^T\mathbf{x}_G$. Since G is a subgraph of H in the weighted sense, the lengths of these edges can only be less. Therefore the energy of f_H is at most the energy of f_G and we have

$$\mathbf{x}_G^T \Pi \mathbf{L}_H^{\dagger} \Pi^T \mathbf{x}_G \leq \mathcal{E}_H(f_H) \leq \mathcal{E}_G(f_G^*) = \mathbf{x}_G^T \mathbf{L}_G^{\dagger} \mathbf{x}_G.$$

For the reverse direction, we use the embedding of $H \setminus G$ into G to transfer the flow from H into G. Let $\mathbf{x}_G$ be any vector orthogonal to the all ones vector, and f_H^* the flow of minimum energy in H that has residue $\Pi^T\mathbf{x}_G$. This flow can be transformed into one in G that has residue $\mathbf{x}_G$ using the embedding. Let vertex/edge mapping of this embedding be π_V and π_E respectively.

If an edge $e \in E_H$ is also in E_G, we keep its flow value in G. Otherwise, we route its flow along the path that the edge is mapped to. Formally, if the edge is from u to v, $f_H(e)$ units of flow is routed from $\pi_V(u)$ to $\pi_V(v)$ along $path(e)$. We first check that the resulting flow, f_G has residue $\mathbf{x}_G$. The net amount of flow into a vertex $u \in V_G$ is

$$\sum_{uv \in E_G} f_H^*(e) + \sum_{u'v' \in E_H \setminus E_G, \pi_V(u') = u} f_H^*(e) = \sum_{uv \in E_G} f_H^*(e) + \sum_{u' \in V_H, \pi_V(u') = u} \left(\sum_{u'v' \in E_H \setminus E_G} f_H^*(e) \right).$$

Reordering the summations and noting that $\Pi(u) = u$ gives

$$= \sum_{u' \in V_H, \pi_V(u') = u} \sum_{u'v' \in E_H} f_H(e) = \sum_{u' \in V_H, \pi_V(u') = u} \left(\Pi^T \mathbf{x}_G\right)(e) = x_G(u).$$

The last equality is because $\pi_V(u) = u$, and all vertices not in V_G having residue 0 in $\Pi^T\mathbf{x}_G$.

To bound the energy of this flow, the property of the embedding gives that if split the edges of G into the paths that form the embedding, each edge is used at most once. Therefore, if we double the weights of G, we can use one copy to support G, and one copy to support the embedding. The energy of this flow is then the same. Hence there is an electrical flow f_G in G such that

$\mathcal{E}_G(f_G) \leq 2\mathcal{E}_H(f_H^*)$. Fact 4.A.1 then gives that it is an upper bound for $\mathbf{x}_G^T \mathbf{L}_G^\dagger \mathbf{x}_G$, completing the proof.

■

Chapter 5

Faster SDD Solvers II: Iterative Methods

The preconditioning step of the Koutis-Miller-Peng algorithm [KMP11] depends crucially on the following lemma:

Lemma 5.0.1 *Assume that a tree T satisfies $\boldsymbol{L}_T \preceq \boldsymbol{L}_G$. Let s be the total stretch of G w.r.t. T. Then, via sampling, we can efficiently construct a graph H with $\mathcal{O}(s \log n)$ edges that is a constant approximation to G.*

The $\log n$ factor in the above lemma comes from the coupon collector problem encountered during sampling. The algorithm given in the previous chapter lets us efficiently find a tree such that $s = \mathcal{O}(m \log n)$. Applying Lemma 5.0.1 directly to it would yield a graph with $\mathcal{O}(m \log^2 n)$ edges, which is of no use. However, we can simply add $\mathbf{L}_T \cdot \log^2 n$ to $\mathbf{L}_G$, obtaining a $\kappa = \mathcal{O}(\log^2 n)$ approximation G' to G. In turn, G' can be sparsified using Lemma 5.0.1. The gap between G' and G can be bridged using Chebyshev iteration, converging in $\mathcal{O}(\sqrt{\kappa})$ steps, which yields $\mathcal{O}(\log n)$ steps here.

How to improve this procedure? We do not yet know how to improve the quality of trees used to find the preconditioners; hence, we focus on the $\mathcal{O}(\log n)$ factor introduced by sampling in the result of Lemma 5.0.1.

The core idea is to forego the requirement for H to be a spectral approximation to G. Say we use H in a Richardson iteration preconditioning the system $\mathbf{L}_G x = b$. We can resample H in every step of the iteration, and all we need for our algorithm to work is for the error $\|\mathbf{L}_G^\dagger b - x\|_{\mathbf{L}_G}$ to reduce *in expectation*.

Of course, this necessitates for H to never be a *very bad*, say, disconnected, approximation to G. To ensure that, we always add a copy of T to H. Then, a sampling scheme mirroring that of KMP is employed. The analysis of the random graph obtained in this fashion is rather nontrivial, and builds on ideas introduced by Kelner, Orecchia, Sidford and Zhu [KOSZ13] in their combinatorial solver of Laplacian systems.

The results of this section are joint work with Michael Cohen, Rasmus Kyng, Richard Peng and Anup Rao [CKP+14, CKM+14].

5.1 Introduction

Randomized constructions of algebraically similar objects are widely used in the design of efficient algorithms. Sampling allows one to reduce the size of a problem while preserving its structure, and then solve the problem on a smaller instance. It is a core component in randomized matrix algorithms [Mah11], stochastic gradient descent [Bot04], and graph algorithms.

Smaller equivalents of graphs are known as *sparsifiers*, and the study of sampling methods for generating them led to the cut sparsifiers by Benczur and Karger [BK96], and spectral sparsifiers by Spielman and Teng [ST11]. Spectral sparsifiers are key routines in the first nearly-linear time solver by Spielman and Teng [ST04a], as well as in the subsequent improvements by Koutis et al. [KMP10, KMP11]. These solvers, in turn, have many applications which are described in detail in surveys by Spielman [Spi10] and Teng [Ten10].

At the core of the Spielman and Teng solver is a recursive preconditioning framework which transfers solutions between a sequence of sparsifiers known as a solver chain. Improvements to this framework led to algorithms that run in about $m \log n$ time under exact arithmetic [KMP11]. The existence of an algorithm that solves a given system in about $m \log^{1/2} n$ time after preprocessing can be derived from the nearly-optimal *ultra-sparsifiers* by Kolla et al. [KMST10]. These ultra-sparsifiers build upon the nearly-optimal spectral sparsifiers by Batson et al. [BSS09], and gain a factor of $\log^{1/2} n$ over randomized constructions. However, the current fastest algorithm for constructing these objects by Zouzias [Zou12] takes cubic time. As a result, finding nearly-linear time algorithms for constructing nearly-optimal sparsifiers and ultra-sparsifiers were posed as an important open question in the article by Batson et al. [BSST13].

Recently, a new approach to solving SDD linear systems was proposed by Kelner et al. [KOSZ13], and extended by Lee and Sidford [LS13]. Instead of constructing spectral sparsifiers, they show that fixing single cycles chosen from an appropriate distribution leads to sufficient decreases in errors *in expectation*. In this chapter, we extend this approach to more general subgraphs, and show that this achieves the same improvement per iteration as the optimal ultra-sparsifiers, *in expectation*. Our results can therefore be viewed as an algorithmic answer to the open question by Batson et al. [BSST13] on efficiently generating nearly-optimal sparsifiers.

Similar to the spectral sparsifiers by Batson et al. [BSS09], our results are applicable to general matrices. Instead of aiming to show that the sampled matrix is a good spectral approximation, our analysis is geared towards the intended application of the sample: use as a preconditioner for iterative methods for solving linear systems. We discuss these iterative methods and the statistical bounds needed for their convergence in Section 6.2. This randomized iterative method resembles to the randomized block Kaczmarz method by Needell and Tropp [NT13]. However, our convergence guarantees are more akin to those of standard iterative methods such as the ones presented in [Axe94].

For linear systems in Laplacians of graphs, our randomized iterative methods can be incorporated into existing solver frameworks. In Section 5.4, we use the recursive preconditioning framework by Koutis et al. [KMP11] to obtain the following result:

Theorem 5.1.1 *Given a graph G with m edges, a vector $\boldsymbol{b} = \boldsymbol{L}_G\boldsymbol{x}$, and any error $\epsilon > 0$, we can find w.h.p. a vector $\boldsymbol{x}$ such that*

$$\left\|\bar{\boldsymbol{x}} - \boldsymbol{x}\right\|_{\boldsymbol{L}_G} \leq \epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}_G},$$

in expected $\mathcal{O}(m \log^{1/2} n \log\log^{3+\delta} n \log(\frac{1}{\epsilon}))$ time for any constant $\delta > 0$.

In Appendix 5.B, we show that this solver can also be used to generate electrical flows with approximate minimum energy in similar time. This problem is dual to solving linear systems, and is the core problem addressed by previous solvers that reduce distance in expectation [KOSZ13, LS13].

Our presentation of the solver in Section 5.4 aims for simplicity, and we do not optimize for the exponent on $\log\log n$. This allows us to reduce to situations where errors of $\text{poly}(\log n)$ can be tolerated. Here we can use existing algorithms that are guaranteed to return good answers with high probability. We believe that this algorithmic dependency is removable, and that the exponent on $\log\log n$ can be reduced, or even entirely removed by a more refined analysis.

We also assume that all arithmetic operations are exact in this chapter. The iterative methods used in our algorithm, namely the preconditioned Chebyshev iteration in Appendix 5.A, are stated with robust bounds that can absorb large absolute error. Therefore, only the Gaussian elimination stages need to be checked to show the numerical stability of our algorithm in the setting of fixed-point arithmetic. Such an analysis of the recursive preconditioning framework can be found in Section 2.6 of [Pen13], and should be readily applicable to our algorithm as well.

5.2 Overview

Our starting point is the simplest iterative method, known as Richardson iteration. In the setting that we use it in, it can also be viewed as iterative refinement. If our goal is to solve a linear system $\boldsymbol{Y}\boldsymbol{x} = \boldsymbol{b}$, and we have a matrix $\boldsymbol{Z}$ that's similar to $\boldsymbol{Y}$, this method generates a new $\boldsymbol{x}'$ using the step

$$\boldsymbol{x}' = \boldsymbol{x} - \alpha\boldsymbol{Z}^{-1}\left(\boldsymbol{Y}\boldsymbol{x} - \boldsymbol{b}\right). \tag{5.1}$$

Here α is a parameter that we can choose based on the approximation factor between $\boldsymbol{Z}$ and $\boldsymbol{Y}$. When $\boldsymbol{Z}$ is an exact approximation, i.e. $\boldsymbol{Z} = \boldsymbol{Y}$, we can set $\alpha = 1$ and obtain

$$\boldsymbol{x}' = \boldsymbol{x} - \boldsymbol{Y}^{-1}\left(\boldsymbol{Y}\boldsymbol{x} - \boldsymbol{b}\right) = \boldsymbol{x} - \boldsymbol{x} + \boldsymbol{Y}^{-1}\boldsymbol{b} = \boldsymbol{Y}^{-1}\boldsymbol{b}.$$

Of course, in this situation we are simply solving $\boldsymbol{Z}\boldsymbol{x} = \boldsymbol{b}$ directly. In general, iterative methods are used when $\boldsymbol{Z}$ is an approximation of $\boldsymbol{Y}$. The quality of this approximation can be measured using relative condition numbers, which are defined using spectral orderings. While our main algorithm relies on a weaker notion of approximation, this view nonetheless plays a crucial role in its intermediate steps, as well as its analysis. Given two matrices $\boldsymbol{A}$ and $\mathbf{B}$, we say $\boldsymbol{A} \preceq \mathbf{B}$ if $\mathbf{B} - \boldsymbol{A}$ is positive semidefinite. Using this ordering, matrix approximations can then be defined by giving both upper and lower bounds. The guarantees of Richardson iteration under this notion of approximation is a fundamental result in iterative methods [Axe94].

Fact 5.2.1 *If $\boldsymbol{Y} \preceq \boldsymbol{Z} \preceq \kappa \boldsymbol{Y}$ for some parameter κ, and $\overline{\boldsymbol{x}}$ is the exact solution satisfying $\boldsymbol{Y}\overline{\boldsymbol{x}} = \boldsymbol{b}$, then taking the step in Equation 5.1 with $\alpha = \kappa$ gives:*

$$\left\|\boldsymbol{x}' - \overline{\boldsymbol{x}}\right\|_{\boldsymbol{Y}} \leq \left(1 - \frac{1}{\kappa}\right) \left\|\boldsymbol{x} - \overline{\boldsymbol{x}}\right\|_{\boldsymbol{Y}},$$

Here $\left\|\cdot\right\|_{\boldsymbol{Y}}$ is the matrix norm of $\boldsymbol{Y}$, $\left\|\cdot\right\|_{\boldsymbol{Y}} = \sqrt{\boldsymbol{x}^T \boldsymbol{Y} \boldsymbol{x}}$. It is the standard norm for measuring the convergence of iterative methods.

As Equation 5.1 requires us to solve a linear system involving $\boldsymbol{Z}$, it is desirable for $\boldsymbol{Z}$ to be smaller than $\boldsymbol{Y}$. One way to do this is to write $\boldsymbol{Y}$ as a sum of matrices, $\boldsymbol{Y} = \sum_{i=1}^{m} Y_i$, and pick a subset of these. This in turn can be done via random sampling. Here a crucial quantity is the statistical leverage score. For a matrix $\boldsymbol{X}$, the leverage score of $\boldsymbol{Y}_i$ w.r.t. $\boldsymbol{X}$ is

$$\overline{\boldsymbol{\tau}}_i \stackrel{\text{def}}{=} \mathbf{tr}\left[\boldsymbol{X}^{-1}\boldsymbol{Y}_i\right].$$

For some $\boldsymbol{X}$ and $\boldsymbol{Y} = \boldsymbol{Y}_1 + \ldots + \boldsymbol{Y}_m$, we can generate a preconditioner $\boldsymbol{Z}$ by sampling a number of Y_is with probabilities proportional to $\overline{\boldsymbol{\tau}}_i$. We can also use upper bounds on the actual leverage scores, $\boldsymbol{\tau}_i$. The pseudocode for a variant of this routine is given in Figure 5.1.

```
$Z = \textsc{Sample}(\{Y_1, \ldots, Y_m\}, X, \boldsymbol{\tau}, \delta)$, where $Y_i = \mathbf{v}_i\mathbf{v}_i^T$ are rank one matrices, $\boldsymbol{\tau}_i$ are upper
bounds of leverage scores, $\boldsymbol{\tau}_i \geq \overline{\boldsymbol{\tau}}_i$ for all $i$, and $\delta < 1$ is an arbitrary parameter.

  1. Initialize $Z$ to $X$.
  2. Let $s$ be $\sum_{i=1}^{m} \boldsymbol{\tau}_i$ and $t = \delta^{-1}s$.
  3. Pick an integer $r$ uniformly at random in the interval $[t, 2t-1]$.
  4. For $j = 1 \ldots r$
       (a) Sample entry $i_j$ with probability proportional to $\boldsymbol{\tau}_{i_j}$.
       (b) $Z \leftarrow Z + \frac{\delta}{\boldsymbol{\tau}_{i_j}} Y_{i_j}$.
  5. Return $Z$.
```

Figure 5.1: Sampling Algorithm

By applying matrix Chernoff bounds such as the ones by Tropp [Tro12], it can be shown that $\frac{1}{2}\boldsymbol{Y} \preceq \boldsymbol{Z} \preceq 2\boldsymbol{Y}$ when δ is set to $\frac{1}{O(\log n)}$. We will formalize this connection in Appendix 5.C. Scaling the resulting $\boldsymbol{Z}$ by a factor of 2 then gives a preconditioner that can be used to make the step given in Equation 5.1. The preconditioner produced contains $\boldsymbol{X}$ plus $O(s \log n)$ of the matrices $\boldsymbol{Y}_i$s. The Kolla et al. [KMST10] result can be viewed as finding $\boldsymbol{Z}$ consisting of only $O(s)$ of the matrices, and $\boldsymbol{Y} \preceq \boldsymbol{Z} \preceq O(1)\boldsymbol{Y}$, albeit in cubic time.

Our main result is showing that if we generate $\boldsymbol{Z}$ using SAMPLE with δ set to a constant, the step given in Equation 5.1 still makes a constant factor progress in expectation, for an appropriate constant α. We do so by bounding the first and second moments of $\boldsymbol{Z}^{-1}$ w.r.t. $\boldsymbol{Y}$. These bounds are at the core of our result. They are summarized in the following Lemma, and proven in Section 5.3.

Lemma 5.2.2 *Suppose $Y_i = \boldsymbol{v}_i\boldsymbol{v}_i^T$ are rank one matrices with sum Y, X is a positive semidefinite matrix satisfying $X \preceq Y$, $\boldsymbol{\tau}_1 \ldots \boldsymbol{\tau}_m$ are values that satisfy $\boldsymbol{\tau}_i \geq \textbf{tr}\left[X^{-1}Y\right]$, and $\delta < \frac{1}{3}$ is an arbitrary parameter. Then the matrix $Z = \text{SAMPLE}(Y_1 \ldots Y_m, X, \boldsymbol{\tau}_1 \ldots \boldsymbol{\tau}_m, \delta)$ satisfies:*

1. $\mathbb{E}_{r,i_1 \ldots i_r}\left[Z^{-1}\right] \preceq \frac{1}{1-2\delta}Y^{-1}$, *and*
2. $\mathbb{E}_{r,i_1 \ldots i_r}\left[Z^{-1}\right] \succeq \frac{1}{3}Y^{-1}$, *and*
3. $\mathbb{E}_{r,i_1 \ldots i_r}\left[Z^{-1}YZ^{-1}\right] \preceq \frac{1}{1-3\delta}Y^{-1}$.

Using these bounds, we can show that an iteration similar to Richardson iteration reduces errors, in expectation, by a constant factor each step.

Lemma 5.2.3 *Suppose $\boldsymbol{X}$ and $\boldsymbol{Y}$ are invertible matrices such that $\boldsymbol{X} \preceq \boldsymbol{Y}$, $\boldsymbol{b} = \boldsymbol{Y}\overline{\boldsymbol{x}}$, and $\boldsymbol{x}$ is an arbitrary vector. If $Z = \text{SAMPLE}(Y_1 \ldots Y_m, X, \boldsymbol{\tau}_1 \ldots \boldsymbol{\tau}_m, \frac{1}{10})$, and $\boldsymbol{x}'$ is generated using*

$$\boldsymbol{x}' = \boldsymbol{x} - \frac{1}{10}\boldsymbol{Z}^{-1}\left(\boldsymbol{Y}\boldsymbol{x} - \boldsymbol{b}\right).$$

Then

$$\mathbb{E}_{r,i_1,i_2,\ldots i_r}\left[\left\|\overline{\boldsymbol{x}} - \boldsymbol{x}'\right\|_{\boldsymbol{Y}}^2\right] \leq \left(1 - \frac{1}{40}\right)\left\|\overline{\boldsymbol{x}} - \boldsymbol{x}\right\|_{\boldsymbol{Y}}^2$$

Proof We first rearrange both sides by substituting in $\mathbf{b} = Y\mathbf{x}$, and letting $\mathbf{y} = \bar{\mathbf{x}} - \mathbf{x}$. The term in the LHS becomes

$$\bar{\mathbf{x}} - \left(\mathbf{x} - \frac{1}{10}Z^{-1}\left(Y\mathbf{x} - \mathbf{b}\right)\right) = \left(I - \frac{1}{10}Z^{-1}Y\right)\mathbf{y},$$

while the RHS becomes $\left(1 - \frac{1}{40}\right)\|\mathbf{y}\|_Y^2$.

Expanding the expression on the LHS and applying linearity of expectation gives

$$\begin{aligned}
&\mathbb{E}_{r,i_1,i_2,\ldots i_r}\left[\left\|\left(I - \frac{1}{10}Z^{-1}Y\right)\mathbf{y}\right\|_Y^2\right] \\
&= \mathbb{E}_{r,i_1,i_2,\ldots i_r}\left[\mathbf{y}^TY\mathbf{y} - \frac{2}{10}\mathbf{y}^TYZ^{-1}Y + \frac{1}{100}\mathbf{y}^TYZ^{-1}YZ^{-1}Y\mathbf{y}\right] \\
&= \mathbf{y}^TY\mathbf{y} - \frac{2}{10}\mathbb{E}_{r,i_1,i_2,\ldots i_r}\left[\mathbf{y}^TYZ^{-1}Y\mathbf{y}\right] + \frac{1}{100}\mathbb{E}_{r,i_1,i_2,\ldots i_r}\left[\mathbf{y}^TYZ^{-1}YZ^{-1}Y\mathbf{y}\right]
\end{aligned}$$

Since $Y\mathbf{y}$ is a fixed vector, we can apply Lemma 5.2.2 with it as $\mathbf{v}$. The lower bound on first moment in Part 1 allows us to upper bound the first term at

$$\begin{aligned}\mathbb{E}_{r,i_1,i_2,\dots i_r}\left[\mathbf{y}^TYZ^{-1}Y\mathbf{y}\right] &\geq \frac{1}{3}\mathbf{y}^TYY^{-1}Y\mathbf{y}\\ &= \frac{1}{3}\mathbf{y}^TY\mathbf{y}.\end{aligned}$$

The second term can be upper bounded using Part 3 with the same substitution.

$$\begin{aligned}\mathbb{E}_{r,i_1,i_2,\dots i_r}\left[\mathbf{y}^TYZ^{-1}YZ^{-1}Y\mathbf{y}\right] &\leq \frac{1}{1-3\delta}\mathbf{y}^TYY^{-1}Y\mathbf{y}\\ &= \frac{1}{1-3\delta}\mathbf{y}^TY\mathbf{y}\\ &\leq 2\mathbf{y}^TY\mathbf{y},\end{aligned}$$

where the last inequality follows from the choice of $\delta = \frac{1}{10}$. Combining these then gives the bound on the expected energy:

$$\begin{aligned}\mathbb{E}_{r,i_1,i_2,\dots i_r}\left[\left\|\bar{\mathbf{x}} - \left(\mathbf{x} - \frac{1}{10}Z^{-1}\left(Y\mathbf{x} - \mathbf{b}\right)\right)\right\|_Y^2\right] &\leq \|\mathbf{y}\|_Y^2 - \frac{2}{30}\mathbf{y}^TY\mathbf{y} + \frac{2}{100}\mathbf{y}^TY\mathbf{y}\\ &\leq \left(1 - \frac{1}{40}\right)\mathbf{y}^TY\mathbf{y}\end{aligned}$$

■

When $\boldsymbol{X}$ and $\boldsymbol{Y}$ are lower rank, we have that $\boldsymbol{Z}$ also acts on the same range space since $\boldsymbol{X}$ is added to it. Therefore, the same bound applies to the case where $\boldsymbol{X}$ and $\boldsymbol{Y}$ have the same null-space. Here it can be checked that the leverage score of $\boldsymbol{Y}_i$ becomes $\mathbf{tr}\left[\boldsymbol{X}^\dagger\boldsymbol{Y}_i\right]$, and the step is made based on pseudoinverse of $\boldsymbol{Z}$, $\boldsymbol{Z}^\dagger$. Also, note that for any nonnegative random variable x and moment $0 < p < 1$, we have $\mathbb{E}\left[x^p\right] \leq \mathbb{E}\left[x\right]^p$. Incorporating these conditions leads to the following:

Corollary 5.2.4 *Suppose $\boldsymbol{X}$ and $\boldsymbol{Y}$ are matrices with the same null space such that $\boldsymbol{X} \preceq \boldsymbol{Y}$, $\boldsymbol{b} = \boldsymbol{Y}\overline{\boldsymbol{x}}$, and $\boldsymbol{x}$ is an arbitrary vector. If $Z = \textsc{Sample}(Y_1 \dots Y_m, X, \boldsymbol{\tau}_1 \dots \boldsymbol{\tau}_m, \frac{1}{10})$, and $\boldsymbol{x}'$ generated using*

$$\boldsymbol{x}' = \boldsymbol{x} - \frac{1}{10}\boldsymbol{Z}^\dagger\left(\boldsymbol{Y}\boldsymbol{x} - \boldsymbol{b}\right).$$

Then

$$\mathbb{E}_{r,i_1,i_2,\dots i_r}\left[\left\|\bar{\boldsymbol{x}} - \left(\boldsymbol{x} - \frac{1}{10}Z^\dagger\left(Y\boldsymbol{x} - \boldsymbol{b}\right)\right)\right\|_Y\right] \leq \left(1 - \frac{1}{80}\right)\|\bar{\boldsymbol{x}} - \boldsymbol{x}\|_Y$$

5.3 Expected Inverse Moments

We now prove the bounds on $\boldsymbol{Z}^{-1}$ and $\boldsymbol{Z}^{-1}\boldsymbol{Y}\boldsymbol{Z}^{-1}$ stated in Lemma 5.2.2. For simplicity, we define $\mathbf{u}_j := Y^{\frac{-1}{2}}\mathbf{v}_j$, and $S := Y^{\frac{-1}{2}}XY^{\frac{-1}{2}}$. Note that, $\sum_{j=1}^{m}\mathbf{u}_j\mathbf{u}_j^T = I$, while

$$\begin{aligned}\mathbf{u}_i^T S^{-1}\mathbf{u}_i &= \mathbf{v}_i^T X^{-1}\mathbf{v}_i\\ &= \mathbf{tr}\left[X^{-1}\mathbf{v}_i\mathbf{v}_i^T\right]\\ &= \boldsymbol{\tau}_i.\end{aligned}$$

The following lemma is then equivalent to Lemma 5.2.2.

Lemma 5.3.1 *Suppose $R_i = \boldsymbol{u}_i\boldsymbol{u}_i^T$ are rank one matrices with $\sum_{j=1}^{m}\boldsymbol{u}_j\boldsymbol{u}_j^T = I$, S is a positive definite matrix satisfying $S \preceq I$ and $\boldsymbol{\tau}_1 \dots \boldsymbol{\tau}_m$ are values that satisfy $\boldsymbol{\tau}_i \geq$ **tr** $[S^{-1}R_i]$, and $0 < \delta < 1$ is an arbitrary parameter. Then the matrix $W = \textsc{Sample}(R_1 \dots R_m, S, \boldsymbol{\tau}_1 \dots \boldsymbol{\tau}_m, \delta)$ satisfies:*

1. $\mathbb{E}_{r,i_1\dots i_r}\left[\boldsymbol{x}^T W^{-1}\boldsymbol{x}\right] \geq \frac{1}{3}\boldsymbol{x}^T\boldsymbol{x}$, *and*
2. $\mathbb{E}_{r,i_1\dots i_r}\left[\boldsymbol{x}^T W^{-1}\boldsymbol{x}\right] \leq \frac{1}{1-2\delta}\boldsymbol{x}^T\boldsymbol{x}$, *and*
3. $\mathbb{E}_{r,i_1\dots i_r}\left[\boldsymbol{x}^T W^{-2}\boldsymbol{x}\right] \leq \frac{1}{1-3\delta}\boldsymbol{x}^T\boldsymbol{x}$.

In remainder of this section, we prove the above lemma. To analyze the SAMPLE algorithm, it will be helpful to keep track of its intermediate steps. Hence, we define W_0 to be the initial value of the sample sum matrix W. This corresponds to the initial value of Z from Line 5.2 in the pseudocode of Figure 5.1, and $W_0 = S$. We define W_j to be the value of W after j samples. Thus $W_{j+1} = W_j + \frac{\delta}{\boldsymbol{\tau}_{i_{j+1}}}\mathbf{u}_{i_{j+1}}\mathbf{u}_{i_{j+1}}^T$ where i_{j+1} is chosen with probability proportional to $\boldsymbol{\tau}_{j+1}$.

Throughout this section, we use δ to refer to the constant as defined in lemma 5.3.1 and let

$$t := \delta^{-1}\sum_{i=1}^{m}\boldsymbol{\tau}_i.$$

The following easily verifiable fact will be useful in our proofs.

Fact 5.3.2 *With variables as defined in lemma 5.3.1, each sample $\frac{\delta}{\boldsymbol{\tau}_{i_j}}\boldsymbol{u}_{i_j}\boldsymbol{u}_{i_j}^T$ obeys*

$$\mathbb{E}_{i_j}\left[\frac{\delta}{\boldsymbol{\tau}_{i_j}}\boldsymbol{u}_{i_j}\boldsymbol{u}_{i_j}^T\right] = \frac{1}{t}I$$

As we will often prove spectral bounds on the inverse of matrices, the following simple statement about positive definite matrices is very useful to us.

Fact 5.3.3 *Given positive definite matrices A and B where $A \preceq B$,*

$$B^{-1} \preceq A^{-1}.$$

The lower bound on $\boldsymbol{W}^{-1}$ can be proven using these two facts, and a generalization of the arithmetic mean (AM) - harmonic mean (HM) inequality for matrices by Sagae and Tanabe [ST94].

Lemma 5.3.4 (matrix AM-HM inequality, part of Theorem 1 of [ST94]) *If $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_r$ are positive numbers such that $\boldsymbol{w}_1 + \ldots + \boldsymbol{w}_r = 1$, and let $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_r$ be positive definite matrices. Then*

$$\left(\boldsymbol{w}_1 \boldsymbol{M}_1^{-1} + \ldots + \boldsymbol{w}_r \boldsymbol{M}_r^{-1}\right)^{-1} \preceq \boldsymbol{w}_1 \boldsymbol{M}_1 + \ldots + \boldsymbol{w}_r \boldsymbol{M}_r.$$

Proof of Lemma 5.3.1, Part 1: For all j, the matrix $\mathbf{u}_j \mathbf{u}_j^T$ is positive semidefinite. Hence, using the fact 5.3.2,

$$\begin{aligned} \mathbb{E}_{r,i_1,\ldots,i_r}\left[W\right] &\preceq \mathbb{E}_{i_1,\ldots,i_r}\left[W | r = 2t\right] \\ &= S + \sum_{j=1}^{2t} \mathbb{E}_{i_j}\left[\frac{\delta}{\boldsymbol{\tau}_{i_j}} \mathbf{u}_{i_j} \mathbf{u}_{i_j}^T\right] \preceq 3I \end{aligned}$$

Consequently, by the AM-HM bound from Lemma 5.3.4 gives

$$\mathbb{E}_{r,i_1,\ldots,i_r}\left[W^{-1}\right]^{-1} \preceq (3I)^{-1}.$$

Inverting both sides using Fact 5.3.3 gives the result.

■

We can now focus on proving the two upper bounds. One of the key concepts in our analysis is the harmonic sum, named after the harmonic mean,

$$\mathrm{HrmSum}\left(x, y\right) \overset{\mathrm{def}}{=} \frac{1}{1/x + 1/y}. \tag{5.2}$$

The following property of the harmonic sum plays a crucial role in our proof:

Fact 5.3.5 *If X is a positive random variable and $\alpha > 0$ is a constant, then*

$$\mathbb{E}\left[\mathrm{HrmSum}\left(X, \alpha\right)\right] \leq \mathrm{HrmSum}\left(\mathbb{E}\left[X\right], \alpha\right).$$

Proof Follows from Jensen's inequality since

$$\mathrm{HrmSum}\left(X, \alpha\right) = \frac{1}{\frac{1}{X} + \frac{1}{\alpha}} = \alpha \left(1 - \frac{\alpha}{X + \alpha}\right)$$

is a concave function in α when $\alpha > 0$.

■

We will also use a matrix version of this:

Fact 5.3.6 *For any unit vector* $\boldsymbol{v}$*, positive definite matrix* A*, and scalar* $\alpha > 0$

$$\boldsymbol{v}^T (A + \alpha I)^{-1} \boldsymbol{v} \leq \mathrm{HrmSum}\left(\boldsymbol{v}^T A^{-1} \boldsymbol{v}, 1/\alpha\right)$$

Proof By a change of basis if necessary, we can assume A is a diagonal matrix with positive entries $(a_1, ..., a_n)$ on its diagonal. Then $\mathbf{v}^T (A + \alpha I)^{-1} \mathbf{v} = \sum_{i=1}^{n} \frac{v_i^2}{a_i + \alpha} = \mathbb{E}\left[\mathrm{HrmSum}(X, \alpha)\right]$ where X is a random variable which satisfying $X = \frac{1}{a_i}$ with probability $\mathbf{v}_i^2$. Then by Fact 5.3.5 we have

$$\sum_{i=1}^{n} \frac{\mathbf{v}_i^2}{a_i + \alpha} = \mathbb{E}\left[\text{HrmSum}\left(X, \frac{1}{\alpha}\right)\right] \leq \text{HrmSum}\left(\mathbb{E}\left[X\right], \frac{1}{\alpha}\right) = \text{HrmSum}\left(\mathbf{v}^T A^{-1} \mathbf{v}, \frac{1}{\alpha}\right)$$

because $\mathbb{E}\left[X\right] = \sum_i \frac{\mathbf{v}_i^2}{a_i} = \mathbf{v}^T A^{-1} \mathbf{v}$.

■

Fact 5.3.7 *The function* $f_{H,\boldsymbol{v}}(x)$ *defined by*

$$f_{H,\boldsymbol{v}}(x) := \boldsymbol{v}^T \left(H + \frac{x}{t} I\right)^{-1} \boldsymbol{v}$$

is convex in x *for any fixed choices of vector* $\mathbf{v}$ *and positive definite matrix* H *.*

Proof By a change of basis, we can assume H to be diagonal matrix without loss of generality. Let its diagonal entries be $(a_1, ..., a_n)$. Since $\boldsymbol{H}$ is positive definite, $a_i > 0$. The result then follows from

$$f_{H,\mathbf{v}}(x) = \sum_i \frac{v_i^2}{a_i + \frac{x}{t}}$$

which is a convex function in x.

■

This implies that

$$\begin{aligned} \mathbf{v}^T W_i^{-1} \mathbf{v} + f'_{W_j,\mathbf{v}}(0) &= f_{W_j,\mathbf{v}}(0) + (1 - 0) f'_{W_j,\mathbf{v}}(0) \\ &\leq f_{W_j,\mathbf{v}}(1). \end{aligned}$$

Also, when $\mathbf{v}$ is a unit vector, we have by Fact 5.3.6:

$$f_{W_j,\mathbf{v}}(1) \leq \mathrm{HrmSum}(\mathbf{v}^T W_j^{-1} \mathbf{v}, t),$$

which rearranges to

$$f'_{W_j,\mathbf{v}}(0) \leq \mathrm{HrmSum}(\mathbf{v}^T W_j^{-1}\mathbf{v}, t) - \mathbf{v}^T W_j^{-1}\mathbf{v}.$$

Also, note that:

$$\begin{aligned} f'_{W_j,\mathbf{v}}(x) &= -\frac{1}{t}\mathbf{v}^T (W_j + (x/t)I)^{-2}\mathbf{v} \\ f'_{W_j,\mathbf{v}}(0) &= -\frac{1}{t}\mathbf{v}^T W_j^{-2}\mathbf{v}. \end{aligned}$$

So

$$-\frac{1}{t}\mathbf{v}^T W_j^{-2}\mathbf{v} \leq \mathrm{HrmSum}(\mathbf{v}^T W_j^{-1}\mathbf{v}, t) - \mathbf{v}^T W_j^{-1}\mathbf{v}. \tag{5.3}$$

We can also obtain a spectral lower bound W_{j+1}^{-1} in terms of W_j^{-1} and W_j^{-2}. using the Sherman-Morrison formula.

Lemma 5.3.8 $\mathbb{E}_{i_{j+1}}\left[W_{j+1}^{-1} | W_j\right] \preceq W_j^{-1} - \frac{(1-\delta)}{t} W_j^{-2}$

Proof

The Sherman-Morrison formula says that adding a single sample $\mathbf{z}_j\mathbf{z}_j^T := \frac{\delta}{\tau_{i_{j+1}}}\mathbf{u}_{i_{j+1}}\mathbf{u}_{i_{j+1}}^T$ to W_j gives:

$$(W_j + \mathbf{z}_j\mathbf{z}_j^T)^{-1} = W_j^{-1} - \frac{W_j^{-1}\mathbf{z}_j\mathbf{z}_j^T W_j^{-1}}{1 + \mathbf{z}_j^T W_j^{-1}\mathbf{z}_j}.$$

We then have

$$\begin{aligned} (W_j + \mathbf{z}_j\mathbf{z}_j^T)^{-1} &= W_j^{-1} - \frac{W_j^{-1}\mathbf{z}_j\mathbf{z}_j^T W_j^{-1}}{1 + \mathbf{z}_j^T W_j^{-1}\mathbf{z}_j} \\ &\preceq W_j^{-1} - \frac{W_j^{-1}\mathbf{z}_j\mathbf{z}_j^T W_j^{-1}}{1 + \delta} \\ &\preceq W_j^{-1} - (1-\delta) W_j^{-1}\mathbf{z}_j\mathbf{z}_j^T W_j^{-1}. \end{aligned}$$

Hence,

$$\begin{aligned} \mathbb{E}_{i_j}\left[(W_j + \mathbf{z}_j\mathbf{z}_j^T)^{-1} | W_j\right] &\preceq W_j^{-1} - (1-\delta) W_j^{-1}\mathbb{E}_{i_j}\left[\mathbf{z}_j\mathbf{z}_j^T\right] W_j^{-1} \\ &\preceq W_j^{-1} - \frac{(1-\delta)}{t} W_j^{-1} I W_j^{-1} \\ &\preceq W_j^{-1} - \frac{(1-\delta)}{t} W_j^{-2}. \end{aligned}$$

■

Combining these two bounds leads us to an upper bound for $\mathbb{E}\left[W_j^{-1}\right]$.

Proof of Lemma 5.3.1, Part 2: Combining Lemma 5.3.8 and Equation 5.3, we have

$$\begin{aligned}\mathbf{v}^T\mathbb{E}_{i_{j+1}}\left[(W_j+\mathbf{z}_{j+1}\mathbf{z}_{j+1}^T)^{-1}|W_j\right]\mathbf{v} &\le \mathbf{v}^TW_j^{-1}\mathbf{v}-\frac{1-\delta}{t}\mathbf{v}^TW_j^{-2}\mathbf{v}\\ &\le \mathbf{v}^TW_j^{-1}\mathbf{v}-(1-\delta)\left(\mathbf{v}^TW_j^{-1}\mathbf{v}-\mathrm{HrmSum}\left(\mathbf{v}^TW_j^{-1}\mathbf{v},t\right)\right)\\ &= \delta\mathbf{v}^TW_j^{-1}\mathbf{v}+(1-\delta)\,\mathrm{HrmSum}\left(\mathbf{v}^TW_j^{-1}\mathbf{v},t\right)\end{aligned}$$

If we now include the choice of W_j in the expectation:

$$\begin{aligned}\mathbb{E}_{i_1,\ldots,i_{j+1}}\left[\mathbf{v}^TW_{j+1}^{-1}\mathbf{v}\right] &\le \mathbb{E}_{i_1,\ldots,i_j}\left[\delta\mathbf{v}^TW_j^{-1}\mathbf{v}+(1-\delta)\,\mathrm{HrmSum}\left(\mathbf{v}^TW_j^{-1}\mathbf{v},t\right)\right]\\ &= \delta\mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^TW_j^{-1}\mathbf{v}\right]+(1-\delta)\mathbb{E}_{i_1,\ldots,i_j}\left[\mathrm{HrmSum}\left(\mathbf{v}^TW_j^{-1}\mathbf{v},t\right)\right].\end{aligned}$$

Applying Fact 5.3.5 with $X=\mathbf{v}^TW_j^{-1}\mathbf{v}$ and $a=t$ gives

$$\mathbb{E}_{i_1,\ldots,i_{j+1}}\left[\mathbf{v}^TW_{j+1}^{-1}\mathbf{v}\right]\le\delta\mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^TW_j^{-1}\mathbf{v}\right]+(1-\delta)\,\mathrm{HrmSum}\left(\mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^TW_j^{-1}\mathbf{v}\right],t\right). \quad (5.4)$$

For convenience, we define $E_j:=\mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^TW_j^{-1}\mathbf{v}\right]$. So inequality 5.4 can be written as

$$E_{i+1}\le\delta E_i+(1-\delta)\,\mathrm{HrmSum}\,(E_i,t)$$

Also, since we start with $W_0=Y^{\frac{-1}{2}}XY^{\frac{-1}{2}}$, we have $W_j\succeq Y^{\frac{-1}{2}}XY^{\frac{-1}{2}}$. Thus, by fact 5.3.3

$$W_j^{-1}\preceq(Y^{\frac{-1}{2}}XY^{\frac{-1}{2}})^{-1}=Y^{\frac{1}{2}}X^{-1}Y^{\frac{1}{2}}.$$

So $\mathbf{tr}\left[W_j^{-1}\right]\le\mathbf{tr}\left[Y^{\frac{1}{2}}X^{-1}Y^{\frac{1}{2}}\right]\le\sum_{i=1}^m\boldsymbol{\tau}_i=t\delta$, and we have $\mathbf{v}^TW_j^{-1}\mathbf{v}\le\left\|W_j^{-1}\right\|\le t\delta$, so $E_j\le t\delta<t$. This lets us write:

$$\begin{aligned}E_{j+1}&=\delta E_j+\frac{1-\delta}{\frac{1}{E_j}+\frac{1}{t}}\\ &=\frac{1+\frac{\delta E_j}{t}}{\frac{1}{E_j}+\frac{1}{t}}\\ &\le\frac{1}{\left(\frac{1}{E_j}+\frac{1}{t}\right)\left(1-\frac{\delta E_j}{t}\right)}\\ &=\frac{1}{\frac{1}{E_j}+\frac{1}{t}-\frac{\delta}{t}-\frac{\delta E_j}{t^2}}\\ &\le\frac{1}{1/E_j+(1-2\delta)/t}.\end{aligned}$$

So

$$\frac{1}{E_{j+1}} \geq \frac{1}{E_j} + (1-2\delta)/t$$

Then it follows by induction that after t steps

$$\frac{1}{E_j} \geq (1-2\delta).$$

Thus we have proved

$$\mathbb{E}_{i_1,\ldots,i_t}\left[\mathbf{v}^T W_t^{-1}\mathbf{v}\right] \leq \frac{1}{1-2\delta}. \tag{5.5}$$

Additionally, for any integer $r \geq t$, $W_r \succeq W_t$, so fact 5.3.3 gives $W_r^{-1} \preceq W_t^{-1}$. This means that with r chosen uniformly at random in the interval $[t, 2t-1]$, we have

$$\mathbb{E}_{r,i_1,\ldots,i_r}\left[\mathbf{v}^T W_r^{-1}\mathbf{v}\right] \leq \frac{1}{1-2\delta}.$$

■

It remains to upper bound W_r^{-2}. Here we use the same proof technique in reverse, by showing that the increase in W_r^{-1} is related to W_r^{-2}. Lemma 5.3.1, Part 2 gives that the total increase between t and $2t-1$ is not too big. Combining this with the fact that we chose r randomly gives that the expected increase at each step, and in turn the expected value of W_r^{-2} is not too big as well.

Proof of Lemma 5.3.1, Part 3: Recall that the expected value of $\mathbf{v}^T W_{j+1}^{-1}\mathbf{v} - \mathbf{v}^T W_j^{-1}\mathbf{v}$, conditional on W_j, was at most

$$\mathbf{v}^T W_{j+1}^{-1}\mathbf{v} - \mathbf{v}^T W_j^{-1}\mathbf{v} \leq (1-\delta) f'(0) = \frac{-(1-\delta)}{t}\mathbf{v}^T W_j^{-2}\mathbf{v}$$

Taking expectation over everything gives:

$$\mathbb{E}_{i_1,\ldots,i_{j+1}}\left[\mathbf{v}^T W_{j+1}^{-1}\mathbf{v}\right] - \mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^T W_j^{-1}\mathbf{v}\right] \leq \mathbb{E}_{i_1,\ldots,i_j}\left[\frac{-(1-\delta)}{t}\mathbf{v}^T W_j^{-2}\mathbf{v}\right]$$

Telescoping this gives

$$\mathbb{E}_{i_1,\ldots,i_{2t}}\left[\mathbf{v}^T W_{2t-1}^{-1}\mathbf{v}\right] - \mathbb{E}_{i_1,\ldots,i_t}\left[\mathbf{v}^T {W_t}^{-1}\mathbf{v}\right] \leq \sum_{j=t}^{2t-1}\mathbb{E}_{i_1,\ldots,i_j}\left[\frac{-(1-\delta)}{t}\mathbf{v}^T W_j^{-2}\mathbf{v}\right]$$

$$\frac{1}{t}\sum_{j=t}^{2t-1}\mathbb{E}_{i_1,\ldots,i_j}\left[\mathbf{v}^T W_j^{-2}\mathbf{v}\right] \leq \frac{1}{1-\delta}\mathbb{E}_{i_1,\ldots,i_t}\left[\mathbf{v}^T W_t^{-1}\mathbf{v}\right] \leq \frac{1}{(1-2\delta)(1-\delta)},$$

where the last inequality follows from equation 5.5. This implies that for an integer r chosen uniformly at random in the interval $[t, 2t-1]$, we have

$$\mathbb{E}_{r,i_1,\ldots,i_r}\left[\mathbf{v}^T W_r^{-2}\mathbf{v}\right] \leq \frac{1}{(1-2\delta)(1-\delta)} < \frac{1}{1-3\delta}.$$

■

5.4 Application to Solving SDD linear systems

We now describe a faster algorithm for solving SDD linear systems that relies on preconditioners that make progress in expectation. The reduction from solving these systems to solving graph Laplacians of doubled size was first shown by Gremban and Miller [Gre96]. This reduction is also well-understood for approximate solvers [ST06], and in the presence of fixed point round-off errors [KOSZ13]. As a result, we only address solving graph Laplacians in our presentation.

The Laplacian of a weighted graph G is an $n \times n$ matrix containing the negated weights in the off-diagonal entries and weighted degrees in the diagonal entries:

Definition 5.4.1 *The graph Laplacian $\boldsymbol{L}_G$ of a weighted graph $G = (V, E, \boldsymbol{w})$ with n vertices is an $n \times n$ matrix whose entries are:*

$$\boldsymbol{L}_{G,uv} = \begin{cases} \sum_{v \neq u} \boldsymbol{w}_{uv} & \text{if } u = v, \\ -\boldsymbol{w}_{uv} & \text{otherwise.} \end{cases}$$

The recursive preconditioning framework due to Spielman and Teng extends the ideas pioneered by Vaidya [Vai91]. It generates graph preconditioners, called *ultra-sparsifiers*, by sampling a number of edges to supplement a carefully chosen spanning tree. Using the notation introduced in Section 6.2, this corresponds to setting $\boldsymbol{X}$ to the graph Laplacian of the tree and the $\boldsymbol{Y}_i$s to the graph Laplacians of the off-tree edges.

The key connection between the statistical leverage score of a tree and combinatorial stretch of an edge was observed by Spielman and Woo [SW09].

Fact 5.4.2 *The statistical leverage score of the rank-1 matrix corresponding to an edge w.r.t. a tree is equal to its combinatorial stretch w.r.t. that tree.*

The reason that it is crucial to pick $\boldsymbol{X}$ to be a tree is that then the sizes of the recursive subproblems only depend on the number of $\boldsymbol{Y}_i$'s considered within. Similar to previous solvers, our algorithm is recursive. However, it chooses a different graph at each iteration, so that many distinct graphs are given in calls at the same level of the recursion. As a result, we will define an abstract Laplacian solver routine for our analyses.

Definition 5.4.3 *A routine* SOLVER$(\cdot)$ *is said to be a* Laplacian solver *when it takes as input a tuple $(G, T, \boldsymbol{\tau}, \boldsymbol{b}, \epsilon)$, where G is a graph, T a spanning tree of this graph, and $\boldsymbol{\tau}$ upper bounds on the* *combinatorial stretch of the off-tree edges of G wrt. T, and the routine returns as output a vector $\boldsymbol{x}$ such that*

$$\left\|\boldsymbol{x} - \boldsymbol{L}_G^\dagger \boldsymbol{b}\right\|_{\boldsymbol{L}_G} \leq \epsilon \left\|\boldsymbol{L}_G^\dagger \boldsymbol{b}\right\|_{\boldsymbol{L}_G}.$$

The following lemma about size reduction can be derived from partial Cholesky factorization. A detailed proof of it can be found in Appendix C of [Pen13].

Lemma 5.4.4 *Given a graph-tree tuple $(H, T, \boldsymbol{\tau})$ with n vertices and m' off-tree edges, and and a Laplacian solver* SOLVER*, there is a routine* ELIMINATE&SOLVE$(H, T, \boldsymbol{\tau}, \text{SOLVER}, \boldsymbol{b}, \epsilon)$ *that for any input $\boldsymbol{b} = \boldsymbol{L}_H \bar{\boldsymbol{x}}$, performs $\mathcal{O}(n + m')$ operations plus one call to* SOLVER *with a graph-tree tuple $(H', T', \boldsymbol{\tau}')$ with $\mathcal{O}(m')$ vertices and edges, the same bounds for the stretch of off-tree edges, and accuracy ϵ and returns a vector $\boldsymbol{x}$ such that*

$$\left\|\bar{\boldsymbol{x}} - \boldsymbol{x}\right\|_{\boldsymbol{L}_H} \leq \epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}_H}.$$

With this in mind, one way to view the recursive preconditioning framework is that it gradually reduces the number of edges using the statistical leverage scores obtained from a tree. For this, Koutis et al. [KMP11] used the low-stretch spanning tree algorithms [AKPW95, EEST08, ABN08, AN12]. However, the state of art result due to Abraham and embeddings takes $\mathcal{O}(m \log n \log \log n)$ time to construct.

Instead, we will use the low-stretch embeddings given by Cohen et al. [CMP+14]. Their result can be summarized as follows:

Lemma 5.4.5 *Given a graph $\hat{G}$ with n vertices, m edges, and any constant $0 < p < 1$, we can construct in $\mathcal{O}(m \log \log n \log \log \log n)$ time in the RAM model a graph-tree tuple $(G, T, \boldsymbol{\tau})$ and associated bounds on stretches of edges $\boldsymbol{\tau}$ such that*

1. *G has at most $2n$ vertices and $n + m$ edges, and*
2. *$\|\boldsymbol{\tau}\|_p^p \leq \mathcal{O}(m \log^p n)$, and*
3. *there is a $|V_{\hat{G}}| \times |V_G|$ matrix $\boldsymbol{\Pi}$ with one 1 in each row and zeros everywhere else such that:*

$$\frac{1}{2}\boldsymbol{L}_{\hat{G}}^\dagger \preceq \boldsymbol{\Pi}_1 \boldsymbol{\Pi} \boldsymbol{L}_G^\dagger \boldsymbol{\Pi}^T \boldsymbol{\Pi}_1^T \preceq \boldsymbol{L}_{\hat{G}}^\dagger.$$

Note that $\boldsymbol{\Pi}$ maps some vertices of G to unique vertices of $\hat{G}$, and $\boldsymbol{\Pi}^T$ maps each vertex of $\hat{G}$ to a unique vertex in G.

The spectral guarantees given in Part 3 allow the solver for $\boldsymbol{L}_G$ to be converted to a solver for $\boldsymbol{L}_{\hat{G}}$ while preserving the error quality.

```
$(H, T') = \textsc{RandPrecon}(G, T, \boldsymbol{\tau}, \delta)$, where $G$ is a graph, $T$ is a tree, $\boldsymbol{\tau}$ are upper bounds
of the stretches of edges in $G$ w.r.t. $T$, and $\delta < 1$ is an arbitrary parameter.

  1. Let $X = \mathbf{L}_T$, $Y = \mathbf{L}_G$, $Y_i$ be the rank-1 matrix corresponding to each edge.
  2. Set $\hat{\boldsymbol{\tau}}$ to be the same as $\boldsymbol{\tau}$ for non tree-edges, and 1 for all tree edges.
  3. Repeat
     (a) $Z = \textsc{Sample}\left(Y, X, \hat{\boldsymbol{\tau}}, \frac{1}{10}\right)$.
     (b) Set
         i. $H$ be the edges corresponding to $Z$, and
         ii. $T'$ be the edges corresponding to the combinatorial components in $T$, and
         iii. $\boldsymbol{\tau}'$ to be $\delta$ times the number of times each off-tree edge is sample.
  4. Until the number of off-tree edges in $H$ is at most $4800 \|\boldsymbol{\tau}\|_p^p$, and $\|\boldsymbol{\tau}'\|_p^p \leq 480 \|\boldsymbol{\tau}\|_p^p$.
  5. Return $(H, T', \boldsymbol{\tau}')$.
```

Figure 5.1: Generation of a Randomized Preconditioner

Fact 5.4.6 *Let $\mathbf{\Pi}$ and $\mathbf{\Pi}_1$ be the two projection matrices defined in Lemma 5.4.5 Part 3. For a vector $\hat{\boldsymbol{b}}$, if $\boldsymbol{x}$ is a vector such that*

$$\left\| \boldsymbol{x} - \boldsymbol{L}_G^{\dagger} \mathbf{\Pi}^T \mathbf{\Pi}_1^T \hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G} \leq \epsilon \left\| \boldsymbol{L}_G^{\dagger} \Pi^T \mathbf{\Pi}_1^T \hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G},$$

for some $\epsilon > 0$. Then the vector $\hat{\boldsymbol{x}} = \mathbf{\Pi}_1 \mathbf{\Pi} \boldsymbol{x}$ satisfies

$$\left\| \hat{\boldsymbol{x}} - \mathbf{\Pi}_1 \mathbf{\Pi} \boldsymbol{L}_G^{\dagger} \mathbf{\Pi}^T \mathbf{\Pi}_1^T \hat{\boldsymbol{b}} \right\|_{\left(\mathbf{\Pi}_1 \mathbf{\Pi} \boldsymbol{L}_G^{\dagger} \mathbf{\Pi}^T \mathbf{\Pi}_1^T\right)^{\dagger}} \leq \epsilon \left\| \mathbf{\Pi}_1 \mathbf{\Pi} \boldsymbol{L}_G^{\dagger} \mathbf{\Pi}^T \mathbf{\Pi}_1^T \hat{\boldsymbol{b}} \right\|_{\left(\mathbf{\Pi}_1 \mathbf{\Pi} \boldsymbol{L}_G^{\dagger} \mathbf{\Pi}^T \mathbf{\Pi}_1^T\right)^{\dagger}}.$$

Therefore, a good solution to $\boldsymbol{L}_G \boldsymbol{x} = \mathbf{\Pi}^T \widehat{\boldsymbol{b}}$ also leads to a good solution to $\boldsymbol{L}_{\widehat{G}} \widehat{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$. The constant relative error can in turn be corrected using preconditioned Richardson iteration described in Section 6.2. For the rest of our presentation, we will focus on solving linear systems in settings where we know small bounds to $\|\boldsymbol{\tau}\|_p^p$.

As SAMPLE will sample edges with high stretch, as well as tree edges, we need to modify its construction bounding both the number of off-tree edges, and the total off-tree ℓ_p-stretch. Pseudocode of this modified algorithm for generating a preconditioner is given in Figure 5.1.

We start by proving some crude guarantees of this algorithm.

Lemma 5.4.7 $\textsc{RandPrecon}(G, T, \boldsymbol{\tau}, \frac{1}{10})$ *runs in expected $\mathcal{O}(m + \|\boldsymbol{\tau}\|_p^p)$ time and produces a graph-tree tuple $(H, T, \boldsymbol{\tau}')$ such that*

1. the number of off-tree edges in H is at most $\mathcal{O}(\|\boldsymbol{\tau}\|_p^p)$, and

2. $\|\boldsymbol{\tau}'\|_p^p \leq \mathcal{O}(\|\boldsymbol{\tau}\|_p^p)$, and

3. for any pair of vectors $\boldsymbol{x}$ and $\boldsymbol{b} = \boldsymbol{L}_G\bar{\boldsymbol{x}}$, we have

$$\mathbb{E}_H\left[\left\|\bar{\boldsymbol{x}} - \left(\boldsymbol{x} - \frac{1}{10}\boldsymbol{L}_H^{\dagger}\left(\boldsymbol{L}_G\boldsymbol{x} - \boldsymbol{b}\right)\right)\right\|_{\boldsymbol{L}_G}\right] \leq \left(1 - \frac{1}{160}\right)\|\bar{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G}$$

Proof For an edge e, let X_e be a random variable indicating the number of times that e is sampled. The call to SAMPLE samples $\frac{s}{\delta}$ edges where s is the total stretch of all edges. In each of these iterations, e is sampled with probability $\frac{\boldsymbol{\tau}_e}{s}$ where $\boldsymbol{\tau}_e = \mathbf{tr}\left[\boldsymbol{\chi}_e^T\mathbf{L}_T^{\dagger}\boldsymbol{\chi}_e\right]$. This means the expected value of X_e is

$$\begin{aligned}\mathbb{E}\left[X_e\right] &\leq \frac{3}{2}\frac{s}{\delta}\frac{\boldsymbol{\tau}_e}{s}\\ &= \frac{3}{2}\delta^{-1}\boldsymbol{\tau}_e = 15\boldsymbol{\tau}_e.\end{aligned}$$

For an edge e, if $\boldsymbol{\tau}_e \geq 1$, then $\boldsymbol{\tau}_e^p \geq 1$; otherwise, $\boldsymbol{\tau}_e \leq \boldsymbol{\tau}_e^p$. Therefore we have that the expected number of distinct edges added to the tree is less than $15\left\|\boldsymbol{\tau}\right\|_p^p$. Markov's inequality then gives that we sample more than $4800\left\|\boldsymbol{\tau}\right\|_p^p$ edges with probability at most $\frac{1}{320}$.

For the expected stretch, note that as T is added to H, the stretch of an edge can only decrease. Combined with the fact that each sampled edge has stretch δ with respect to T:

$$\mathbb{E}\left[(\boldsymbol{\tau}'_e)^p\right] \leq \mathbb{E}\left[(\delta X_e)^p\right] \leq \mathbb{E}\left[\delta X_e\right]^p \leq \frac{3}{2}\boldsymbol{\tau}_e^p.$$

So the expected total ℓ_p-stretch of all off-tree edges is at most $\frac{3}{2}\boldsymbol{\tau}_e^p$. Applying Markov's inequality once again gives that the probability of $\left\|\boldsymbol{\tau}'\right\|_p^p \leq 480\left\|\boldsymbol{\tau}\right\|_p^p$ is also at most $\frac{1}{320}$.

Taking a union bound gives that each sample H fails the conditions with probability at most $\frac{1}{160}$. This means that the loop is expected to terminate in $\mathcal{O}(1)$ iterations. Also, this means that the expected deviation in H only increases by a constant factor, giving

$$\begin{aligned}\mathbb{E}_H\left[\left\|\bar{\mathbf{x}} - \left(\mathbf{x} - \frac{1}{10}\mathbf{L}_H^{\dagger}\left(\mathbf{L}_G\mathbf{x} - \mathbf{b}\right)\right)\right\|_{\mathbf{L}_G}\right] &\leq \frac{1}{1 - \frac{1}{160}}\left(1 - \frac{1}{80}\right)\|\bar{\mathbf{x}} - \mathbf{x}\|_{\mathbf{L}_G}\\ &\leq \left(1 - \frac{1}{160}\right)\|\bar{\mathbf{x}} - \mathbf{x}\|_{\mathbf{L}_G}.\end{aligned}$$

■

We can then apply the elimination routine from Lemma 5.4.4 to obtain a high-quality solution to a linear system by solving a small number of systems whose edge count is $\mathcal{O}(\|\boldsymbol{\tau}\|_p^p)$.

However, note that the error is in the $\mathbf{L}_H$-norm. To relate this to the $\mathbf{L}_G$-norm, we can use a spectral bound derived from matrix concentration bounds. Such a bound is central in operator based solvers by Koutis et al. [KMP10, KMP11], while we feel our use of it here is more tangential.

Lemma 5.4.8 *There exists a constant c such that for any graph-tree tuple G, T, $\boldsymbol{\tau}$, $H = \text{RANDPRECON}(G, T, \boldsymbol{\tau}, \frac{1}{10})$ satisfies*

$$\frac{1}{c \log n} \boldsymbol{L}_G \preceq \boldsymbol{L}_H \preceq c \log n \boldsymbol{L}_G$$

with high probability.

We prove this bound in Appendix 5.C. It means that the decrease in energy can still be guaranteed if we set $\epsilon = \mathcal{O}(\frac{1}{c_s \log n})$ in our bounds. We can also check whether we have reached such an error using coarser solvers.

Lemma 5.4.9 *There exist a constant c_Z such that given a graph-tree tuple G, T, $\boldsymbol{\tau}$, we can construct with high probability a linear operator Z such that under exact arithmetic*

1. *$\boldsymbol{L}_G^\dagger \preceq Z \preceq c_Z \log^4 n \boldsymbol{L}_G^\dagger$, and*
2. *given any vector $\boldsymbol{b}$, $Z\boldsymbol{b}$ can be evaluated in $\mathcal{O}(m + \|\boldsymbol{\tau}\|_p^p)$ time where p is any constant $> 1/2$.*

Proof Consider scaling up the tree by a factor of $\log^2 n$ and scaling down the bounds on leverage scores accordingly to obtain $G', T', \boldsymbol{\tau}'$. Then $\boldsymbol{L}_G \preceq \boldsymbol{L}_{G'} \preceq \log^2 n \boldsymbol{L}_G$ and Lemma 5.4.7 gives that $H = \text{RANDPRECON}(G', T', \boldsymbol{\tau}')$ has $\mathcal{O}(\|\boldsymbol{\tau}'\|_p^p) = \mathcal{O}(\log^{-2p} n \|\boldsymbol{\tau}\|_p^p)$ off-tree edges, and

$$\frac{1}{c \log n} \boldsymbol{L}_{G'} \preceq \boldsymbol{L}_H \preceq c \log n \boldsymbol{L}_{G'}.$$

Applying partial Cholesky factorization on H and then the solver algorithm by Koutis et al. [KMP11] then gives an operator $\boldsymbol{Z}$ such that

$$\frac{1}{2} \boldsymbol{L}_H^\dagger \preceq \boldsymbol{Z} \preceq 2 \boldsymbol{L}_H^\dagger,$$

and $\boldsymbol{Z}\boldsymbol{b}$ can be evaluated in $\mathcal{O}(m + \log^{-2p} n \|\boldsymbol{\tau}\|_p^p \log n \log \log^{1/2} n) \leq O(m + \|\boldsymbol{\tau}\|_p^p)$ time. Propagating the error guarantees then gives

$$\boldsymbol{Z} \preceq 2\boldsymbol{L}_H^\dagger \preceq 2c \log n \boldsymbol{L}_{G'}^\dagger \preceq 2c \log n \boldsymbol{L}_G^\dagger,$$

for the upper bound, and

$$\boldsymbol{Z} \succeq \frac{1}{2} \boldsymbol{L}_H^\dagger \succeq \frac{1}{2c \log n} \boldsymbol{L}_{G'}^\dagger \succeq \frac{1}{2c \log^3 n} \boldsymbol{L}_G^\dagger,$$

$\mathbf{x} = \text{RANDRICHARDSON}(G, T, \boldsymbol{\tau}, \text{SOLVER}, \mathbf{b}, \epsilon)$, where G is a graph, T is a tree, $\boldsymbol{\tau}$ are upper bounds of the stretches of edges of G w.r.t. T, $\mathbf{b}$ is the vector to be solved, and ϵ is the target error.

```
1. Set $\epsilon_1 = \frac{1}{320 c_s \log n}$ and $t = \mathcal{O}\left(\log\left(\epsilon^{-1} \log n\right)\right)$.
2. Let $Z$ be the linear operator corresponding to the solver given in Lemma 5.4.9
3. Repeat
   (a) $\mathbf{x}_0 = 0$.
   (b) For $i = 1 \ldots t$
       i. $(H_i, T_i, \boldsymbol{\tau}_i) = \text{RANDPRECON}(G, T, \boldsymbol{\tau}, \delta)$.
       ii. $\mathbf{r}_i = \mathbf{L}_G \mathbf{x}_{i-1} - \mathbf{b}$.
       iii. $\mathbf{y}_i = \text{ELIMINATE\&SOLVE}\left(H_i, T_i, \boldsymbol{\tau}_i, \text{SOLVER}, \mathbf{r}_i, \epsilon_1\right)$.
       iv. $\mathbf{x}_i = \mathbf{x}_{i-1} - \frac{1}{10}\mathbf{y}_i$.
4. Until $\left\|Z\left(\mathbf{L}_G \mathbf{x}_t - \mathbf{b}\right)\right\|_{\mathbf{L}_G} \leq \frac{\epsilon}{c_Z \log^4 n} \left\|Z\mathbf{b}\right\|_{\mathbf{L}_G}$.
5. Return $\mathbf{x}_t$
```

Figure 5.2: Randomized Richardson Iteration

for the lower bound. Scaling $\boldsymbol{Z}$ by a factor of $2c\log^3 n$ then gives the required operator.

■

Using this routine allows us to convert the expected convergence to one that involves expected running time, but converges with high probability. This is mostly to simplify our presentation and we believe such a dependency can be removed. Using this routine leads us to our randomized preconditioned Richardson iteration routine, whose pseudocode is given in Figure 5.2.

The guarantees of this routine is as follows.

Lemma 5.4.10 *Given a Laplacian solver* SOLVER*, any graph-tree pair* (G, T)*, bounds on stretch* $\boldsymbol{\tau}$*, vector* $\boldsymbol{b} = \boldsymbol{L}_G\bar{\boldsymbol{x}}$ *and error* $\epsilon > 0$, $\text{RANDRICHARDSON}(G, T, \boldsymbol{\tau}, \text{SOLVER}, \boldsymbol{b}, \epsilon)$ *returns with high probability a vector* $\boldsymbol{x}$ *such that*

$$\left\|\boldsymbol{x} - \bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}_G} \leq \epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}_G}, \tag{5.6}$$

and the algorithm takes an expected $\mathcal{O}(\log(\epsilon^{-1}) + \log\log n))$ *iterations. Each iteration consists of one call to* SOLVER *on a graph with* $\mathcal{O}(\|\boldsymbol{\tau}\|_p^p)$ *edges and error* $\frac{1}{\mathcal{O}(\log n)}$*, plus an overhead of* $\mathcal{O}(m + \|\boldsymbol{\tau}\|_p^p)$ *operations.*

Proof Consider each iteration step using the preconditioner H_i generated by RANDPRECON. The error reduction given in Lemma 5.4.7 gives:

$$\mathbb{E}_H \left[\left\| \bar{\mathbf{x}} - \left(\mathbf{x}_{i-1} - \frac{1}{10} \mathbf{L}_H^{\dagger} \mathbf{r}_i \right) \right\|_{\mathbf{L}_G} \right] \leq \left(1 - \frac{1}{160} \right) \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G}.$$

On the other hand, the guarantee for SOLVER gives

$$\left\| \mathbf{y}_i - \mathbf{L}_H^{\dagger} \mathbf{r}_i \right\|_{\mathbf{L}_H} \leq \epsilon_1 \left\| \mathbf{L}_H^{\dagger} \mathbf{r}_i \right\|_{\mathbf{L}_H}.$$

Substituting in the spectral bound between $\mathbf{L}_G$ and $\mathbf{L}_H$ given by Lemma 5.4.8 in turn gives:

$$\begin{aligned} \left\| \mathbf{y}_i - \mathbf{L}_H^{\dagger} \mathbf{r}_i \right\|_{\mathbf{L}_G} &\leq \sqrt{c_s \log n} \epsilon_1 \left\| \mathbf{L}_G \left(\bar{\mathbf{x}} - \mathbf{x}_{i-1} \right) \right\|_{\mathbf{L}_H^{\dagger}} \\ &\leq c_s \log n \epsilon_1 \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G} \\ &\leq \frac{1}{320} \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G}. \end{aligned}$$

Combining this with the above bound via the triangle inequality then gives

$$\begin{aligned} \mathbb{E}_H \left[\left\| \bar{\mathbf{x}} - \left(\mathbf{x}_{i-1} - \frac{1}{10} \mathbf{L}_H^{\dagger} \mathbf{r}_i \right) \right\|_{\mathbf{L}_G} \right] &\leq \left(1 - \frac{1}{160} \right) \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G} + \frac{1}{320} \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G} \\ &\leq \left(1 - \frac{1}{320} \right) \left\| \bar{\mathbf{x}} - \mathbf{x}_{i-1} \right\|_{\mathbf{L}_G}. \end{aligned}$$

Hence the expected error $\|\bar{\mathbf{x}} - \mathbf{x}_i\|$ decreases by a constant factor per iteration. After $\mathcal{O}(\log(\epsilon^{-1} \log n))$ iterations the expected error is less than $\frac{1}{2} \frac{\epsilon}{c_Z \log^4 n}$, where c_Z is the constant from Lemma 5.4.9. Markov's inequality gives that

$$\left\| \mathbf{x}_t - \bar{\mathbf{x}} \right\|_{\mathbf{L}_G} \leq \frac{\epsilon}{c_Z \log^4 n} \left\| \bar{\mathbf{x}} \right\|_{\mathbf{L}_G} \tag{5.7}$$

with probability at least $\frac{1}{2}$. By lemma 5.4.9 we have w.h.p

$$\mathbf{L}_G^{\dagger} \preceq Z \preceq c_Z \log^4 n \mathbf{L}_G^{\dagger}.$$

If this equation holds, then the termination criterion is satisfied whenever equation 5.7 holds, because

$$\begin{aligned} \left\| Z \left(\mathbf{L}_G \mathbf{x}_t - \mathbf{b} \right) \right\|_{\mathbf{L}_G} &\leq c_z \log^4 n \left\| \mathbf{x}_t - \bar{\mathbf{x}} \right\|_{\mathbf{L}_G} \\ &\leq \epsilon \left\| \bar{\mathbf{x}} \right\|_{\mathbf{L}_G} \\ &\leq \epsilon \left\| Z \mathbf{b} \right\|_{\mathbf{L}_G}. \end{aligned}$$

On the other hand, when the termination criterion holds,

$$\begin{aligned}
\|\bar{\mathbf{x}} - \mathbf{x}_t\|_{\mathbf{L}_G} &\leq \|\mathbf{L}_G (\bar{\mathbf{x}} - \mathbf{x}_t)\|_Z \\
&\leq \|Z(\mathbf{L}_G \mathbf{x}_t - \mathbf{b})\|_{\mathbf{L}_G} \\
&\leq \frac{\epsilon}{c_Z \log^4 n} \|Z\mathbf{b}\|_{\mathbf{L}_G} \\
&\leq \epsilon \|\bar{\mathbf{x}}\|_{\mathbf{L}_G} .
\end{aligned}$$

This means that w.h.p. equation 5.6 is satisfied when the algorithm terminates, and the algorithm terminates with probability at least $\frac{1}{2}$ on each iteration. So the expected number of iterations of the outer loop is $\mathcal{O}(1)$.

■

It remains to give use this routine recursively. We correct for the errors of introducing scaling factors into the tree using preconditioned Chebyshev iteration.

Lemma 5.4.11 (Preconditioned Chebyshev Iteration) *Given a matrix A and a matrix B such that $A \preceq B \preceq \kappa A$ for some constant $\kappa > 0$, along with error $\epsilon > 0$ and a routine $\textsc{Solve}_B$ such that for any vector $\boldsymbol{b}$ we have*

$$\left\|\textsc{Solve}_B(\boldsymbol{b}) - B^{\dagger}\boldsymbol{b}\right\|_B \leq \frac{\epsilon^4}{30\kappa^4} \|\boldsymbol{b}\|_{B^{\dagger}} ;$$

the preconditioned Chebyshev iteration routine $\textsc{Solve}_A(\cdot) = \textsc{PreconCheby}(A, B, \textsc{Solve}_B, \cdot)$ is such that in the exact arithmetic model, for any vector $\boldsymbol{b}$,

- $$\left\|\textsc{Solve}_A(\boldsymbol{b}) - A^{\dagger}\boldsymbol{b}\right\|_A \leq \epsilon \|\boldsymbol{b}\|_{A^{\dagger}} ,$$

  and

- *$\textsc{Solve}_A(\boldsymbol{b})$ takes $O(\sqrt{\kappa}\log(1/\epsilon))$ iterations, each consisting of one call to $\textsc{Solve}_B$ and a matrix-vector multiplication using A.*

The pseudocode of our algorithm is given in Figure 5.3. Below we prove its guarantee.

Lemma 5.4.12 *Given a parameter $1/2 < p < 1$ and a graph-tree tuple $(G, T, \boldsymbol{\tau})$ with m edges such that $\|\boldsymbol{\tau}\|_p^p \leq m \log^p n$. For any vector $\boldsymbol{b} = \boldsymbol{L}_G\bar{\boldsymbol{x}}$, $\textsc{Solve}(G, T, \boldsymbol{\tau}, \boldsymbol{b}, \frac{1}{320 c_s \log n})$ returns w.h.p. a vector $\boldsymbol{x}$ such that*

$$\|\bar{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{L}_G} \leq \frac{1}{320 c_s \log n} \|\boldsymbol{x}\|_{\boldsymbol{L}_G} ,$$

$\mathbf{x} = \textsc{Solve}(G, T, \boldsymbol{\tau}, \mathbf{b}, \epsilon)$, where G is a graph, T is a tree, $\boldsymbol{\tau}$ are upper bounds of the stretches of edges in G w.r.t. T, $\mathbf{b}$ is the vector to be solved, and ϵ is the goal error.

```
1. Set $\kappa = c(\log\log n)^{4/(2p-1)} \left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{1/p}$ for an appropriate constant $c$ (dependent on $p$).
2. Let $(H, T', \boldsymbol{\tau}')$ be the graph-tree tuple with $T$ scaled up by a factor of $\kappa$, $\boldsymbol{\tau}$ scaled
   down by a factor of $\kappa$.
3. $\mathbf{x} = \textsc{PreconCheby}(G, H, \textsc{RandRichardson}(H, T', \boldsymbol{\tau}', \textsc{Solve}, \epsilon^4\kappa^{-4}), \mathbf{b})$.
4. Return $\mathbf{x}$
```

Figure 5.3: Recursive Solver

and its expected running time is

$$\mathcal{O}\left(m\left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{\frac{1}{2p}} \log\log^{2+\frac{2}{2p-1}} n\right).$$

Proof The proof is by induction on graph size. As our induction hypothesis, we assume the lemma to be true for all graphs of size $m' < m$. The choice of κ gives

$$\|\boldsymbol{\tau}'\|_p^p \leq \frac{m}{c^p \log\log^{2+\frac{2}{2p-1}} n}.$$

The guarantees of randomized Richardson iteration from Lemma 5.4.10 gives that all the randomized preconditioners have both off-tree edge count and off-tree stretch bounded by $\mathcal{O}(\|\boldsymbol{\tau}'\|_p^p) = \mathcal{O}\left(\frac{m}{c^p \log\log^{2+\frac{2}{2p-1}} n}\right)$.

An appropriate choice of c makes both of these values strictly less than m, and this allows us to apply the inductive hypothesis on the graphs obtained from the randomized preconditioners by ELIMINATE&SOLVE.

As κ is bounded by $c\log^2 n$ and ϵ is set to $\frac{1}{320c_s \log n}$, the expected cost of the recursive calls made by RANDRICHARDSON is

$$\mathcal{O}(m \log\log n).$$

Combining this with the iteration count in PRECONCHEBY of

$$\mathcal{O}(\sqrt{\kappa}\log(1/\epsilon)) = \mathcal{O}\left((\log\log n)^{\frac{2}{2p-1}} \left(\frac{\|\boldsymbol{\tau}\|_p^p}{m}\right)^{\frac{1}{2p}} \log\log n\right)$$

gives the inductive hypothesis.

■

To prove theorem 5.1.1, we first invoke SOLVE with ϵ set to a constant. Following an analysis identical to the proof of lemma 5.4.12, at the top level each iteration of PRECONCHEBY will require $\mathcal{O}(m \log \log n)$ time, but now only

$$\mathcal{O}(\sqrt{\kappa} \log(1/\epsilon)) = \mathcal{O}\left((\log \log n)^{\frac{2}{2p-1}} \left(\frac{\|\boldsymbol{\tau}\|_p^p}{m} \right)^{\frac{1}{2p}} \right)$$

iterations are necessary. Setting p arbitrarily close to 1 means that for any constant $\delta > 0$ and relative error ϵ, there is a solver for $\boldsymbol{L}_G$ that runs in $\mathcal{O}(m \log^{1/2} n \log \log^{3+\delta} n)$ time. This error can be reduced using Richardson iteration as stated below.

Lemma 5.4.13 *If $\boldsymbol{A}$, $\boldsymbol{B}$ are matrices such that $\boldsymbol{A} \preceq \boldsymbol{B} \preceq 2\boldsymbol{A}$ and* SOLVE$_{\boldsymbol{B}}$ *is a routine such that for any vector $\boldsymbol{b}$, we have $\left\| \text{SOLVE}_{\boldsymbol{B}}(\boldsymbol{b}) - \boldsymbol{B}^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{B}} \leq \frac{1}{5} \left\| \boldsymbol{B}^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{B}}$, then there is a routine* SOLVE$_{\boldsymbol{A},\epsilon}$ *which runs in $\mathcal{O}(c_\alpha \log(\frac{1}{\epsilon}))$ iterations with the guarantee that for any vector $\boldsymbol{b}$ we have $\left\| \text{SOLVE}_{\boldsymbol{A},\epsilon}(\boldsymbol{b}) - A^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{A}} \leq \epsilon \left\| \boldsymbol{A}^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{A}}$. Each iteration involves one call to* SOLVE$_{\boldsymbol{B}}$*, a matrix-vector multiplication involving $\boldsymbol{A}$ and $O(1)$ arithmetic operations on vectors.*

We will use Richardson iteration as the outer loop, while transferring solutions and errors to the original graph using the guarantees of the embeddable tree given in Lemma 5.4.5.

Proof of Theorem 5.1.1: Using Fact 5.4.6 on the solver described above for $\boldsymbol{L}_G$ gives a solver for $(\boldsymbol{\Pi}_1 \boldsymbol{\Pi} \boldsymbol{L}_G^{\dagger} \boldsymbol{\Pi}^T \boldsymbol{\Pi}_1^T)^{\dagger}$ with relative error $\frac{1}{5}$. This condition and Lemma 5.4.5 Part 3 then allows us to invoke the above Lemma with $\boldsymbol{A} = \boldsymbol{L}_{\hat{G}}$ and $\mathbf{B} = \boldsymbol{\Pi}_1 \boldsymbol{\Pi} \boldsymbol{L}_G^{\dagger} \boldsymbol{\Pi}^T \boldsymbol{\Pi}_1^T$. Incorporating the $O(\log(\frac{1}{\epsilon}))$ iteration count and the reduction from SDD linear systems then gives the overall result.

■

Appendix

5.A Chebyshev Iteration with Errors

We now check that preconditioned Chebyshev iteration can tolerate a reasonable amount of error in each of the calls to the preconditioner. A more detailed treatment of iterative methods can be found in the book by Trefethen and Bau [TB97]. Our presentation in this section is geared to proving the following guarantee.

Lemma 5.A.1 (Preconditioned Chebyshev Iteration) *Given a matrix A and a matrix B such that $A \preceq B \preceq \kappa A$ for some constant $\kappa > 0$, along with error $\epsilon > 0$ and a routine* $\textsc{Solve}_B$ *such that for any vector $\boldsymbol{b}$ we have*

$$\left\| \textsc{Solve}_B(\boldsymbol{b}) - B^{\dagger}\boldsymbol{b} \right\|_B \leq \frac{\epsilon^4}{30\kappa^4} \left\| \boldsymbol{b} \right\|_{B^{\dagger}};$$

the preconditioned Chebyshev iteration routine $\textsc{Solve}_A(\cdot) = \textsc{PreconCheby}\left(A, B, \textsc{Solve}_B, \cdot\right)$ *is such that in the exact arithmetic model, for any vector $\boldsymbol{b}$,*

- $$\left\| \textsc{Solve}_A(\boldsymbol{b}) - A^{\dagger}\boldsymbol{b} \right\|_A \leq \epsilon \left\| \boldsymbol{b} \right\|_{A^{\dagger}},$$

  and

- $\textsc{Solve}_A(\boldsymbol{b})$ *takes $O(\sqrt{\kappa}\log(1/\epsilon))$ iterations, each consisting of one call to* $\textsc{Solve}_B$ *and a matrix-vector multiplication using A.*

As the name suggests, Chebyshev iteration is closely related with Chebyshev Polynomials. There are two kinds of Chebyshev Polynomials, both defined by recurrences. Chebyshev polynomials of the first kind, $T_n(x)$ can be defined as:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{i+1}(x) &= 2xT_i(x) - T_{i-1}(x). \end{aligned}$$

Preconditioned Chebyshev iteration is given by the following recurrence with δ set to $1 + \frac{1}{\kappa}$:

Base case:

$$\begin{aligned}\mathbf{x}_0 &= 0\\ \mathbf{x}_1 &= \textsc{Solve}_B(\mathbf{b})\end{aligned}$$

Iteration:

$$\begin{aligned}\mathbf{y}_{i+1} &= \textsc{Solve}_B\left(A\mathbf{x}_i - \mathbf{b}\right)\\ \mathbf{x}_{i+1} &= \frac{2\delta T_i\left(\delta\right)}{T_{i+1}\left(\delta\right)}\left(\mathbf{x}_i - \mathbf{y}_{i+1}\right) - \frac{T_{i-1}\left(\delta\right)}{T_{i+1}\left(\delta\right)}\mathbf{x}_{i-1}\end{aligned}$$

Figure 5.A.1: Preconditioned Chebyshev Iteration

To bound the convergence of this iteration, it is helpful to use the following closed form for $T_i(x)$:

$$T_i(x) = \frac{\left(x - \sqrt{x^2-1}\right)^i + \left(x + \sqrt{x^2-1}\right)^i}{2}.$$

The following facts about Chebyshev polynomials of the first kind will be used to bound convergence.

Fact 5.A.2 *If $x = \cos(\theta)$, then*

$$T_i(x) = \cos(i\theta).$$

This implies that if $|x| \leq 1$, $|T_i(x)| \leq 1$, and we will pass the error of the algorithm through it. For convergence, we also need the opposite statement for lower bounding $T_n(x)$ when x is large.

Fact 5.A.3 *If $x = 1 + \frac{1}{\kappa}$, then:*

$$\begin{aligned}T_i(x) &\geq \frac{1}{2}\left(x + \sqrt{x^2-1}\right)^i,\\ &\geq \frac{1}{2}\left(1 + \sqrt{1 + \frac{2}{\kappa} - 1}\right)^i,\\ &\geq \frac{1}{2}\left(1 + \frac{1}{\sqrt{\kappa}}\right)^i.\end{aligned}$$

We can also show that these terms are steadily increasing:

Fact 5.A.4 *If $i \leq j$ and $x \geq 1$, then $T_i(x) \geq \frac{1}{2}T_j(x)$.*

Proof $x \geq 1$ implies $0 \leq x - \sqrt{x^2-1} \leq 1$ and $1 \leq x + \sqrt{x^2-1}$. Therefore

$$\begin{aligned} T_{i+1}(x) &\geq \frac{\left(x+\sqrt{x^2-1}\right)^{i+1}}{2}, \\ &\geq \frac{\left(x+\sqrt{x^2-1}\right)^{i}}{2}, \\ &\geq T_i(x) - \frac{1}{2}. \end{aligned}$$

Fact 5.A.3 also gives $T_{i+1}(x) \geq \frac{1}{2}$. Combining these gives $T_i(\delta) \geq \frac{1}{2} T_j(\delta)$.

■

The errors given by $\textsc{Solve}_B$ will accumulate over the iterations. To bound them, we need Chebyshev polynomials of the second kind. These polynomials, $U_n(x)$, follow the same recurrence but have a different base case:

$$\begin{aligned} U_{-1}(x) &= 0, \\ U_0(x) &= 1, \\ U_{i+1}(x) &= 2xT_i(x) - T_{i-1}(x). \end{aligned}$$

Chebyshev polynomials of the second kind are related to Chebyshev polynomials of the first kind by the following identity:

Fact 5.A.5

$$U_i(x) = \begin{cases} 2\sum_{j \leq i \text{ odd}} T_j(x) & \textit{If } i \textit{ is odd, and} \\ \left(2\sum_{j \leq i \text{ even}} T_j(x)\right) - 1 & \textit{If } i \textit{ is even.} \end{cases}$$

Since $T_0 = 1$, and $|T_j(x)| \leq 1$ whenever $x \leq 1$, this implies

Fact 5.A.6 *For all x satisfying $|x| \leq 1$,*

$$|U_i(x)| \leq i+1$$

We will let the deviation caused by $\textsc{Solve}_B$ at iteration i to be $\mathbf{err}_i$, giving

$$\mathbf{y}_{i+1} = B^{\dagger}\left(A\mathbf{x}_i - \mathbf{b}\right) + \mathbf{err}_i$$

where $\|\mathbf{err}_i\|_B \leq \|A\mathbf{x}_i - \mathbf{b}\|_{B^{\dagger}}$. To analyze the recurrence, it is crucial to consider the matrix

$$X = \delta\left(I - A^{1/2}B^{\dagger}A^{1/2}\right).$$

The given condition of $A \preceq B \preceq \kappa A$ gives

$$\frac{1}{\kappa} I \preceq A^{1/2} B^{\dagger} A^{1/2} \preceq I,$$

which when combined with the setting of $\delta = 1 + \frac{1}{\kappa}$ gives

$$0 \preceq X \preceq I.$$

Fact 5.A.2 then gives that $T_i(X)$ has all eigenvalues between $[-1, 1]$. This 'shrinkage' property is key to our analysis.

We can show that the deviation between $\mathbf{x}_i$ and $\bar{\mathbf{x}} = A^{\dagger} b$ behaves according to Chebyshev polynomials of the first kind in X while the errors accumulate according to Chebyshev polynomials of the second kind in X.

Lemma 5.A.7 *If $\bar{X} = A^{\dagger} b$, then at iteration i we have*

$$T_i\left(\delta\right)\left(\boldsymbol{x}_i - \bar{\boldsymbol{x}}\right) = A^{\dagger 1/2} T_i\left(X\right) A^{1/2} \bar{\boldsymbol{x}} + 2\delta \sum_{j=1}^{i} T_{j-1}\left(\delta\right) A^{\dagger 1/2} U_{i-j}\left(X\right) A^{1/2} \boldsymbol{err}_j,$$

where $X = \delta\left(I - A^{1/2} B^{\dagger} A^{1/2}\right)$ and $T_i(X)$ and $U_i(x)$ are Chebyshev polynomials of the first and second kind respectively

Proof The proof is by induction.

The base case can be checked as follows:

$$\begin{aligned}
\bar{\mathbf{x}} - \mathbf{x}_0 &= \bar{\mathbf{x}}, \\
&= A^{\dagger 1/2} A^{1/2} \bar{\mathbf{x}}; \\
\bar{\mathbf{x}} - \mathbf{x}_1 &= \bar{\mathbf{x}} - B^{\dagger} \mathbf{b} + \mathbf{err}_1, \\
&= \bar{\mathbf{x}} - B^{\dagger} A \bar{\mathbf{x}}, \\
&= A^{\dagger 1/2}\left(I - A^{1/2} B^{\dagger} A^{1/2}\right) A^{1/2} \bar{\mathbf{x}} + A^{\dagger 1/2} A^{1/2} \mathbf{err}_1.
\end{aligned}$$

For the inductive case, the recurrence can be rearranged to give:

$$T_{i+1}\left(\delta\right) \mathbf{x}_{i+1} = 2\delta T_i\left(\delta\right)\left(\mathbf{x}_i - \mathbf{y}_{i+1}\right) - T_{i-1}\left(\delta\right)\left(\delta\right) \mathbf{x}_{i-1}$$

Recall from the definition of Chebyshev polynomials of the first kind that:

$$T_{i+1}\left(\delta\right) = 2\left(\delta\right) T_i\left(\delta\right) - T_{i-1}\left(\delta\right)$$

So we can subtract both sides from $T_{i+1}(\delta)\bar{\mathbf{x}}$ to get:

$$T_{i+1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_{i+1}) = 2\delta T_i(\delta)(\bar{\mathbf{x}}_i - \mathbf{x}_i + \mathbf{y}_i) - T_{i-1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_{i-1})$$

The change, y_i, can be viewed as computed by multiplying the difference at iteration i by $B^{-1}A$, plus the error vector $\mathbf{err}_i$:

$$\begin{aligned} y_{i+1} &= B^{\dagger}(A\mathbf{x}_i - \mathbf{b}) + \mathbf{err}_{i+1} \\ &= B^{\dagger}(A\mathbf{x}_i - A\bar{\mathbf{x}}) + \mathbf{err}_{i+1} \\ &= B^{\dagger}A(\mathbf{x}_i - \bar{\mathbf{x}}) + \mathbf{err}_{i+1} \end{aligned}$$

Combining this gives

$$\begin{aligned} T_{i+1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_{i+1}) &= 2\delta T_i(\delta)\left(I - B^{\dagger}A\right)(\bar{\mathbf{x}} - \mathbf{x}_i) - T_{i-1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_{i-1}) + 2\delta T_i(\delta)\mathbf{err}_{i+1} \\ &= 2A^{\dagger 1/2}XA^{1/2}T_i(\delta)(\bar{\mathbf{x}} - \mathbf{x}_i) - T_{i-1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_{i-1}) + 2\delta T_i(\delta)\mathbf{err}_{i+1}. \end{aligned}$$

From this, we can then show the inductive case by collecting all the terms and checking that the coefficients satisfy the recurrences for Chebyshev polynomials. Substituting in the inductive hypothesis gives:

$$\begin{aligned} T_{i+1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_i) = 2A^{\dagger 1/2}XA^{1/2} &\left(A^{\dagger 1/2}T_i(X)A^{1/2}\bar{\mathbf{x}} + 2\delta\sum_{j=1}^{i} T_{j-1}(\delta)A^{\dagger 1/2}U_{i-j}(X)A^{1/2}\mathbf{err}_j\right) \\ &+ A^{\dagger 1/2}T_{i-1}(X)A^{1/2}\bar{\mathbf{x}} \\ &+ 2\delta\sum_{j=1}^{i-1} T_{j-1}(\delta)A^{\dagger 1/2}U_{i-1-j}(X)A^{1/2}\mathbf{err}_j \\ &+ 2\delta T_i(\delta)\mathbf{err}_{i+1} \end{aligned}$$

Since A, B and X share the same null space and the first term is left-multiplied by $A^{\dagger 1/2}$, the $A^{1/2}$ and $A^{\dagger 1/2}$ terms cancel with each other. Collecting the terms according to $\bar{\mathbf{x}}$ and $\mathbf{err}_j$ then gives

$$\begin{aligned} T_{i+1}(\delta)(\bar{\mathbf{x}} - \mathbf{x}_i) = A^{\dagger 1/2}&(2XT_i(X) - T_{i-1}(X))A^{1/2}\bar{\mathbf{x}} \\ &+ 2\delta\sum_{j=1}^{i} T_{j-1}(\delta)A^{\dagger 1/2}(2XU_{i-j}(X) - U_{i-1-j}(X))A^{1/2}\mathbf{err}_j \\ &+ 2\delta T_i(\delta)\mathbf{err}_{i+1} \\ = A^{\dagger 1/2}&T_{i+1}(X)A^{1/2}\bar{\mathbf{x}} + 2\delta\sum_{j=1}^{i} T_{j-1}(\delta)U_{i+1-j}(X)A^{1/2}\mathbf{err}_j + 2\delta T_i(\delta)\mathbf{err}_{i+1} \end{aligned}$$

As $U_{-1}(x) = 0$, we can also include in the $j = i$ term in the summation of error terms. So the inductive hypothesis holds for $i + 1$ as well.

■

The bound on Chebyshev polynomials of the second kind (Fact 5.A.6) then allows us to bound the error in the A-norm.

Lemma 5.A.8 *The accumulation of errors after i iterations can be bounded by:*

$$\|\bar{\boldsymbol{x}} - \boldsymbol{x}_i\|_A \leq \frac{1}{T_i(\delta)} \|\bar{\boldsymbol{x}}\|_A + \sum_{j=1}^{i} 6i \left\|\boldsymbol{err}_j\right\|_A$$

Proof By the identity proven in Lemma 5.A.7 above, and the property of norms, we have:

$$\begin{aligned}\|\bar{\mathbf{x}} - \mathbf{x}_i\|_A &= \left\| A^{\dagger 1/2} T_{i+1}(X) A^{1/2} \bar{\mathbf{x}} + 2\delta \sum_{j=1}^{i-1} T_{j-1}(\delta) U_{i+1-j}(X) A^{1/2} \mathbf{err}_j + 2\delta T_i(\delta) \mathbf{err}_{i+1} \right\|_A \\ &= \frac{1}{T_i(\delta)} \left\| T_{i+1}(X) A^{1/2} \bar{\mathbf{x}} + 2\delta \sum_{j=1}^{i} T_{j-1}(\delta) U_{i+1-j}(X) A^{1/2} \mathbf{err}_j \right\|_2,\end{aligned}$$

on which triangle inequality gives:

$$\|\bar{\mathbf{x}} - \mathbf{x}_i\|_A \leq \frac{1}{T_i(\delta)} \left\|T_{i+1}(X) A^{1/2} \bar{\mathbf{x}}\right\|_2 + \frac{2\delta T_{j-1}(\delta)}{T_i(\delta)} \sum_{j=1}^{i} \left\|U_{i-j}(X) A^{1/2} \mathbf{err}_j\right\|_2$$

The upper bound on T implies that the eigenvalues of $T_i(X)$ all have absolute value at most 1; similarly the upper bound on U given in Fact 5.A.6 implies that all eigenvalues of $U_k(X)$ have absolute value at most $k + 1$. This implies that for any vector $\mathbf{x}$, $\|T_i(X)\mathbf{x}\|_2 \leq \|\mathbf{x}\|_2$ and $\|U_k(X)\mathbf{x}\|_2 \leq (k + 1) \|\mathbf{x}\|_2$. Furthermore, by Fact 5.A.3, $\frac{2\delta T_{j-1}(\delta)}{<}6$. Applying these bounds, and the definition of A-norm, gives

$$\begin{aligned}\|\bar{\mathbf{x}} - \mathbf{x}_i\|_A &\leq \frac{1}{T_i(\delta)} \|\bar{\mathbf{x}}\|_A + 6 \sum_{j=1}^{i} (i - j + 1) \|\mathbf{err}_j\|_A \\ &\leq\leq \frac{1}{T_i(\delta)} \|\bar{\mathbf{x}}\|_A + 6 \sum_{j=1}^{i} i \|\mathbf{err}_j\|_A\end{aligned}$$

■

As the error bound guarantee of $\textsc{Solver}_B$ is relative, we need to inductively show that the total error is small. This then leads to the final error bound.

Proof of Lemma 5.A.1: The proof is by induction. We show that as long as $i < \kappa\epsilon^{-1}$, we have

$$\left\|\bar{\mathbf{x}} - \mathbf{x}_i\right\|_A \leq \left(\frac{1}{T_i\left(\delta\right)} + \frac{\epsilon^2 i}{2\kappa}\right)\left\|\bar{\mathbf{x}}\right\|_A,$$

and $\left\|\mathbf{err}_j\right\|_A \leq \frac{\epsilon^3}{24\kappa^2}\left\|\bar{\mathbf{x}}\right\|_A$ for all $j \leq i$.

The base case of $i = 0$ follows from $T_0(\delta) = 0$. For the inductive case, suppose the result is true for $i - 1$. Then as $i < \kappa\epsilon^{-1}$ and $T_i(\delta) \geq 1$, we have $\left\|\bar{\mathbf{x}} - \mathbf{x}_{i-1}\right\|_A \leq 2\left\|\bar{\mathbf{x}}\right\|_A$. As the vector passed to SOLVE$_B$ is $A\mathbf{x}_{i-1} - \mathbf{b} = A(\mathbf{x}_{i-1} - \bar{\mathbf{x}})$ and $B^\dagger \preceq A^\dagger$, we have

$$\begin{aligned}
\left\|A\left(\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right)\right\|_{B^\dagger} &= \sqrt{\left(\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right)^T AB^\dagger A\left(\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right)} && (5.8)\\
&\leq \sqrt{\left(\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right)^T A\left(\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right)} && (5.9)\\
&= \left\|\mathbf{x}_{i-1} - \bar{\mathbf{x}}\right\|_A && (5.10)\\
&\leq 2\left\|\bar{\mathbf{x}}\right\|_A. && (5.11)
\end{aligned}$$

Therefore the guarantees of SOLVER$_B$ gives $\left\|\mathbf{err}_i\right\|_B \leq \frac{\epsilon^3}{48\kappa^2}$. Combining this with $A \preceq B$ gives the bound on $\mathbf{err}_i$.

Substituting these bounds into Lemma 5.A.8 in turn gives the inductive hypothesis for i. The lower bound on $T_i(\delta)$ gives that when $i = \mathcal{O}(\sqrt{\kappa}\log(1/\epsilon))$, the first term is less than $\frac{\epsilon}{2}$. As $\log(1/\epsilon) \leq \frac{1}{\epsilon}$, the second term can be bounded by $\frac{\epsilon}{2}$ as well. Combining these two error terms gives the overall error.

■

We remark that the exponent on κ and ϵ in this analysis are not tight, and will be improved in a future version.

5.B Finding Electrical Flows

We now show that the solver given in Theorem 5.1.1 can also be used to find electrical flows in similar time. This problem can be viewed as the dual of computing vertex potentials, and is the core problem solved in the flow energy reduction based algorithms by Kelner et al. [KOSZ13] and Lee and Sidford [LS13]. As flows to defined on the edges of graphs instead of vertices, it is helpful to define the edge vertex incidence matrix.

Definition 5.B.1 *The edge-vertex incidence matrix of a weighted graph $G = (V, E)$ is given by*

$$\boldsymbol{B}_{e,u} = \begin{cases} 1 & \text{if } u \text{ is the head of } e \\ -1 & \text{if } u \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases}$$

It can be checked that if $\boldsymbol{R}$ is the diagonal matrix containing all the resistances, the graph Laplacian is given by $\boldsymbol{L} = \mathbf{B}^T \boldsymbol{R} \mathbf{B}$.

Given a flow $\boldsymbol{f}$, its residual at vertices is given by $\mathbf{B}^T \boldsymbol{f}$. Also, the energy of the flow is given by $\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f}) = \|\boldsymbol{f}\|_{\boldsymbol{R}}$. The electrical flow problem is finding the minimum energy flow whose residue meets a set of demands $\boldsymbol{d}$. It can be characterized as follows.

Fact 5.B.2 *For a demand $\boldsymbol{d}$, the minimum energy electrical flow $\bar{\boldsymbol{f}}$ is given by*

$$\bar{\boldsymbol{f}} = \boldsymbol{R}^{-1} \boldsymbol{B} \boldsymbol{L}^{\dagger} \boldsymbol{d},$$

and its energy, $\mathcal{E}_{\boldsymbol{R}}(\bar{\boldsymbol{f}})$ equals to $\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}$.

As a result, a natural algorithm for computing a flow that approximately minimizes electrical energy is to solve for approximate potentials $\boldsymbol{L}^{\dagger}\boldsymbol{d}$. Previous reductions between these problems such as the one by Christiano et al. [CKM$^+$11] ran the solver to high accuracy to recover these potentials. Then any difference between the residue and demands are fixed combinatorially. Here we show that this exchange can happen with low error in a gradual fashion. The following lemma is the key to our algorithm.

Lemma 5.B.3 *If $\boldsymbol{x}$ is a vector such $\|\boldsymbol{x} - \boldsymbol{L}^{\dagger}\boldsymbol{d}\|_{\boldsymbol{L}} \leq \epsilon \|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}$, then $\boldsymbol{f} = \boldsymbol{R}^{-1}\boldsymbol{B}\boldsymbol{x}$ is a flow such that $\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f}) \leq (1+\epsilon)\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}$, and the energy required to send the flow $\boldsymbol{d} - \boldsymbol{B}\boldsymbol{f}$ is at most $\epsilon\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}$.*

Proof

Both steps can be checked algebraically. For the energy of the flow, we have

$$\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f})^2 = (\boldsymbol{R}^{-1}\mathbf{B}\boldsymbol{x})^T \boldsymbol{R} (\boldsymbol{R}^{-1}\mathbf{B}\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x} = \|\boldsymbol{x}\|_{\boldsymbol{L}}^2.$$

Combining this with the error guarantees gives

$$\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f}) = \|\boldsymbol{x}\|_{\boldsymbol{L}} \leq \|\boldsymbol{L}^{\dagger}\boldsymbol{d}\|_{\boldsymbol{L}} + \|\boldsymbol{x} - \boldsymbol{L}^{\dagger}\boldsymbol{d}\|_{\boldsymbol{L}} \leq (1+\epsilon)\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}.$$

For the energy needed to reroute the demands, note that $\mathbf{B}\boldsymbol{f} = \boldsymbol{L}\boldsymbol{x}$. Substituting this in gives:

$$\|\mathbf{B}\boldsymbol{f} - \boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}} = \|\boldsymbol{L}\boldsymbol{x} - \boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}} = \|\boldsymbol{x} - \boldsymbol{L}^{\dagger}\boldsymbol{d}\|_{\boldsymbol{L}} = \epsilon\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}.$$

■

This means that we can solve the resulting re-routing problem to a much lower accuracy. This decrease in accuracy in turn allows us to change our graph, leading to a faster running time for this correction step.

Claim 5.B.4 *Given a graph $G = (V, E, \boldsymbol{r})$, a set of demands $\boldsymbol{d}$, and any error parameter $\epsilon > 0$ we can find in expected $O(m \log^{1/2} n poly(\log\log n)\log(\epsilon^{-1}))$ time a flow $\boldsymbol{f}$ such that with high probability $\boldsymbol{f}$ meets the demands, and $\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f}) \leq (1+\epsilon)\|\boldsymbol{d}\|_{\boldsymbol{L}^{\dagger}}$.*

Proof Consider running the solver given Theorem 5.1.1 to an accuracy of $\frac{\epsilon}{\log^3 n}$, and using the resulting flow $\boldsymbol{f}$. Lemma 5.B.3 then gives that it suffices to find another flow with a set of demands $\boldsymbol{d}'$ such that $\left\|\boldsymbol{d}'\right\|_{\boldsymbol{L}^\dagger} \leq \frac{\epsilon}{\log^3 n}\left\|\boldsymbol{d}\right\|_{\boldsymbol{L}^\dagger}$. As the energy of $\boldsymbol{f}$ is at most $(1+\frac{\epsilon}{\log^3 n})\left\|\boldsymbol{d}\right\|_{\boldsymbol{L}^\dagger}$, it suffices to find a flow $\boldsymbol{f}'$ meeting demands $\boldsymbol{d}'$ such that $\mathcal{E}_{\boldsymbol{R}}(\boldsymbol{f}') \leq \frac{\epsilon}{2}\left\|\boldsymbol{d}\right\|_{\boldsymbol{L}^\dagger}$.

The fact that we can tolerate a $\frac{\log^3 n}{2}$ factor increase in energy in $\boldsymbol{f}'$ allows us to find this flow on a graph with some resistances increased by the same factor. This allows us to reduce the value of $\|\boldsymbol{\tau}\|_p^p$ in Lemma 5.4.5 by a factor of about $\log^{3p} n$. It can also be checked that it suffices to find electrical flows a sparsified version of this graph. Therefore, the solve can be ran to an accuracy of $\frac{1}{\text{poly}(n)}$ on this smaller graph without being a bottleneck in the running time.

Adding this flow in means that we in turn need to find a flow for some demand $\boldsymbol{d}''$ with energy at most $\text{poly}(n)\left\|\boldsymbol{d}''\right\|_{\boldsymbol{L}^\dagger}$. As the relative condition number of the minimum spanning tree with the graph can be bounded by $\text{poly}(n)$, using it to reroute the flow allows us to arrive at the final flow.

∎

5.C Relation to Matrix Chernoff Bounds

We now show a matrix Chernoff bounds based analysis of our sampling routine which gives bounds that are off by log factors on each side with high probability. The matrix Chernoff bound that we will use is as follows:

Lemma 5.C.1 (Matrix Chernoff, Theorem 1.1 from [Tro12]) *Let $\boldsymbol{M}_k$ be a sequence of independent, random, self-adjoint matrices with dimension n. Assume that each random matrix satisfies $\boldsymbol{0} \preceq \boldsymbol{M}_k$ and $\lambda_{\max}(\boldsymbol{M}_k) \leq R$. Define $\mu_{\min} = \lambda_{\min}\left(\sum_k \mathbb{E}\left[\boldsymbol{M}_k\right]\right)$ and $\mu_{\max} = \lambda_{\max}\left(\sum_k \mathbb{E}\left[\boldsymbol{M}_k\right]\right)$. Then*

$$\mathbb{P}\left[\lambda_{\min}\left(\sum_k \mathbb{E}\left[\boldsymbol{M}_k\right]\right) \leq (1-\delta)\,\mu_{\min}\right] \leq n \cdot \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu_{\min}/R} \textit{for } \delta \in [0,1],$$

and

$$\mathbb{P}\left[\lambda_{\min}\left(\sum_k \mathbb{E}\left[\boldsymbol{M}_k\right]\right) \leq (1+\delta)\,\mu_{\max}\right] \leq n \cdot \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{\mu_{\max}/R} \textit{for } \delta \geq 0.$$

As this bound is tailored for low error, we need an additional smoothing step. Here the fact that we add $\boldsymbol{X}$ to the resulting sample is crucial for our analysis. It allows us to analyze the deviation between $\boldsymbol{Z} + \kappa\boldsymbol{X}$ and $\boldsymbol{Y} + \kappa\boldsymbol{X}$ for a parameter κ that we will pick. We will actually prove a generalization of both the $\delta = \frac{1}{\mathcal{O}(\log n)}$ case and the $\delta = \mathcal{O}(1)$ case.

Lemma 5.C.2 *There exists a constant c such that $\boldsymbol{Z} = \text{SAMPLE}(\{Y_1, \ldots, Y_m\}, X, \boldsymbol{\tau}, \delta)$ satisfies with high probability*

$$\frac{1}{c\delta \log n} \cdot \boldsymbol{Y} \preceq \boldsymbol{Z} \preceq c\delta \log n \cdot \boldsymbol{Y}.$$

Proof Note that our sampling algorithm also picks the number of samples, r, randomly between t and $2t-1$. However, as we can double c, it suffices to show the result of taking t samples is tightly concentrated.

Let $\kappa > 0$ be a parameter that we set later to about $\delta \log n$, and consider the approximation between $\boldsymbol{Z} + \kappa \boldsymbol{X}$ and $\boldsymbol{Y} + \kappa \boldsymbol{X}$. We let $\boldsymbol{M}_1 \ldots \boldsymbol{M}_t$ be the matrices corresponding to the samples, normalized by $\boldsymbol{Y} + \kappa \boldsymbol{X}$:

$$\boldsymbol{M}_i \stackrel{\text{def}}{=} \frac{\delta}{\boldsymbol{\tau}_{i_j}} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \boldsymbol{Y}_{i_j} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2}.$$

As all $\boldsymbol{Y}_i$s are positive semidefinite, this random matrix is also positive semidefinite. Its maximum eigenvalue can be bounded via its trace

$$\begin{aligned} \textbf{tr}\left[\boldsymbol{M}_i\right] &= \frac{\delta}{\boldsymbol{\tau}_{i_j}} \textbf{tr}\left[\left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \boldsymbol{Y}_{i_j} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \right] \\ &= \delta \frac{\textbf{tr}\left[\left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1} \boldsymbol{Y}_{i_j} \right]}{\textbf{tr}\left[\boldsymbol{X}^{-1} \boldsymbol{Y}_{i_j} \right]} \\ &\leq \frac{\delta}{\kappa}. \end{aligned}$$

Where the last inequality follows from $\left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1} \preceq \left(\kappa \boldsymbol{X} \right)^{-1} = \frac{1}{\kappa} \boldsymbol{X}^{-1}$. It can also be checked that $\mathbb{E}_{i_j} \left[\frac{\delta}{\boldsymbol{\tau}_{i_j}} \right] \boldsymbol{Y}_{i_j} = \frac{\delta}{s} \boldsymbol{Y}$, therefore

$$\mathbb{E}_{i_1 \ldots i_t} \left[\sum_{j=1}^{t} \boldsymbol{M}_j \right] = \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \boldsymbol{Y} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2}.$$

This gives $\mu_{\max} = 1$, but $\mu_{\min}$ can still be as low as $\frac{1}{1+\kappa}$. Note however that $\boldsymbol{Z}$ is formed by adding $\boldsymbol{X}$ to the result. Therefore, to improve the bounds we introduce $\delta^{-1}\kappa$ more matrices each equaling to $\delta \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \boldsymbol{X} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2}$. As $\left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right)^{-1/2} \preceq \frac{1}{\kappa} \boldsymbol{X}^{-1}$, the maximum eigenvalue in each of these is also at most $\frac{\delta}{\kappa}$. They on the other hand gives $\mathbb{E}\left[\sum_k \boldsymbol{M}_k \right] = \boldsymbol{I}$, and therefore $\mu_{\min} = \mu_{\max} = 1$.

Invoking Lemma 5.C.1 with $R = \delta^{-1}\kappa$ then gives that when $\kappa \stackrel{\text{def}}{=} c\delta \log n$, we have that the eigenvalues of $\sum_k \boldsymbol{M}_k$ are between $\frac{1}{2}$ and 2 with high probability. Rearranging using the fact that the samples taken equals to $\boldsymbol{Z} + (\kappa - 1)\boldsymbol{X}$ gives

$$\frac{1}{2} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right) \preceq \boldsymbol{Z} + (\kappa - 1)\boldsymbol{X} \preceq 2 \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right).$$

The $\boldsymbol{X}$ terms can then be removed using the fact that $\boldsymbol{0} \preceq \boldsymbol{X} \preceq \boldsymbol{Y}$, giving

$$\frac{1}{2} \boldsymbol{Y} \preceq \frac{1}{2} \left(\boldsymbol{Y} + \kappa \boldsymbol{X} \right) \preceq \boldsymbol{Z} + (\kappa - 1)\boldsymbol{X} \preceq \kappa \boldsymbol{Z},$$

for the lower bound, and

$$\boldsymbol{Z} \preceq \boldsymbol{Z} + (\kappa - 1)\boldsymbol{X} \preceq 2\left(\boldsymbol{Y} + \kappa \boldsymbol{X}\right) \preceq 2(\kappa + 1)\boldsymbol{Y},$$

for the upper bound. Recalling that $\kappa = c\delta \log n$ then gives the bound.

■

Invoking this lemma with $\delta = \mathcal{O}(1)$, and analyzing the amplification of error caused by sampling too many off-tree edges in the same way as Lemma 5.4.7 then gives Lemma 5.4.8.

5.D Propagation and Removal of Errors

As all intermediate solutions in our algorithms contain errors, we need to check that these errors propagate in a natural way across the various combinatorial transformations. We do this by adapting known analyses of the recursive preconditioning framework [ST06] and Steiner tree preconditioners [MMP$^+$05, Kou07] to a vector convergence setting. We also check that it suffices to perform all intermediate computations to a constant factor relative errors by showing an outer loop that reduces this error to ϵ in $O(\log(1/\epsilon))$ iterations.

5.D.1 Partial Cholesky Factorization

Lemma 5.4.4 *Given a graph-tree tuple $(H, T, \boldsymbol{\tau})$ with n vertices and m' off-tree edges, and and a Laplacian solver* SOLVER*, there is a routine* ELIMINATE&SOLVE$(H, T, \boldsymbol{\tau}, \text{SOLVER}, \boldsymbol{b}, \epsilon)$ *that for any input $\boldsymbol{b} = \boldsymbol{L}_H \bar{\boldsymbol{x}}$, performs $\mathcal{O}(n + m')$ operations plus one call to* SOLVER *with a graph-tree tuple $(H', T', \boldsymbol{\tau}')$ with $\mathcal{O}(m')$ vertices and edges, the same bounds for the stretch of off-tree edges, and accuracy ϵ and returns a vector $\boldsymbol{x}$ such that*

$$\left\|\bar{\boldsymbol{x}} - \boldsymbol{x}\right\|_{\boldsymbol{L}_H} \leq \epsilon \left\|\bar{\boldsymbol{x}}\right\|_{\boldsymbol{L}_H}.$$

Proof The greedy elimination procedure from Section 4.1. of [ST06] gives a factorization of $\boldsymbol{L}_H$ into

$$\boldsymbol{L}_H = \boldsymbol{U}^T \begin{pmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{L}_{H'} \end{pmatrix} \boldsymbol{U},$$

where $\boldsymbol{L}_{H'}$ has $O(m')$ vertices and edges and for any vector $\boldsymbol{y}$, both $\boldsymbol{U}^{-T}\boldsymbol{y}$ and $\boldsymbol{U}^{-1}\boldsymbol{y}$ can be evaluated in $O(n)$ time. It can also be checked that this elimination routine preserves the stretch of off-tree edges, giving a tree T' as well.

For notational simplicity, we will denote the block-diagonal matrix with $\boldsymbol{I}$ and $\boldsymbol{L}_{H'}$ as $\boldsymbol{P}$. Note that $\boldsymbol{I}$ and $\boldsymbol{L}_{H'}$ act on orthogonal subspaces since their support are disjoint and solving a linear system in $\boldsymbol{I}$ is trivial. This means that making one call to SOLVE with $(H', T', \boldsymbol{\tau}')$ plus $O(n)$ overhead gives solver routine for $\boldsymbol{P}$. More specifically, we have access to a routine SOLVE$_{\boldsymbol{P}}$ such that for any vector $\boldsymbol{b}'$, $\boldsymbol{x}' = \text{SOLVE}_{\boldsymbol{P}}(\boldsymbol{b}', \epsilon)$ obeys:

$$\left\|\boldsymbol{x}' - \boldsymbol{P}^{\dagger}\boldsymbol{b}'\right\|_{\boldsymbol{P}} \leq \epsilon \left\|\boldsymbol{P}^{\dagger}\boldsymbol{b}'\right\|_{\boldsymbol{P}}.$$

We can then check incorporating $\boldsymbol{U}^{-1}$ and $\boldsymbol{U}^{-T}$ the natural way preserves errors. Given a vector $\boldsymbol{b}$, we call SOLVE$_{\boldsymbol{P}}$ with the vector $\boldsymbol{b}' = \boldsymbol{U}^{-T}\boldsymbol{b}$, and return $\boldsymbol{x} = \boldsymbol{U}^{-1}\boldsymbol{x}'$. Substituting the error guarantees above gives

$$\left\| \boldsymbol{U}\boldsymbol{x} - \boldsymbol{P}^{\dagger}\boldsymbol{U}^{-T}\boldsymbol{b} \right\|_{\boldsymbol{P}} \leq \epsilon \left\| \boldsymbol{P}^{\dagger}\boldsymbol{U}^{-T}\boldsymbol{b} \right\|_{\boldsymbol{P}}.$$

Incorporating $\boldsymbol{L}_H^{\dagger} = \boldsymbol{U}^{-1}\boldsymbol{P}^{\dagger}\boldsymbol{U}^{-T}$ then gives

$$\left\| \boldsymbol{U}\left(\boldsymbol{x} - \boldsymbol{L}_H^{\dagger}\right)\boldsymbol{b} \right\|_{\boldsymbol{P}} \leq \epsilon \left\| \boldsymbol{b} \right\|_{\boldsymbol{L}_H^{\dagger}},$$

which simplifies to

$$\left\| \boldsymbol{x} - \boldsymbol{L}_H^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{L}_H} \leq \epsilon \left\| \boldsymbol{L}_H^{\dagger}\boldsymbol{b} \right\|_{\boldsymbol{L}_H}.$$

■

5.D.2 Transfer of Errors

Fact 5.4.6 *Let $\boldsymbol{\Pi}$ and $\boldsymbol{\Pi}_1$ be the two projection matrices defined in Lemma 5.4.5 Part 3. For a vector $\hat{\boldsymbol{b}}$, if $\boldsymbol{x}$ is a vector such that*

$$\left\| \boldsymbol{x} - \boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G} \leq \epsilon \left\| \boldsymbol{L}_G^{\dagger}\Pi^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G},$$

for some $\epsilon > 0$. Then the vector $\hat{\boldsymbol{x}} = \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{x}$ satisfies

$$\left\| \hat{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}} \leq \epsilon \left\| \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}}.$$

Proof We first check that the RHS terms are equal to each other by switching the matrix norms.

$$\left\| \boldsymbol{L}_G^{\dagger}\Pi^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G} = \left\| \Pi^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G^{\dagger}} = \left\| \hat{\boldsymbol{b}} \right\|_{\boldsymbol{\Pi}_1\Pi\boldsymbol{L}_G^{\dagger}\Pi^T\boldsymbol{\Pi}_1^T} = \left\| \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}}.$$

A similar manipulation of the LHS gives:

$$\left\| \hat{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}} = \left\| \boldsymbol{\Pi}_1\boldsymbol{\Pi}\left(\boldsymbol{x} - \boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right) \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}}.$$

Note that $\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}$ is the Schur complement of $\boldsymbol{L}_G$ on its rank space onto the column space of $\boldsymbol{\Pi}_1\boldsymbol{\Pi}$. As the Schur complement quadratic form gives the minimum energy over all extensions of the vector w.r.t. the original quadratic form, we have:

$$\left\| \boldsymbol{\Pi}_1\boldsymbol{\Pi}\left(\boldsymbol{x} - \boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}}\right) \right\|_{\left(\boldsymbol{\Pi}_1\boldsymbol{\Pi}\boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\right)^{\dagger}} \leq \left\| \boldsymbol{x} - \boldsymbol{L}_G^{\dagger}\boldsymbol{\Pi}^T\boldsymbol{\Pi}_1^T\hat{\boldsymbol{b}} \right\|_{\boldsymbol{L}_G}.$$

which when combined with the equality for the RHS completes the result.

```
1. $\boldsymbol{x} = \textsc{Solve}_{\mathbf{B}}\left(\boldsymbol{b}\right)$

2. Let $t = \log_\alpha(\frac{1}{\epsilon})$. For $i = 0...t$
   $\boldsymbol{y} = \textsc{Solve}_A\left(\mathbf{b} - A\mathbf{x}\right)$
   $\boldsymbol{x} = \mathbf{x} + \mathbf{y}$

3. Return $\mathbf{x}$
```

Figure 5.D.1: Preconditioned Richardson Iteration

5.D.3 Preconditioned Richardson Iteration

Lemma 5.D.1 *If $\boldsymbol{A}$, $\boldsymbol{B}$ are matrices such that $\boldsymbol{A} \preceq \boldsymbol{B} \preceq 2\boldsymbol{A}$ and $\textsc{Solve}_{\boldsymbol{B}}$ is a routine such that for any vector $\boldsymbol{b}$ we have $\left\|\textsc{Solve}_{\boldsymbol{B}}(\boldsymbol{b}) - A^{\dagger}\boldsymbol{b}\right\|_{\boldsymbol{B}} \leq \frac{1}{5}\left\|A^{\dagger}\boldsymbol{b}\right\|_{\boldsymbol{B}}$. There is a routine $\textsc{Solve}_{\boldsymbol{A},\epsilon}$ which runs in $\mathcal{O}(c_\alpha \log(\frac{1}{\epsilon}))$ iterations with the guarantee that for any vector $\boldsymbol{b}$ we have $\left\|\textsc{Solve}_{\boldsymbol{A},\epsilon}(\boldsymbol{b}) - A^{\dagger}\boldsymbol{b}\right\|_{\boldsymbol{A}} \leq \epsilon\left\|\boldsymbol{A}^{\dagger}\boldsymbol{b}\right\|_{\boldsymbol{A}}$. Each iteration involves one call to $\textsc{Solve}_{\boldsymbol{B}}$, a matrix-vector multiplication involving $\boldsymbol{A}$ and operations on vectors.*

Proof A pseudocode of the routine $\textsc{Solve}_B$ is given in Figure 5.D.1. It suffices to show that each iteration, $\left\|\mathbf{x} - A^{\dagger}\mathbf{b}\right\|_A$ decreases by a constant factor.

We will use $\boldsymbol{x}'$ to denote the solution vector produced for the next iteration. As our convergence is in terms of distance to the exact solution, it is convenient to denote the current error using $\boldsymbol{r} = \mathbf{x} - A^{\dagger}\mathbf{b}$.

Applying the triangle inequality to the new error gives:

$$\left\|\boldsymbol{x}' - \boldsymbol{A}^{\dagger}\mathbf{b}\right\|_{\boldsymbol{A}} = \left\|\boldsymbol{x} + \boldsymbol{y} - \boldsymbol{A}^{\dagger}\mathbf{b}\right\|_{\boldsymbol{A}} \leq \left\|\boldsymbol{x} - \boldsymbol{A}^{\dagger}\boldsymbol{b} + \mathbf{B}^{\dagger}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\right)\right\|_{\boldsymbol{A}} + \left\|\boldsymbol{y} - \boldsymbol{b}^{\dagger}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\right)\boldsymbol{r}\right\|_{\boldsymbol{A}}.$$

If $\boldsymbol{b}$ is in the column space of $\boldsymbol{A}$ and $\mathbf{B}$, $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}(\boldsymbol{A}^{\dagger}\boldsymbol{b} - \boldsymbol{x}) = -\boldsymbol{A}\boldsymbol{r}$. As the error is measured in the $\boldsymbol{A}$-norm, we can make this substitution, giving:

$$\left\|\boldsymbol{x}' - \boldsymbol{A}^{\dagger}\mathbf{b}\right\|_{\boldsymbol{A}} \leq \left\|\left(\boldsymbol{I} - \mathbf{B}^{\dagger}\boldsymbol{A}\right)\boldsymbol{r}\right\|_{\boldsymbol{A}} + \left\|\boldsymbol{y} - \mathbf{B}^{\dagger}\boldsymbol{A}\boldsymbol{r}\right\|_{\boldsymbol{A}}.$$

The first term equals to

$$\sqrt{\boldsymbol{r}^T\boldsymbol{A}^{1/2}\left(\boldsymbol{I} - \boldsymbol{A}^{1/2}\mathbf{B}^{\dagger}\boldsymbol{A}^{1/2}\right)^2\boldsymbol{A}^{1/2}\boldsymbol{r}}$$

Rearranging $\boldsymbol{A} \preceq \mathbf{B} \preceq 2\boldsymbol{A}$ gives $0 \preceq \boldsymbol{I} - \boldsymbol{A}^{1/2}\mathbf{B}^{\dagger}\boldsymbol{A}^{1/2} \preceq \frac{1}{2}\boldsymbol{I}$, which means the first term can be bounded by $\frac{1}{2}\left\|\boldsymbol{r}\right\|_{\boldsymbol{A}}$.

The second term can be bounded using the guarantees of $\textsc{Solve}_{\boldsymbol{A}}$ and the bounds between $\boldsymbol{A}$ and $\mathbf{B}$:

$$\left\|\boldsymbol{y} - \mathbf{B}^{\dagger}\boldsymbol{A}\boldsymbol{r}\right\|_{\boldsymbol{A}} \leq \left\|\boldsymbol{y} - \mathbf{B}^{\dagger}\boldsymbol{A}\boldsymbol{r}\right\|_{\mathbf{B}} \leq \alpha\left\|\boldsymbol{r}\right\|_{\mathbf{B}} \leq 2\alpha\left\|\boldsymbol{r}\right\|_{\boldsymbol{A}}.$$

Summing these two terms gives $\left\|\boldsymbol{x}' - \boldsymbol{A}^{\dagger}\mathbf{b}\right\|_{\boldsymbol{A}} \leq \frac{9}{10}\left\|\boldsymbol{x} - \boldsymbol{A}^{\dagger}\mathbf{b}\right\|_{\boldsymbol{A}}$, and therefore the convergence rate.

■

Chapter 6

Clustering and Routing for Directed Graphs

An outstanding open problem in spectral graph theory is the search for fast algorithms for maximum flow problems. The recent breakthroughs in the area [Mad13, LS14, KLOS14, She13] combine the combinatorial algorithms for undirected graphs with improved analysis of interior point methods for directed graphs. The combinatorial methods include fast Laplacian solvers and oblivious routings, both of which can be derived from low stretch trees and low diameter clustering algorithms.

Our goal is to initiate a study of combinatorial properties of directed graphs in the context of spectral approaches to flow problems. In particular, we focus our attention on *balanced graphs*. This class generalizes residual graphs (graphs that can be obtained as the residual of an approximate maximum s-t flow through an undirected graph) and Eulerian graphs.

We show and analyze a nearly-linear time low diameter clustering algorithm for balanced graphs, and show how to obtained a *low-stretch arborescence* for such graphs. In particular, this implies the existence of $\tilde{\mathcal{O}}(1)$-competitive single-source oblivious routings for balanced graphs. Currently, our algorithm for constructing these routings works in time $\tilde{\mathcal{O}}(mn)$. Similar algorithms for undirected graphs have been sped up in the past using various approaches [Mad10, RST14]. Bringing the runtime down is an open problem, and would immediately translate to faster maximum flow algorithms.

We also give new lower bounds for oblivious routing in directed graphs and a fast algorithm for computing maximum flow in balanced graphs.

The results presented in this chapter are joint work with Alina Ene, Gary Miller and Aaron Sidford [EMPS16].

6.1 Introduction

In this chapter, we study several fundamental routing questions in *directed graphs* that are *nearly* Eulerian. We introduce the notion of *balance* for directed graphs that quantifies how far away a graph is from being Eulerian[1]: a weighted directed graph is α-balanced if for every cut $S \subseteq V$, the

[1]A directed graph is Eulerian if, for each vertex, the total weight of its incoming edges is equal to the total weight of its outgoing edges. An equivalent definition is that for each cut $S \subseteq V$, the total weight of edges from S to $V \setminus S$ is equal to the total weight of edges from $V \setminus S$ to S.

total weight of edges going from S to $V \setminus S$ is within factor α of the total weight of edges going from $V \setminus S$ to S. Several important families of graphs are nearly balanced, in particular, Eulerian graphs (with $\alpha = 1$) and residual graphs of $(1+\epsilon)$-approximate undirected maximum flows (with $\alpha = \mathcal{O}(1/\epsilon)$).

We use the notion of balance to give a more fine-grained understanding of several well-studied routing questions that are considerably harder in directed graphs. The first question that we address is that of designing *oblivious routing* schemes for directed graphs. Oblivious routing schemes were introduced in the seminal work of Räcke [Räc02]. They are motivated by practical applications in routing traffic in massive networks such as the Internet, where it is necessary to route each request independently of the other requests and the current traffic in the network. Oblivious routing schemes were developed in a sequence of works [Räc02, ACF$^+$03, BKR03, HKLR05, HKLR06, HKRL07, Räc08, ER09]. In particular, if the graph is undirected, there exist oblivious routing schemes that achieve competitive ratio $O(\log n)$ [Räc08], where n is the number of nodes, and this result is optimal [BL99, MMVW97, MMVW97]. In contrast, Hajiaghayi *et al.* [HKRL07] show a strong lower bound of $\Omega(\sqrt{n})$ on the competitive ratio of routing obliviously in directed graphs. This lower bound holds even for *single-source* instances of bounded degree graphs, as well as for instances with symmetric demands.

In this chapter, we revisit oblivious routing in directed graphs, and we show that balanced graphs bridge the gap between directed and undirected graphs (see Section 6.3). Our main algorithmic result is an oblivious routing scheme for *single-source* instances that achieve an $\mathcal{O}(\alpha \cdot \log^3 n / \log\log n)$ competitive ratio. In the process, we make several technical contributions which may be of independent interest. In particular, we give an efficient algorithm for computing *low-radius decompositions* of directed graphs parameterized by balance. We also define and construct *low-stretch arborescences*, a new concept generalizing low-stretch spanning trees to directed graphs. Given the far-reaching implications of low-diameter decompositions and low-stretch spanning trees, we hope that our techniques may find other applications.

Our result is a generalization to directed graphs of Räcke's influential work [Räc08] that established a remarkable connection between oblivious routing in undirected graphs and metric embeddings into trees.

On the negative side, we present new lower bounds for oblivious routing problems on directed graphs. We show that the competitive ratio of oblivious routing algorithms for directed graphs has to be $\Omega(n)$ in general; this result improves upon the long-standing best known lower bound of $\Omega(\sqrt{n})$ [HKRL07]. We also show that the restriction to single-source instances is necessary by showing an $\Omega(\sqrt{n})$ lower bound for multiple-source oblivious routing in Eulerian graphs.

The second question that we study is that of finding an *approximate maximum flow* in balanced graphs. The maximum flow problem has received considerable attention in recent years, leading to several breakthrough results. This line of work has led to the development of almost linear time algorithms for approximate maximum flows in undirected graphs [KLOS14, She13] and the subsequent improvement of [Pen14, RST14]. In contrast, progress on directed graphs has been comparatively more modest, and the only improvements are the breakthrough results of Madry, yielding an $\tilde{\mathcal{O}}(m^{10/7})$-time algorithm for *unit-capacity* directed graphs with m edges [Mad13] and of Lee and Sidford, obtaining a running time of $\tilde{\mathcal{O}}(m\sqrt{n})$ for arbitrary directed graphs [LS14]. These improve over the long-standing best running time of $\tilde{\mathcal{O}}(m\min(\sqrt{m}, n^{2/3}))$ given by Goldberg and Rao [GR98].

In this chapter, we study the maximum flow problem in balanced directed graphs with arbitrary capacities (see Section 6.5). We develop an efficient algorithm that finds an $(1+\epsilon)$-approximate maximum flows in α-balanced graphs in time $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^2)$. Our algorithm builds on the work of Sherman [She13] and it can be viewed as an analogue of his result for directed graphs. The running time of our algorithm degrades gracefully with the imbalance of the graph and thus it suggests that balanced graphs provide a meaningful bridge between undirected and directed graphs.

We show that, using our approximate maximum flow algorithm, we can efficiently determine whether a given directed graph is α-balanced (see Section 6.5.2). Additionally, we give an application to the directed sparsest cut problem (see Section 6.5.3).

6.1.1 Related Work

Oblivious Routing. Oblivious routing schemes are well-studied and several results are known; we refer the reader to [Räc09] for a comprehensive survey of results for undirected graphs. As mentioned previously, in edge-weighted undirected graphs one can achieve a competitive ratio of $\mathcal{O}(\log n)$ [Räc08], and it is the best possible [BL99, MMVW97, MMVW97]. Hajiaghayi *et al.* [HKRL07] studied oblivious routing schemes in node-weighted undirected graphs and directed graphs. Their work gives an $\Omega(\sqrt{n})$ lower bound on the competitive ratio for both node-capacitated undirected graphs and directed graphs. They also show that these lower bounds still hold in more restricted settings, such as single-source instances. On the positive side, they give oblivious routing scheme with competitive ratios of $\mathcal{O}(\sqrt{n}\log n)$ for single-source instances in bounded-degree directed graphs, and $\mathcal{O}(\sqrt{k}n^{1/4}\log n)$ for general instances in directed graphs, where k is the number of commodities and in the worst case $k = \Theta(n^2)$.

Maximum s-t Flows. The maximum flow problem is one of the most central problems in combinatorial optimization and has been studied extensively over the past several decades. Until recently, most approaches have been based on combinatorial methods such as augmenting paths, blocking flows, push-relabel, etc. This line of work culminated in the seminal algorithm of Goldberg and Rao [GR98] that computes a maximum flow in time $O(\min(n^{2/3}, m^{1/2})\log(n^2/m)\log U)$ in directed graphs with integer weights that are at most U.

Over the past decade, a new approach emerged based on techniques drawn from several areas such as continuous optimization, numerical linear algebra, and spectral graph theory. These approaches led to a nearly-linear time algorithm for approximate maximum flows in *undirected graphs* [She13, KLOS14, Pen14], an $\tilde{\mathcal{O}}(m^{10/7})$-time algorithm for maximum flows in *unit-capacity directed graphs* [Mad13] and an $\tilde{\mathcal{O}}(m\sqrt{n})$-time algorithm for arbitrary directed graphs [LS14].

6.1.2 Organization

The rest of this chapter is organized as follows. In Section 6.2, we give an overview of our main results and introduce the definitions and notation we use throughout the chapter. In Section 6.3, we give our oblivious routing scheme for single-source instances. In Section 6.4, we state our lower bounds for oblivious routing. In Section 6.5 we give our approximate maximum flow algorithm and applications.

6.2 Overview

6.2.1 Basic Definitions

We study directed graphs $G = (V, E, w, l)$ with edge set $E \subseteq V \times V$, edge weights $w : E \to \mathbb{R}_+$ and edge lengths $l : E \to \mathbb{R}_+$. Throughout this chapter, we assume that G is strongly connected. In several applications we deal with graphs without weights or lengths. For graphs with edge lengths, we let $d(u, v)$ denote the shortest path distance from u to v.

We associate the following matrices with the graph G. The matrix of edge weights is defined as $\mathbf{C} \stackrel{\text{def}}{=} \mathrm{d}\mathbf{iag}(w)$ and the vertex-edge incidence matrix $\mathbf{B} \in \mathbb{R}^{V \times E}$ is defined as $\mathbf{B}_{s,(u,v)} \stackrel{\text{def}}{=} -1$ if $s = u$, 1 if $s = v$ and 0 otherwise. We are interested in finding flows that route demands with low congestion. The congestion incurred by a flow f is $\left\|\mathbf{C}^{-1} f\right\|_\infty$, and we say f routes demands b if $\mathbf{B}f = b$. The problem of finding a minimum congestion flow for a given demand vector, and its dual, the maximum congested cut, can be formulated as follows:

$$\begin{aligned} &\min_f. \quad \|\mathbf{C}^{-1} f\|_\infty \quad \text{s.t.} \quad && \mathbf{B}f = d, f \geq 0. \\ &\max_v. \quad b^\top v \quad \text{s.t.} \quad && \|\mathbf{C}\max(\mathbf{B}^\top v, 0)\|_1 \leq 1. \end{aligned}$$

We let OPT_b denote the optimum value of these problems. Throughout this chapter, we let $b_S = \sum_{u \in S} b_u$ and $w(S, T)$ denote the total weight of edges from S to T. It is well-known that for the second problem, one of the threshold cuts with respect to v achieves $b_S / w(S, V - S) \geq b^\top v$.

6.2.2 Balance

We parameterize strongly connected directed graphs by their *imbalance*:

Definition 6.2.1 (Imbalance) *Let $G = (V, E, w)$ be a strongly connected directed graph. We define its* imbalance*, $\mathrm{bal}(G)$, as the minimum α such that $w(S, V \setminus S) \leq \alpha \cdot w(V \setminus S, S)$ for every $S \subseteq V$.*

Two canonical families of balanced graphs are Eulerian graphs. and residual graphs of approximate undirected maximum flows.

Fact 6.2.2 *A strongly connected directed graph G is Eulerian if and only if $\mathrm{bal}(G) = 1$. If G is the residual graph of a $(1 + \epsilon)$-approximate undirected maximum flow, then $\mathrm{bal}(G) = \mathcal{O}(\epsilon^{-1})$.*

Theorem 6.2.3 (Equivalent definitions of balance) *Let $G = (V, E, w)$ be a directed graph. The following statements are equivalent:*

1. $\mathrm{bal}(G) \leq \alpha$.

2. There exists a circulation f on G with all edge congestions in $[1, \alpha]$.

3. Let $d = \mathbf{B}\vec{1}$ be the residual degrees in G. Then $-d$ can be routed with congestion $\alpha - 1$.

6.2.3 Oblivious Routing Schemes

An oblivious routing scheme is a linear operator that, for each source-destination pair $(s, t) \in V \times V$, specifies how to route one unit of flow from s to t independently of the other pairs. Given a *demand vector* $\vec{d} : D \to \mathbb{R}_+$ on a set $D \subseteq V \times V$ of source-sink pairs, one can produce a multi-commodity flow that meets these demands by routing each demand pair using the (pre-specified) operator, independently of the other demands. The *competitive ratio* of an oblivious routing scheme is the worst ratio among all possible demand vectors between the congestion of the multi-commodity flow given by the scheme and the congestion of the minimum congestion multi-commodity flow for the given demand vector.

Our main positive result concerning oblivious routings, given in Section 6.3, is the existence of good single-source oblivious routings for balanced graphs. A single-source oblivious routing with source $s \in V$ has $D = \{s\} \times V$.

Theorem 6.2.4 (Single Source Oblivious Routings) *Every strongly connected graph G admits a single-source oblivious routing, from any source, with competitive ratio* $\mathcal{O}(\mathrm{bal}(G) \cdot \log^3 n / \log\log n)$.

We achieve this result by generalizing an algorithm for undirected graphs given by Racke [Räc08]. The core difficulty that we need overcome is to find a good way to cluster the vertices of a directed balanced graph. We define the *radius* of a cluster $C \subseteq V$ as $\min_{u \in C} \max_{v \in C} d(u, v)$.. The *volume* $\mathrm{vol}(G)$ of G is defined as $\mathrm{vol}(G) \stackrel{\text{def}}{=} \sum_{e \in E} l(e)w(e)$. Our clustering algorithm is presented in Section 6.3.1, and its guarantees can be formalized as follows:

Theorem 6.2.5 (Balanced Graph Clustering) *Let $G = (V, E, w, l)$ be a directed graph. Then for every $r > 0$, V can be partitioned into clusters such that every cluster has radius at most r, and the total weight of edges going between different clusters is* $\mathcal{O}(\mathrm{bal}(G)\mathrm{vol}(G) \log n / r)$. *Moreover, such a partition can be found in expected linear time.*

The guarantees of Theorem 6.2.5 for undirected graphs match those given by prior work [Awe85, AKPW95, Bar96, MPX13]. Extending the statement to directed graphs is nontrivial, as it requires making the notion of cluster radii directed.

In Section 6.4 we give a new lower bound for all-pairs oblivious routings in directed graphs.

Theorem 6.2.6 *No oblivious routing algorithm for directed graphs can guarantee competitive ratio better than* $\Omega(n)$.

We also show that restricting ourselves to single-source oblivious routings is necessary to achieve a small competitive ratio even when $\mathrm{bal}(G) = 1$.

Theorem 6.2.7 *No oblivious routing algorithm for Eulerian graphs can guarantee competitive ratio better than $\Omega(\sqrt{n})$.*

6.2.4 Maximum Flows

Finally, we consider the maximum s-t flow problem in directed graphs parameterized by balance. Given a source s and a destination t, the maximum s-t flow problem asks us to find a flow f that routes as much flow as possible from s to t while sending at most w_e units of flow along each edge e. In Section 6.5 we show the following result.

Theorem 6.2.8 (Approximate Maximum Flow) *Given a strongly connected directed graph G, a source s, and a sink t there is an algorithm that finds a $(1+\epsilon)$-approximate maximum s-t flow and a $(1-\epsilon)$-approximate minimum s-t cut in G in time $\tilde{\mathcal{O}}(m \cdot \mathrm{bal}(G)^2/\epsilon^2)$.*

To achieve quadratic dependency on ϵ, in Section 6.5.4 we provide a general analysis of gradient descent for composite function minimization under non-Euclidean norms.

We also show applications of this result to computing the sparsest cut (Section 6.5.3) and we prove the following result on computing the imbalance of a graph (Section 6.5.2).

Lemma 6.2.9 *There is an algorithm that either certifies that $\mathrm{bal}(G) \le \alpha$ or shows that $\mathrm{bal}(G) > (1-\epsilon)\alpha$ in time $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^2)$.*

6.3 Oblivious Routing on Balanced Graphs

6.3.1 Low-radius Decompositions

Our algorithm for clustering directed graphs, presented in Figure 6.1, is based on the scheme given by Miller, Peng and Xu [MPX13]. We first pick a start time x_v for every vertex v from an exponential distribution, and then explore the graph, starting the search from v at time x_v and proceeding at unit speed. Each vertex u is assigned to the vertex v that reached it first.

Our goal is to show that this procedure *cuts* few edges, i.e. assigns the endpoints of few edges to different clusters. The original analysis of [MPX13] shows that for undirected graphs, this approach guarantees cutting each edge e with low probability, namely $\mathcal{O}(l(e) \log n/r)$. It turns out that even in the case of unweighted Eulerian graphs such a guarantee no longer holds; there may exist edges that are cut with very high probability. Consider for instance (Figure 6.2) a directed cycle of length 3^k, with an undirected star of 2^{k^2} leaves attached to one of its vertices, v. Set $r := 2^k$. Let u be the vertex preceding v on the cycle. It is now easy to verify by calculation that the edge (u, v) is cut with probability arbitrarily close to 1 for a large enough k. With high probability, v will be

```
$(V_1, V_2, \ldots) = $ CLUSTER-DIRECTED$(G, r)$, where $G = (V, E, l)$ is a directed graph and $r > 0$.

    1. Set $\beta := \log n/(10r)$.
    2. For every vertex $v \in V$ pick $x_v \sim \mathrm{Exp}(\beta)$.[2]
    3. For each vertex $u \in V$, assign $u$ to the cluster rooted at the vertex $v \in V$ which
       minimizes $-x_v + d(v, u)$.
    4. If any of the clusters has radius greater than $r$, return to step 2. Otherwise, return the
       clusters.
```

Figure 6.1: The low-radius decomposition algorithm for directed graphs.

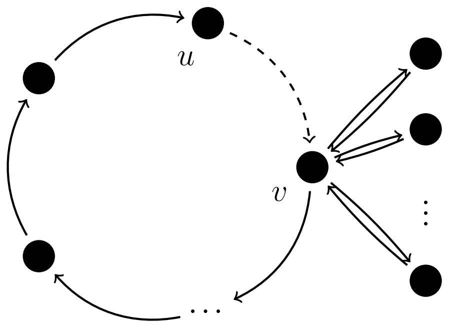


Figure 6.2: An unweighted Eulerian graph where a particular edge is very likely to be cut by the scheme of [MPX13].

contained in a cluster rooted at one of the 2^{k^2} leaves attached to it; also with high probability, no such cluster will contain u.

This issue requires us to find a new way to guarantee that the total weight of cut edges is low. Our key idea is to show that, for any fixed *cycle*, the expected number of edges in the cycle that are cut is small. The desired guarantees then follow by noting that any graph G can be approximated up to a factor $\mathrm{bal}(G)$ by a sum of cycles (Theorem 6.2.3).

Lemma 6.3.1 *Let $\mathcal{P}$ be the partition returned by* CLUSTER-DIRECTED(G, r). *For any simple cycle C in G, the expected number of edges in C that go between different clusters in $\mathcal{P}$ is an $\mathcal{O}(\log n/r)$ fraction of the length of C.*

As the above example demonstrates, we cannot base the proof of Lemma 6.3.1 on the *location* of the cuts, as it might depend strongly on the input graph. However, we can prove that, intuitively, cuts occur *infrequently* as the graph is explored. This is the crucial idea of the proof: we analyze the occurrence of cuts over time rather than bounding the probabilities of particular cuts. Then we use

[2]$\mathrm{Exp}(\beta)$ is the exponential distribution with parameter β, with p.d.f. $f(x) = \beta e^{-\beta x}$ on $x \geq 0$.

the fact that a cycle of length L is fully explored within L time steps after the time it is visited for the first time. The analysis is presented in Appendix 6.B.

6.3.2 Low-stretch Arborescences

Let G be a directed graph and let s be a vertex in G. We say that a directed graph T is an *arborescence* rooted at s for every vertex v, there is a unique directed path in T from s to v. In this section, we define and construct *low-stretch arborescences*, which are a key intermediate step between low-radius decompositions and oblivious routings.

Definition 6.3.2 *Let $G = (V, E, w, l)$ be a directed graph. We define the* stretch *of an edge $(u, v) \in E$ with respect to an arborescence T on the vertex set V as $w(u, v) \cdot d_T(u, v)$, where $d_T(u, v)$ is the distance between u and v in the* ***undirected*** *tree corresponding to T.*

Following the notation of [Räc08], we define the *load*, $\mathrm{load}_T(e)$, of an edge $e \in T$ as the sum of the weights of edges $(u, v) \in E(G)$ such that e is on the path between u and v in the *undirected* tree corresponding to T. Note that the total load of the edges in T is equal to the total stretch of the edges in G.

In order to construct low-stretch arborescences, we will recursively cluster V using the algorithm from the previous section. The algorithm FIND-ARBORESCENCE is presented in Figure 6.3. It is similar to the scheme given by Bartal [Bar96]. One major difficulty is that the clusters returned by CLUSTER-DIRECTED may be very imbalanced; in particular, they need not be strongly connected. In order to resolve this issue, we introduce the notion of *additive imbalance* and prove that our clustering algorithms still give good guarantees for graphs with low additive imbalance.

Definition 6.3.3 *We define the additive imbalance* $\mathrm{abal}(G)$ *of a directed graph G as the minimum ι such that it is possible to add edges of total weight ι to G to make it Eulerian.*

In order to make the running time of our algorithm independent of the diameter of the graph, we will attempt to *collapse* very short edges in the upper levels of the recursion, that is, contract their endpoints into a single vertex. This is similar to the scheme proposed in [CMP$^+$14, CKM$^+$14]. However, this operation is not always feasible in directed graphs; thus, we will only perform the contraction if both endpoints of the edge can reach each other by following only very short edges.

Definition 6.3.4 *Let $G = (V, E, w, l)$ be a directed graph and $x_L, x_R \in \mathbb{R}$ be such that $0 < x_L < x_R$. We construct G* collapsed *to $[x_L, x_R]$ by:*

- *merging any vertices that can reach each other while following only arcs of length at most x_L, and*
- *reducing the length of all arcs longer than x_R to x_R.*

```
$T = $ FIND-ARBORESCENCE$(G, s)$, where $G = (V, E, l)$ is a directed graph and $s \in V$ is
such that all vertices in $G$ are reachable from $s$.

 1. If $n = 1$, return a single-vertex graph.
 2. Let $r := \max_{v \in V} d_G(s, v)$.
 3. Let $r' := r/(c \cdot \log n)$.
 4. Let $G'$ be the graph $G$ collapsed to $[r'/n, 2r']$. Let $s'$ be the vertex in $G'$ corresponding
    to $s$.
 5. Let $V_1', V_2', \ldots, V_k' :=$ CLUSTER-DIRECTED-ROOTED$(G', s', r')$.
 6. Expand the clusters $V_1', \ldots, V_k'$ back into $G$, obtaining $V_1, \ldots, V_k$.
 7. Let $G_i$ be the graph induced by $V_i$, for $i = 1, \ldots k$, and $u_i$ denote the center of cluster
    $V_i$ (with $u_1 = s_1$).
 8. Let $T' := \bigcup_{i=1}^{k}$ FIND-ARBORESCENCE$(G_i, u_i)$.
 9. Let $T$ be $T'$ with the arcs $(s, u_i)$ of length $d_G(s, u_i)$ added for each $i = 2, \ldots, k$.
10. Return $T$.
```

Figure 6.3: The low-stretch arborescence finding algorithm.

Lemma 6.3.5 *Let $s \in V, r > 0$. Let $V_1, \ldots, V_k =$ CLUSTER-DIRECTED-ROOTED(G, s, r). Then:*

- *each cluster V_i has radius at most r,*
- *the cluster V_1 containing s has radius at most r from s,*
- *the expected total weight of edges going between different clusters is $\mathcal{O}(\mathrm{vol}(G)\log n/r + \mathrm{abal}(G)\log n)$, and*
- *the expected total additive imbalance of the clusters is $\mathcal{O}(\mathrm{vol}(G)\log n/r + \mathrm{abal}(G)\log n)$.*

Moreover, the algorithm works in expected linear time.

Proof First, note that the expected total weight of edges between V_1 and $V - V_1$ is $\mathcal{O}(\mathrm{vol}(G)/r)$. Hence the expected additive imbalances of the cluster on V_1 and that of G' are both $\mathcal{O}(\mathrm{abal}(G) + \mathrm{vol}(G)/r)$.

By the definition of additive imbalance, we can add edges of expected total weight $\mathcal{O}(\mathrm{abal}(G) + \mathrm{vol}(G)/r)$ to G' to make it Eulerian. We obtain the graph G'' by adding such edges, each with length $2r$. The expected volume of G'' is $\mathcal{O}(\mathrm{vol}(G)) + \mathcal{O}(\mathrm{abal}(G) + \mathrm{vol}(G)/r) \cdot 2r = \mathcal{O}(\mathrm{vol}(G) +$

```
$(V_1, V_2, \ldots) = $ CLUSTER-DIRECTED-ROOTED$(G, s, r)$, where $G = (V, E, l)$ is a directed
graph, $s \in V$ and $r > 0$.

  1. Choose $r'$ uniformly at random from $[0, r]$.
  2. Let $V_1$ be the set of vertices at distance at most $r'$ from $s$.
  3. Let $G'$ be the induced graph on $V - V_1$.
  4. Let $V_2, V_3, \ldots V_k :=$ CLUSTER-DIRECTED$(G', r)$.
  5. Return $V_1, V_2, \ldots, V_k$.
```

Figure 6.4: The decomposition algorithm with a specified root.

$\text{abal}(G) \cdot r)$. Now by Theorem 6.2.5 we can partition G'' into clusters of radius at most r, with the expected total weight of edges going between clusters $\mathcal{O}(\text{vol}(G'') \log n / r) = \mathcal{O}(\text{vol}(G) \log n / r + \text{abal}(G) \log n)$. Note that if we remove the added edges, the radii of these clusters cannot change, as the edges have length greater than r; at the same time, their total additive imbalance can increase by at most $\mathcal{O}(\text{abal}(G) + \text{vol}(G)/r)$ in expectation. To complete the analysis, observe that in fact the edges added in the above reasoning are ignored by the decomposition algorithm. Hence, they are only necessary for the analysis.

■

Lemma 6.3.6 *Let $G = (V, E, w, l)$ be a directed Eulerian graph and $x_L, x_R \in \mathbb{R}$ be such that $0 < x_L < x_R$. Let $G' = (V', E', w', l')$ be G collapsed to $[x_L, x_R]$. Then $\text{vol}(G')$ is at most*

$$2 \cdot \sum_{e \in E : l(e) > x_L/n} w(e) \min(l(e), x_R).$$

Proof Since G' is Eulerian, it can be represented as a sum of simple cycles of uniform weight. Consider any such decomposition and take any cycle C in it. Then C must contain an edge of length at least x_L, and it contains at most n edges of length not exceeding x_L/n. Hence, the length of C is at most two times greater than the sum of its edge lengths greater than x_L/n. Summing over all the cycles yields the desired bound.

■

Theorem 6.3.7 *Let $G = (V, E, w, l)$ be a strongly connected directed graph. Let $s \in V$. Let $T =$* FIND-ARBORESCENCE(G, s). *Then:*

- *T has vertex set V and is rooted at s,*
- *every arc (u, v) in T can be mapped to a path from u to v in G of equal length, and*

- *the expected total stretch of G with respect to T is* $\mathcal{O}(\mathrm{bal}(G)\mathrm{vol}(G)\log^3 n/\log\log n)$.

Moreover, the algorithm works in expected $\mathcal{O}(m\log n)$ *time.*

Proof First, note that by Theorem 6.2.3 the edge weights in G can be increased to obtain an Eulerian graph with volume at most $\mathrm{bal}(G)\mathrm{vol}(G)$. Since the algorithm is oblivious to weights, it is enough to consider Eulerian graphs in the proof; from now on we assume $\mathrm{bal}(G) = 1$.

Properties 1 and 2 are easy to verify. Assume the constants hidden in the big-oh notation in Lemma 6.3.5 are bounded by c_0. We set $c := 2c_0 + 4$.

Consider the i-th level ($i \geq 0$) of the tree of recursive calls of FIND-ARBORESCENCE(G, s). Let $r_i = r/(c\log n)^i$. It can easily be shown by induction that the radii of the graphs in the i-th level are at most r_i, and the radii of the returned arborescences are at most $2r_i$, since $c \geq 4$. Let ν_i be the total volume of the *collapsed* graphs at level i.

By Lemma 6.3.5 the additive imbalance of the graphs in the i-th level can be bounded by

$$\begin{aligned}&(c_0\log n)^i \cdot \nu_0/r_1\\ &+(c_0\log n)^{i-1} \cdot \nu_1/r_2\\ &+(c_0\log n)^{i-2} \cdot \nu_2/r_3\\ &+\dots\\ &+(c_0\log n)^1 \cdot \nu_{i-1}/r_i.\end{aligned}$$

Since $c > 2c_0$, the above sum is bounded by

$$(c\log n)^{i+1}\sum_{j<i}\left(\nu_j/2^{i-j}\right).$$

Hence, the total weight of edges cut at level i is at most

$$(c_0\log n)\left(\nu_i/r_{i+1} + (c\log n)^{i+1}\sum_{j<i}\left(\nu_j/2^{i-j}\right)\right) \leq (c\log n)^{i+2}/2\cdot\sum_{j\leq i}\left(\nu_j/2^{i-j}\right).$$

Since the radius of the arborescence returned at level i is at most $2r_i$, we have that the total stretch incurred at level i is at most

$$2r_i\cdot(c\log n)^{i+2}/2\cdot\sum_{j\leq i}\left(\nu_j/2^{i-j}\right). \leq (c\log n)^2\cdot\sum_{j\leq i}\left(\nu_j/2^{i-j}\right).$$

Hence the total stretch is at most

$$(c \log n)^2 \cdot \sum_i \sum_{j \le i} \left(\nu_j / 2^{i-j}\right) = (c \log n)^2 \cdot \sum_j \left(\nu_j 2^j \sum_{i \ge j} 2^{-i}\right)$$
$$\le 2(c \log n)^2 \cdot \sum_j \nu_j.$$

Observe that all the collapsed graphs at level j are subgraphs of G collapsed to $[r_{j+1}/n, 2r_{j+1}]$. Hence, by Lemma 6.3.6, we have

$$\nu_j \le 2 \cdot \sum_{e \in E : l(e) > r_{j+1}/n^2} w(e) \min(l(e), 2r_{j+1}).$$

Hence

$$\sum_j \nu_j \le 2 \cdot \sum_{e \in E} \sum_{j : r_{j+1} < l(e) \cdot n^2} w(e) \min(l(e), 2r_{j+1})$$
$$= \mathcal{O}(\mathrm{vol}(G) \log n / \log \log n).$$

Combining this with the previous bound yields the thesis.

■

6.3.3 Constructing the Routing

Given an algorithm for constructing low-stretch arborescences, we can use it to compute a good oblivious routings using the approach proposed by [Räc08]. The oblivious routing that we construct for a given source s will be a convex combination of arborescences rooted at s, with the flow for demand (s, u) being defined as the convex combination of the corresponding paths. The algorithm is given in Figure 6.5.

The key idea we employ to extend the analysis of the algorithm to a directed graph G is to prove that the routing scheme we construct is competitive even for the *undirected* graph underlying G.

Lemma 6.3.8 ([Räc08], adapted) *Let G be a strongly connected directed graph and s be a vertex in G. Let $((T_1, \lambda_1), \dots, (T_k, \lambda_k)) :=$ FIND-ROUTING(G, s). Then with high probability $((T'_1, \lambda_1), \dots, (T'_k, \lambda_k))$ is an $\mathcal{O}(\mathrm{bal}(G) \log^3 n / \log \log n)$-competitive oblivious routing for G', where $T'_1, \dots, T'_k, G'$ are the undirected counterparts of $T_1, \dots, T_k$ and G, respectively, that we obtain by ignoring the directions.*

Proof It is enough to note that in step 2c) of Figure 6.5, with high probability, T'_k is a tree with total stretch $\mathcal{O}(\mathrm{bal}(G) \log^3 n / \log \log n)$ in G'_k, where T'_k and G'_k are undirected counterparts of T_k and G_k, respectively. Hence, the analysis of [Räc08] can be applied to complete the proof.

■

```
$((T_1, \lambda_1), \ldots, (T_k, \lambda_k)) = \textsc{Find-Routing}(G, s)$ where $G = (V, E, w)$ is a strongly connected directed graph and $s \in V$.

  1. Set $k := 0$ and $p_e^{(0)} := 1$ for all $e \in E$.
  2. While $\sum_{i=1}^{k} \lambda_i < 1$:
      (a) $k := k + 1$.
      (b) Let $G_k = (V, E, l_k)$ be a copy $G$ with edge lengths
              $l_k(e) := p_e^{(k-1)} / \left( w(e) \sum_{e'} p_{e'}^{(k-1)} \right).$
      (c) $T_k := \textsc{Find-Arborescence}(G, s)$ (pick the minimum-stretch arborescence
          out of $\mathcal{O}(\log n)$ runs).
      (d) $\ell_k := \max_e \{\text{load}_{T_k}(e)/w(e)\}$.
      (e) $\lambda_k := \min\left(1/\ell_k, 1 - \sum_{i=1}^{k-1} \lambda_i\right)$.
      (f) For all edges $e$ set:
              $p_e^{(k)} := p_e^{(k-1)} \cdot \exp(\lambda_k \cdot \text{load}_{T_k}(e)/w(e)).$
  3. Return $((T_1, \lambda_1), \ldots, (T_k, \lambda_k))$.
```

Figure 6.5: The algorithm for finding single-source oblivious routings on balanced graphs (adapted from [Räc08]).

In order to finish the analysis, we only need to note that $((T_1, \lambda_1), \ldots, (T_k, \lambda_k))$ is an oblivious routing for G.

Proof of Theorem 6.2.4: We prove that for any s, the output of $\textsc{Find-Routing}(G, s)$ satisfies the criteria stated in the theorem statement. It follows from Lemma 6.3.8 that with high probability, $((T_1', \lambda_1), \ldots, (T_k', \lambda_k))$ is an $\mathcal{O}(\text{bal}(G) \log^3 n / \log\log n)$-competitive oblivious routing for G', where $T_1', \ldots, T_k', G'$ are undirected counterparts of $T_1, \ldots, T_k$, G, respectively. In particular, it is also an $\mathcal{O}(\text{bal}(G) \log^3 n / \log\log n)$-competitive oblivious routing from s. Now it is enough to observe that since $T_1, \ldots, T_k$ are directed away from s, $((T_1, \lambda_1), \ldots, (T_k, \lambda_k))$ is an oblivious routing from s in G. Since it is $\mathcal{O}(\text{bal}(G) \log^3 n / \log\log n)$-competitive in G', it must also be $\mathcal{O}(\text{bal}(G) \log^3 n / \log\log n)$-competitive in G.

■

6.4 Lower Bounds

We prove new lower bounds for oblivious routings in directed graphs.

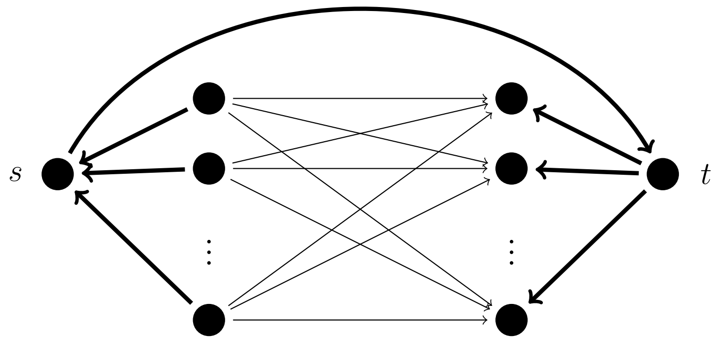


Figure 6.1: The example from Theorem 6.2.6. The thick edges have weight n, the other edges have weight 1. Any oblivious routing must put too much flow on the edge (s,t) when routing between the vertices of the biclique.

Theorem 6.2.6 *No oblivious routing algorithm for directed graphs can guarantee competitive ratio better than* $\Omega(n)$.

Proof of Theorem 6.2.6: Let $k \geq 1$. Let G be a directed graph on the vertex set

$$V = S \cup T \cup \{s\} \cup \{t\}, \text{ where } |S| = |T| = k$$

and edge set

$$\begin{aligned} E = {} & S \times T \text{ with weight } 1 \\ & \cup S \times \{s\} \text{ with weight } k \\ & \cup \{(s,t)\} \text{ with weight } k \\ & \cup \{t\} \times T \text{ with weight } k. \end{aligned}$$

Assume some oblivious routing $\mathcal{A}$ achieves competitive ratio c on G. Let $u \in S$ and $v \in T$. The optimal congestion for the unit flow from u to v is at most $1/k$, which can be achieved by routing the flow through s and t. Therefore, $\mathcal{A}$ must achieve congestion at most c/k, hence putting at least $1 - c/k$ units of flow on the edge (s,t).

The optimal congestion for the multicommodity flow with unit demand between every pair in $S \times T$ is clearly at most 1. Simultaneously, by the above argument, $\mathcal{A}$ must put at least $k(k-c)$ flow on the edge (s,t). Hence we have $c \geq k - c$, implying $c \geq k/2$. As $n = 2k + 2$ we have $c = \Omega(n)$.

■

Theorem 6.2.7 *No oblivious routing algorithm for Eulerian graphs can guarantee competitive ratio better than* $\Omega(\sqrt{n})$.

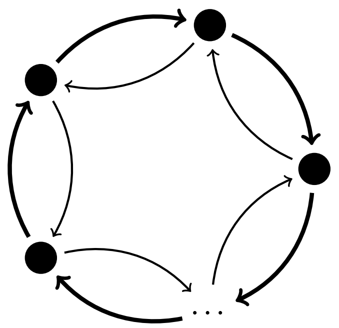


Figure 6.2: The example from Theorem 6.2.7. The thick edges have weight $\sqrt{n}$, the other edges have weight 1. Any oblivious routing must put too much flow on the outer cycle when routing between consecutive vertices of the inner cycle.

Proof of Theorem 6.2.7: Let $n \geq 2$. Let G be a directed graph on the vertex set

$$V = \{v_1, \ldots, v_n\}$$

and edge set

$$\begin{aligned} E &= C_1 \cup C_2, \text{ where} \\ C_1 &= \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)\} \text{ with weight } 1\text{, and} \\ C_2 &= \{(v_n, v_{n-1}), (v_{n-1}, v_{n-2}), \ldots, (v_2, v_1), (v_1, v_n)\} \text{ with weight } \sqrt{n}. \end{aligned}$$

Note that G is Eulerian. Assume some oblivious routing $\mathcal{A}$ achieves competitive ratio c on G. Let $i < n$. The optimal congestion for the unit flow from v_i to v_{i+1} is at most $1/\sqrt{n}$, which can be achieved by routing the flow through C_2. Therefore, $\mathcal{A}$ must achieve congestion at most $c/\sqrt{n}$, hence putting at least $1 - c/\sqrt{n}$ units of flow on the edge $(v_n, 1)$.

The optimal congestion for the multicommodity flow with unit demand between every such pair (v_i, v_{i+1}) is clearly at most 1. Simultaneously, by the above argument, $\mathcal{A}$ must put at least $(n-1)(1 - c/\sqrt{n})$ flow on the edge $(v_n, 1)$. Hence we have $c \geq (n-1)/\sqrt{n} - c$, implying $2c \geq \sqrt{n-1}$. Therefore $c = \Omega(\sqrt{n})$.

■

6.5 Maximum Flow and Applications

6.5.1 Directed Maximum Flow

In this subsection we show how to efficiently compute an $(1+\epsilon)$-approximate maximum flow in directed graphs given a good *congestion-approximator*.

Definition 6.5.1 *An α-congestion-approximator for G is a matrix $\mathbf{R}$ such that for any demand vector b, $\|\mathbf{R}b\|_\infty \leq OPT_b \leq \alpha\|\mathbf{R}b\|_\infty$.*

Since $\left\|\mathbf{R}b\right\|_\infty = \left\|-\mathbf{R}b\right\|_\infty$, only well-balanced graphs admit good congestion approximators:

Fact 6.5.2 *If G admits an α-congestion approximator, $\mathrm{bal}(G) \leq \alpha$.*

For undirected graphs, $\tilde{\mathcal{O}}(1)$-congestion-approximators can be computed in nearly linear time [Mad10, She13, KLOS14, Pen14]. This implies that for directed G we can compute $\tilde{\mathcal{O}}(\mathrm{bal}(G))$-congestion-approximators in nearly linear time by the following fact:

Fact 6.5.3 *Let G be a directed graph and G' be its undirected copy. Then for any demand vector b $OPT_b(G') \leq OPT_b(G) \leq (1 + \mathrm{bal}(G))OPT_b(G')$.*

Our main result is the following:

Theorem 6.5.4 *Let G be a directed graph. Given an α-congestion-approximator $\mathbf{R}$, we can compute an $(1+\epsilon)$-approximate maximum flow and minimum congested cut for any demand vector in time $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^2)$, assuming multiplication by $\mathbf{R}$ and $\mathbf{R}^\top$ can be done in $\tilde{\mathcal{O}}(m)$ time.*

Our algorithm is based very heavily on the approach for undirected graphs given by Sherman [She13]. The main difference is the implementation of the key optimization procedure, presented in Figure 6.1. In this section we only outline the main changes needed to extend the algorithm of [She13] to balanced graphs.

Let G be a directed graph and b be a demand vector. Assume we are given an α-congestion-approximator $\mathbf{R}$. Let $\mathrm{lmax}(x) \stackrel{\text{def}}{=} \ln \sum_i (e^{x_i} + e^{-x_i})$ and define

$$\mu(f) \stackrel{\text{def}}{=} \mathrm{lmax}(2\alpha\mathbf{R}(b - \mathbf{B}f))$$
$$\phi(f) \stackrel{\text{def}}{=} \left\|\mathbf{C}^{-1}f\right\|_\infty + \mu(f)$$

Lemma 6.5.5 *After* ALMOST-ROUTE-DIRECTED(b, ϵ, f_0) *terminates, we have*

$$\phi(f) \leq (1+\epsilon)\frac{b^\top v}{\|\mathbf{C}\max(\mathbf{B}^\top v, 0)\|_1},$$

assuming $\epsilon \leq 1/2$.

Proof

We have

$$\bigtriangledown\mu(f) = -2\alpha\mathbf{B}^\top v.$$

```
$(f, v) = $ ALMOST-ROUTE-DIRECTED$(b, \epsilon, f_0)$
  1. Initialize $f := f_0, \delta := \frac{\epsilon}{10\alpha^2}$.
  2. Scale $f$ and $b$ so that $\left\|\mathbf{C}^{-1} f\right\|_\infty + 2\alpha\left\|\mathbf{R}(b - \mathbf{B}f)\right\|_\infty = 20\epsilon^{-1} \ln n$.
  3. Repeat while any of the following conditions is satisfied:
      (a) if $\phi(f) < 16\epsilon^{-1} \ln n$, scale $f$ and $b$ up by $17/16$ and restart step 3.
      (b) let $s$ be $w(e)$ on the coordinates $e$ where $\nabla\mu(f)$ is negative and $0$ elsewhere. If
          $-\nabla\mu(f)^\top s > 1 + \frac{\epsilon}{4}$, set $f := f + \delta s$ and restart step 3.
      (c) if $\left\|\mathbf{C}^{-1} f\right\|_\infty + \nabla\mu(f)^\top f > \frac{\epsilon}{4}$, set $f := f - \delta f$ and restart step 3.
  4. Set $x := 2\alpha\mathbf{R}(b - \mathbf{B}f)$.
  5. Set $p := \nabla \mathrm{lmax}(x)$.
  6. Set $v := \mathbf{R}^\top p$.
```

Figure 6.1: The algorithm for computing the maximum flow and minimum congested cut.

Therefore

$$2\alpha \cdot \|\mathbf{C} \max(\mathbf{B}^\top v, 0)\|_1 \leq 1 + \frac{\epsilon}{4}.$$

It also holds that

$$\begin{aligned} \left\|\mathbf{C}^{-1} f\right\|_\infty + 2\alpha v^\top (b - \mathbf{B}f) &= \left\|\mathbf{C}^{-1} f\right\|_\infty + p^T x \\ &\geq \phi(f) - 4 \ln n \\ &\geq \left(1 - \frac{\epsilon}{4}\right) \phi(f). \end{aligned}$$

Simultaneously, we have

$$\begin{aligned} \frac{\epsilon}{4}\phi(f) \geq \frac{\epsilon}{4} &\geq \left\|\mathbf{C}^{-1} f\right\|_\infty + \nabla\mu(f)^T f \\ &= \left\|\mathbf{C}^{-1} f\right\|_\infty - 2\alpha f^T \mathbf{B}^\top v. \end{aligned}$$

Hence

$$2\alpha v^T b \geq \left(1 - \frac{\epsilon}{2}\right) \phi(f),$$

and so

$$\frac{b^T v}{\|\mathbf{C}\max(\mathbf{B}^\top v, 0)\|_1} \geq \frac{\phi(f)}{1+\epsilon}.$$

■

Lemma 6.5.6 ALMOST-ROUTE-DIRECTED(b, ϵ, f_0) *terminates within* $\tilde{\mathcal{O}}(\log(1+\epsilon_0)\alpha^2/\epsilon^3)$ *iterations, where* $\epsilon_0 = \max(\phi(f_0)/OPT_b - 1, \epsilon)$*, assuming* $\epsilon \leq 1/2$.

Proof Let us call the iterations between each scaling in step 2a) a *phase*. Since the initial scaling gives us the correct scale to within factor $1 + \epsilon_0$, we will scale at most $\mathcal{O}(\log(1 + \epsilon_0))$ times. Moreover, if $\epsilon_0 < 1/10$, step 2a) will never be executed.

If step 2b) is about to be executed, then

$$\begin{aligned}\phi(f + \delta s) &\leq \phi(f) + \delta + \delta \bigtriangledown \mu(f)^\top s + 2\alpha^2\delta^2 \\ &\leq \phi(f) - \frac{\epsilon\delta}{4} + 2\alpha^2\delta^2.\end{aligned}$$

If step 2c) is about to be executed, then

$$\begin{aligned}\phi(f - \delta f) &\leq \phi(f) - \delta\left\|\mathbf{C}^{-1} f\right\|_\infty - \delta \bigtriangledown \mu(f)^\top f + 2\alpha^2\delta^2 \\ &\leq \phi(f) - \frac{\epsilon\delta}{4} + 2\alpha^2\delta^2.\end{aligned}$$

In both cases we have

$$\begin{aligned}\frac{\epsilon\delta}{4} - 2\alpha^2\delta^2 &\geq \frac{\epsilon^2}{40\alpha^2} - \frac{\epsilon^2}{50\alpha^2} \\ &= \frac{\epsilon^2}{200\alpha^2}.\end{aligned}$$

Hence each iteration of steps 2b) and 2c) decreases $\phi(f)$ by at least $\Omega(\epsilon^2\alpha^{-2})$.

For $\epsilon_0 \geq 1/10$, every scaling in step 2a) increases $\phi(f)$ by at most $\epsilon^{-1}\ln n$. Hence, for such ϵ_0 there can be at most $\tilde{\mathcal{O}}(\log(1+\epsilon_0)\alpha^2\epsilon^{-3})$ iterations in total.

For $\epsilon_0 < 1/10$, step 2a) will never be executed. Moreover, the $\phi(f)$ after the initial scaling must be at most $OPT_b + \tilde{\mathcal{O}}(\epsilon_0\epsilon^{-1})$. Hence steps 2b) and 2c) can be executed at most $\tilde{\mathcal{O}}(\epsilon_0\alpha^2\epsilon^{-3}) = \tilde{\mathcal{O}}(\log(1+\epsilon_0)\alpha^2\epsilon^{-3})$ times.

■

Note that Lemma 6.5.5 implies that v is a potential vector for a $(1 + \epsilon)$-approximate minimum congested cut. In order to recover the corresponding flow, we can employ the recursion described in [She13]. The only additional component necessary for directed graphs is an $\mathcal{O}(\text{poly}(n, \alpha))$-competitive oblivious routing. Since by Fact 6.5.2 it must be that $\alpha \geq \text{bal}(G)$, this can be obtained easily by taking the maximum spanning in- and out-arborescences from any fixed vertex.

If we run ALMOST-ROUTE-DIRECTED with $f_0 = \vec{0}$, we can find $(1+\epsilon)$-approximate solutions in time $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^3)$. In order to improve the dependency on ϵ, we can employ a general form of composite function minimization, introduced in Section 6.5.4. Define

$$\psi(f) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if for some e, } f_e/w(e) \notin [0, 50 \ln n/\epsilon] \\ \left\|\mathbf{C}^{-1} f\right\|_\infty & \text{otherwise.} \end{cases}$$

The faster algorithm is presented in Figure 6.2.

```
(f, v) = FAST-ALMOST-ROUTE(b, ε)

  1. Set $f_0$ using ALMOST-ROUTE-DIRECTED $\left(b, \frac{1}{2}, \vec{0}\right)$, keeping the rescaling.

  2. Set $K := \lceil \alpha^2/\epsilon^2 \rceil$.

  3. For $k = 0, \dots, K-1$ let

        $f_{k+1} := \text{argmin}_{f \in \mathbb{R}^E} \left( \nabla \mu(f_k)^\top f + \frac{\alpha^2}{2} \left\|\mathbf{C}^{-1}(f - f_k)\right\|_\infty^2 + \psi(f) \right).$

  4. Return ALMOST-ROUTE-DIRECTED$(b, \epsilon, f_K)$.
```

Figure 6.2: Faster algorithm for computing the maximum flow and minimum congested cut.

If we apply the analysis from Section 6.5.4 (encapsulated in Theorem 6.5.9), we obtain the following.

Lemma 6.5.7 FAST-ALMOST-ROUTE(b, ϵ) *terminates in* $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^2)$ *time, assuming* $\epsilon \leq 1/2$.

Proof As $\phi(\vec{0}) = \tilde{\mathcal{O}}(OPT_b \alpha)$, step 1. works in $\tilde{\mathcal{O}}(m\alpha^2)$ time by Lemma 6.5.6.

Now note that we can apply Theorem 6.5.9 to $\psi'(x) + g(x)$ with $\psi'(x) = \psi(\mathbf{C}x), g(x) = \mu(\mathbf{C}x), L = \alpha^2, D = 50\epsilon^{-1} \ln n$. This yields that $\phi(f_K) \leq (1 + \tilde{\mathcal{O}}(\epsilon)) OPT_b$. Hence by Lemma 6.5.6, step 4. runs in $\tilde{\mathcal{O}}(m\alpha^2\epsilon^{-2})$ time.

The only remaining thing to show is that we can solve the optimization problem in step 3. in $\tilde{\mathcal{O}}(m)$ time. It can be reformulated by introducing an auxiliary variable z:

$$\begin{aligned} \underset{f,z}{\text{minimize}} \quad & \nabla \mu(f_k)^\top f + \frac{1}{2}\alpha^2 z^2 + \psi(f) \\ \text{subject to} \quad & \left\|\mathbf{C}^{-1}(f - f_k)\right\|_\infty \leq z. \end{aligned}$$

For a fixed z, the problem can easily be solved in $\mathcal{O}(m \log m)$ time by sorting. Hence we can employ ternary search over z to achieve $\tilde{\mathcal{O}}(m)$ runtime.

■

6.5.2 Computing Imbalance

As verifying balance can be reduced to a maximum flow computation by Theorem 6.2.3, we obtain the following result:

Lemma 6.2.9 *There is an algorithm that either certifies that* $\mathrm{bal}(G) \leq \alpha$ *or shows that* $\mathrm{bal}(G) > (1-\epsilon)\alpha$ *in time* $\tilde{\mathcal{O}}(m\alpha^2/\epsilon^2)$.

Proof Construct G' by adding the reverse of G multiplied by $\frac{1}{4\alpha}$ to G. Note that $\mathrm{bal}(G') \leq 4\alpha$. Let b' be the residual degrees in G'. Now by Theorem 6.2.8 we can compute a 2-overestimate c' to the minimum congestion to route $-b'$ in G', in time $\tilde{\mathcal{O}}(m\alpha^2)$. Note that we have

$$\mathrm{bal}(G') \leq c' - 1 \leq 2\mathrm{bal}(G') \leq 2\mathrm{bal}(G).$$

Hence if $c' - 1 > 2\alpha$ we can conclude that $\mathrm{bal}(G) > \alpha$ and return the corresponding cut.

Otherwise, we must have $\mathrm{bal}(G) \leq 2\alpha$. Hence we can compute a $(1+\epsilon)$-overestimate c to the minimum congestion to route $-b$ in G, in time $\tilde{\mathcal{O}}(m\alpha^2\epsilon^{-2})$, where b are the residual degrees in G. Now we have

$$\mathrm{bal}(G) \leq c - 1 \leq (1+\epsilon)\mathrm{bal}(G),$$

and so if $c - 1 \leq \alpha$ then $\mathrm{bal}(G) \leq \alpha$, and otherwise we can return a cut proving that $\mathrm{bal}(G) > (1-\epsilon)\alpha$.

■

6.5.3 Application to Directed Sparsest Cut

In this subsection, assume $G = (V, E)$ is a directed graph that is unweighted, strongly connected, simple, and with an even number of vertices. We define the sparsity of a cut $(S, V \setminus S)$ as $\frac{w(S, V\setminus S)}{|S|\cdot|V\setminus S|}$, where $w(S, V \setminus S)$ is the number of edges going from S to $V \setminus S$. Note that under this definition, no cut can have sparsity greater than one.

As a second application of our maximum flow algorithm, we get the following sparsest cut algorithm. While blocking flows could also possibly be used for our purpose, our approach is clean and may easily generalize to weighted graphs.

Lemma 6.5.8 *Given* $\phi \leq 1$, *we can find a cut of sparsity* ϕ *in* G *or determine that all cuts in* G *have sparsity* $\Omega(\phi/\log^2 n)$ *in time* $\tilde{\mathcal{O}}(m/\phi^2)$.

Proof First, we can use Lemma 6.2.9 to check whether $\mathrm{bal}(G) \leq \phi^{-1}$. If it is not, we can return the smaller direction of the imbalanced cut as the result. Otherwise, we use can apply the cut-matching game algorithm given by Louis [Lou10] for $\phi' = \frac{n\phi}{4}$ [3] and reduce the problem to a sequence of $\tilde{O}(1)$ maximum flow queries. Each of the queries fixes some $S \subseteq V$ with $|S| = n/2$ and asks for a flow in G with demands $-\phi'$ on S and ϕ' on $V \setminus S$. We can compute the 2-approximate minimum congestion flow for such a query. If the returned congestion is at most 1, we return the flow. Otherwise, we have a set $T \subseteq V$ which achieves

$$\begin{aligned} b_T / w(T, V \setminus T) &\geq \frac{1}{2}, \\ w(T, V - T) &\leq 2b_T \\ &\leq 2\phi' \min(|T|, |V \setminus T|) \\ &\leq \frac{n}{2}\phi \min(|T|, |V \setminus T|) \\ &\leq \phi |T| \cdot |V \setminus T|. \end{aligned}$$

■

6.5.4 Composite Function Minimization

In this section, we provide a non-Euclidean gradient descent method for minimizing a composite function $f(x) \overset{\mathrm{def}}{=} g(x) + \psi(x)$, where g and ψ have specific properties. The algorithm and its convergence guarantee are encapsulated in the following theorem, and they build on several works in convex optimization, such as [Nes13, RT14].

Theorem 6.5.9 *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function given by $f(x) \overset{\mathrm{def}}{=} g(x) + \psi(x)$ where g is convex and L-smooth*[4] *with respect to some norm $\|\cdot\|$. Moreover, assume that $f(x)$ is only finite on some region of diameter D in $\|\cdot\|$. Starting with some $x_0 \in \mathbb{R}^n$ for all k let*

$$x_{k+1} := \mathrm{argmin}_{x \in \mathbb{R}^n} \Big(\big\langle \nabla g(x_k), x \big\rangle + \frac{L}{2} \big\|x - x_k\big\|^2 + \psi(x) \Big) .$$

Then for all $k \geq 1$ we have

$$\epsilon_k \leq \max \left\{ \frac{2 \cdot L \cdot D^2}{\lfloor \frac{k-1}{2} \rfloor + 4}, \left(\frac{1}{2}\right)^{\lfloor \frac{k-1}{2} \rfloor} \epsilon_0 \right\}$$

where $\epsilon_k = f(x_k) - \min_x f(x)$.

Note that the norm we use is arbitrary and we get a gradient descent analysis without appealing to the dual norm. Also we do not require convex ψ we only require convex f.

[3]The rescaling by n is used due to a slightly different definition of sparsity in [Lou10].

[4]A function is L-smooth with respect to the norm $\|\cdot\|$ if, for all $\vec{x}$ and $\vec{y}$, $\|\nabla f(\vec{x}) - \nabla f(\vec{y})\| \leq L \|\vec{x} - \vec{y}\|$.

We break the proof into 2 parts, first we prove a lemma about the progress of each gradient step and then we use this to prove the theorem. Let X_* be the set of all optimal solutions to $\min_x f(x)$.

Lemma 6.5.10 (Gradient Descent Progress) *For all $k \geq 0$ we have that for all $x_* \in X_*$*

$$f(x_{k+1}) \leq f(x_k) - \min\left\{ \frac{1}{2L}\left(\frac{\epsilon_k}{\|x_k - x_*\|}\right)^2, \frac{f(x_k) - f(x_*)}{2} \right\}$$

Proof By the smoothness of g we know that for all $x \in \mathbb{R}^n$ we have

$$f(x) \leq g(x_k) + \langle \nabla g(x_k), x - x_k \rangle + \frac{L}{2}\|x - x_k\|^2 + \psi(x) \,.$$

By definition of x_{k+1} we then have that

$$f(x_{k+1}) \leq \min_x \Big(g(x_k) + \langle \nabla g(x_k), x - x_k \rangle + \frac{L}{2}\|x - x_k\|^2 + \psi(x) \Big) \,.$$

Now it follows from the convexity of g that

$$g(x) \geq g(x_k) + \langle \nabla g(x_k), x - x_k \rangle,$$

and combining these yields that

$$f(x_{k+1}) \leq \min_{x \in \mathbb{R}^n} \Big(f(x) + \frac{L}{2}\|x - x_k\|^2 \Big) \tag{6.1}$$

Since f is convex, for all $\alpha \in [0,1]$ and $x_* \in X_*$ we have

$$f(\alpha x_* + (1-\alpha)x_k) \leq \alpha f(x_*) + (1-\alpha) f(x_k) = f(x_k) - \alpha(f(x_k) - f(x_*)).$$

Consequently

$$\min_{x \in \mathbb{R}^n} \Big(f(x) + \frac{L}{2}\|x - x_k\| \Big) \leq \min_{\alpha \in [0,1]} \Big(f(x_k) - \alpha(f(x_k) - f(x_*)) + \frac{L\alpha^2}{2}\|x_k - x_*\|^2 \Big).$$

By taking the derivative with respect to α of the expression on the right hand side above and setting it to zero, we see that the optimal α satisfies

$$-(f(x_k) - f(x_*)) + \alpha L \|x_k - x_*\|^2 = 0$$

and thus using $\alpha = \min\left\{\frac{f(x_k)-f(x_*)}{L\left\|y-x_*\right\|^2}, 1\right\}$ yields the result, since $f(x_k) - f(x_*) \geq 0$, and when $f(x_k) - f(x_*) \geq L\left\|x_k - x_*\right\|^2$, we have

$$\min_{x\in\mathbb{R}^n}\left(f(x) + \frac{L}{2}\left\|x_k - x\right\|^2\right) \leq f(x_*) + \frac{L}{2}\left\|x_k - x_*\right\| \leq f(x_*) + \frac{f(x_k)-f(x_*)}{2}\,.$$

■

Using the lemma, we can complete the proof of the theorem as follows.

Proof of Theorem 6.5.9: By Lemma 6.5.10 we have that $\epsilon_{k+1} \leq \epsilon_k$ for all k and

$$\epsilon_{k+1} \leq \max\left\{\epsilon_k - \frac{1}{2L}\left(\frac{\epsilon_k}{D}\right)^2, \frac{\epsilon_k}{2}\right\}$$

Consequently for $k \geq 1$ such that $\epsilon_k - \frac{1}{2L}\left(\frac{\epsilon_k}{D}\right)^2 \geq \frac{\epsilon_k}{2}$ we have

$$\frac{1}{\epsilon_k} - \frac{1}{\epsilon_{k+1}} \leq \frac{\epsilon_{k+1}-\epsilon_k}{\epsilon_k\epsilon_{k+1}} \leq -\frac{1}{2L}\cdot\frac{1}{D^2}\cdot\frac{\epsilon_k}{\epsilon_{k+1}} \leq -\frac{1}{2LD^2}$$

Summing yields that

$$\frac{1}{\epsilon_1} - \frac{1}{\epsilon_k} \leq -\frac{N_k}{2LD^2}$$

where N_k is the number of steps $k \geq 1$ for which $\epsilon_k - \frac{1}{2L}\left(\frac{\epsilon_k}{D}\right)^2 \geq \frac{\epsilon_k}{2}$. Furthermore, clearly by 6.5.10 we have that

$$\epsilon_1 \leq \frac{L}{2}D^2$$

and thus

$$\epsilon_k \leq \frac{2LD^2}{N_k + 4}$$

On the other hand we have that

$$\epsilon_{k+1} \leq \left(\frac{1}{2}\right)^{k-1-N_k}$$

and noting that either $N_k \geq \lfloor\frac{k-1}{2}\rfloor$ or $k - 1 - N_k \geq \lfloor\frac{k-1}{2}\rfloor$ then yields the result.

■

Appendix

6.A Missing proofs from Section 6.2

Lemma 6.A.1 *Let G be a strongly connected directed graph. If demand vector d can be routed in G with congestion c, then $-d$ can be routed in G with congestion at most $\mathrm{bal}(G) \cdot c$.*

Proof

Note that for any $v \in \mathbb{R}^n$

$$\|\mathbf{U}\max(\mathbf{B}v, 0)\|_1 \leq \mathrm{bal}(G)\|\mathbf{U}\max(-\mathbf{B}v, 0)\|_1$$

follows from the definition of balance.

Hence it is easily seen that the optimum value for the dual problem is within a factor $\mathrm{bal}(G)$ for demands d and $-d$. Our theorem now follows from strong duality to the original problem.

■

Lemma 6.A.2 *Let $l, r \in \mathbb{R}$ with $l \leq r$. Let $C \subseteq \mathbb{R}^m$ be a convex set such that for any $S \subseteq \{1, 2, \ldots, m\}$ there exists a point $x \in C$ such that x_i is at least l for $i \in S$ and x_i is at most r for $i \notin S$. Then there exists a point in C with all coordinates in $[l, r]$.*

Proof Let $P_i(S)$, for $i \in \{0, \ldots, m\}, S \subseteq \{1, \ldots, m\}$ be the subset of points $x \in C$ that satisfy

- $x_j \in [l, r]$ for $j \leq i$ and
- $x_j \geq l$ for $j \in S$ and
- $x_j \leq r$ for $j \notin S$.

We prove that $P_i(S)$ is nonempty for every i and S by induction on i. The base case $i = 0$ follows from the assumption on C. Assume $i \in \{0, \ldots, m-1\}$ and the thesis holds for i. Let S be any subset of $\{1, \ldots, m\}$. Let $S_L := S \cup \{i+1\}, S_R = S \setminus \{i+1\}$. Pick any $x_L \in P_i(S_L)$ and $x_R \in P_i(S_R)$. Then a convex combination of x_L and x_R must belong to $P_{i+1}(S)$. Since S was arbitrary, this concludes the proof.

■

Proof of Theorem 6.2.3: The implication $(2. \to 1.)$ and the equivalence $(2. \leftrightarrow 3.)$ are easy to check (note that the circulation of 2. is the sum of $\vec{1}$ and a routing of $-d$.). We now prove that if $\mathrm{bal}(G) \le \alpha$ there exists a circulation in G with each congestion in $[1, \alpha]$.

Note that for any subset S of edges of G we can route the residual degree d_S induced by these edges with congestion 1. Hence by Lemma 6.A.1 we can route $-d_S$ with congestion at most α. Adding these flows yields a circulation with congestion in $[1, \alpha+1]$ on edges in S and in $[0, \alpha]$ on the other edges. Since the choice of S was arbitrary, the thesis follows by Lemma 6.A.2.

■

We now prove the following lemma, implying Fact 6.2.2.

Lemma 6.A.3 *Let $G = (V, E, w)$ be an undirected graph and $s, t \in V$. Let d be a demand vector that can be routed in G with congestion at most 1. Let f be a flow from s to t in G with congestion not exceeding 1 satisfying demands $(1-\epsilon)d$. Let H be the residual graph of f in G. Then $\mathrm{bal}(H) \le (2\epsilon^{-1} - 1)$.*

Proof We use the third equivalent definition of balance from Theorem 6.2.3. The residual degrees in H are $2(\epsilon - 1)d$. Since there exists a flow satisfying demands ϵd with congestion 1, $2(1-\epsilon)d$ can be routed in H with congestion $2(1-\epsilon)\epsilon^{-1} = 2\epsilon^{-1} - 2$.

■

6.B Analysis of the clustering algorithm

Before we prove Lemma 6.3.1, we shall study the properties of a class of two-way infinite sequences.

For better understanding, we now attempt to provide the intuition on how the sequences defined below are used in the proof. For simplicity, assume we are trying to analyze the clustering of a *path* rather than a cycle. Imagine that every vertex v in the graph sends a runner of unit speed to every vertex of the path, starting at time $-x_v$. After reaching the path, the runner keeps running on it until they reach the end. We will call a runner a *local leader* if they were the first one to reach the end of the path out of all the runners that entered the path at a no later position. It is easy to see that the sequence of origins of local leaders in the order they reach the end of the path is the same as the sequence of roots of clusters into which the path is partitioned by the algorithm. Therefore, it is enough to observe the local leaders as they reach the end of the path. It can be shown that in any time interval $[y, y+\epsilon]$ the probability of the origin of the last local leader to reach the end of the path changing is $\mathcal{O}(\beta\epsilon)$. Unfortunately, the entire process could take an arbitrary amount of time in the case of a path.

To apply the above reasoning to a cycle, we will 'unroll' it into a two-way infinite path. We will set the 'finish line' at an arbitrary vertex (at position 0) and observe the local leaders for any period of time $[y, y+L]$.

Assume the length of the cycle is L and it has l vertices. Let $i \in \{0, \dots, l-1\}, i \in \mathbb{Z}$. Then, the $i \cdot n + j$-th element of the sequence s will intuitively be equal to the time the runner sent from the j-th vertex of the graph to the i-th vertex of the unrolled cycle reaches vertex 0 of the unrolled cycle. The sequence a will simply label the origin of the runner relevant to the current index (and so $a_i = (i \bmod n)$). The sequence c will label the cluster to which the relevant vertex of the cycle is assigned (the origin of the first runner to reach it). The function $f(y)$ will give the origin of the runner that reached vertex 0 before time y and entered the cycle at the earliest position. Since only such runners will correspond to clusters, our goal will be to bound the frequency with which f may change.

6.B.1 Periodically Decreasing Sequences

Let $k, n \in \mathbb{N}$ and $L \in \mathbb{R}_+$.

Let s_i be a two-way infinite sequence of real numbers indexed by $i \in \mathbb{Z}$ with the property

$$\forall_i \; s_{i+k} = s_i - L.$$

Let a_i be a two-way infinite sequence of integers in $\{0, \ldots, n-1\}$ indexed by $i \in \mathbb{Z}$, periodic with period k, that is

$$\forall_i \; a_{i+k} = a_i.$$

We construct the sequence c_i by defining

$$c_i = a_j,$$

where j is the minimum q that minimizes the value of s_q among $q \leq i$.

We can similarly construct $f : \mathbb{R} \to \{0, \ldots, n-1\}$ by setting for every real number y

$$f(y) = a_j,$$

where j is the minimum q that satisfies $s_q \leq y$.

Fact 6.B.1 *The sequence c_i is periodic with period k.*

Fact 6.B.2 *The function f is periodic with period L.*

Fact 6.B.3 *For any $i \in \mathbb{Z}$ and $y \in \mathbb{R}$, the number of times the sequence $c_i, c_{i+1}, \ldots, c_{i+k}$ changes values is equal to the number of times f changes values on the interval $[y, y+L]$.*

6.B.2 Random Periodically Decreasing Sequences

Let $k, n \in \mathbb{N}, L \in \mathbb{R}_+$.

Let t_i be a two-way infinite sequence of real numbers indexed by $i \in \mathbb{Z}$ with the property

$$\forall_i \; t_{i+k} = t_i - L.$$

Let a_i be a two-way infinite sequence of integers in $\{0, \ldots, n-1\}$ indexed by $i \in \mathbb{Z}$, periodic with period k, that is

$$\forall_i \; a_{i+k} = a_i.$$

Let $x_0, x_1, \ldots, x_{n-1}$ be independent random variables drawn from the exponential distribution with parameter β.

We define for every $i \in Z$:

$$s_i = t_i - x_{a_i}$$

We define the function $f : \mathbb{R} \to \{0, \ldots, n-1\}$ as in the previous section, that is

$$f(y) = a_j,$$

where j is the minimum q that satisfies $s_q \leq y$.

In the following lemmas, our goal will be to bound the expected number of times the value of f changes on any interval.

Lemma 6.B.4 *For any $y \in \mathbb{R}, \epsilon \in \mathbb{R}_+$, the probability that f is not constant on the interval $[y, y+\epsilon]$ is bounded by $\mathcal{O}(\beta\epsilon)$.*

Proof Fix y and ϵ. We condition on the value of $f(y+\epsilon)$; assume it is k. We also condition on x_i for all $i \neq k$. Now the condition $f(y+\epsilon) = k$ is equivalent to assuming $x_k \geq c$ for some constant c. Because we have no more information about x_k, the conditional probability that $x_k \geq c + \epsilon$ is $1 - \mathcal{O}(\beta\epsilon)$. This implies the thesis.

■

In order to exploit Lemma 6.B.4 to bound the expected number of changes in f we will attempt to condition on the event D_ϵ.

Definition 6.B.5 *Let $\epsilon \in \mathbb{R}_+$. The event D_ϵ occurs iff for all pairs $i, j \in \mathbb{Z}$ such that $a_i \neq a_j$ or $t_i \neq t_j$ it holds that*

$$|s_i - s_j| > \epsilon.$$

Fact 6.B.6

$$\lim_{\epsilon \to 0} P(D_\epsilon) = 1.$$

Using Fact 6.B.6, we pick an $\epsilon > 0$ that satisfies

$$P(D_\epsilon) \geq 1 - \min\left(\frac{1}{2}, \frac{\beta L}{k}\right) \quad \text{and} \quad \frac{L}{\epsilon} \in \mathbb{N}\,.$$

Lemma 6.B.7 *Assume ϵ is chosen as above. Conditioning on D_ϵ, for any $y \in \mathbb{R}$, the probability that f is not constant on the interval $[y, y+\epsilon]$ is bounded by $\mathcal{O}(\beta\epsilon)$.*

Proof Because $P(D_\epsilon) \geq \frac{1}{2}$, the conditional probability is at most two times larger than in the case where we do not condition on D_ϵ. The thesis follows from Lemma 6.B.4.

■

Lemma 6.B.8 *Assume ϵ is chosen as above. Conditioning on D_ϵ, for any $y \in \mathbb{R}$, the expected number of times f changes values in $[y, y+L]$ is bounded by $\mathcal{O}(\beta L)$.*

Proof Because we assume D_ϵ, we know that f can change at most once on any interval of length ϵ. Hence it follows from Lemma 6.B.7 that the expected number of time f changes on any interval of length ϵ is bounded by $\mathcal{O}(\beta\epsilon)$. Because $L/\epsilon \in \mathbb{N}$, we can cover the interval $[y, y+L]$ with L/ϵ intervals of length ϵ. Because of linearity of expectation, the expected number of times f changes values on $[y, y+L]$ is therefore bounded by $\mathcal{O}(\beta L)$.

■

Lemma 6.B.9 *For any $y \in \mathbb{R}$, the expected number of times f changes values in $[y, y+L]$ is bounded by $\mathcal{O}(\beta L)$.*

Proof It follows from Fact 6.B.3 that f cannot change values more than k times on an interval of length L. For ϵ chosen as above, we can apply this observation together with Lemma 6.B.8 to see that the expected number of changes is bounded by

$$P(D_\epsilon)\mathcal{O}(\beta L) + (1 - P(D_\epsilon))k = \mathcal{O}(\beta L) + \frac{\beta L}{k} \cdot k = \mathcal{O}(\beta L)$$

■

6.B.3 Low-radius Decompositions

Recall that we are considering the clustering algorithm CLUSTER-DIRECTED applied to a directed graph $G = (V, E)$. Consider a cycle C in G. Assume the length of C is L and the number of vertices on C is l. Let the vertices on the cycle be $u_0, \ldots, u_{l-1}$, in order, with u_0 chosen arbitrarily.

For $i \in \{0, \ldots, l-1\}$, define p_i to be the distance from u_0 to u_i when going along the cycle.

Let $k = l \cdot n$. We now define the two-way infinite sequence t as follows, for $z \in \mathbb{Z}, i \in \{0, \ldots, m-1\}, j \in \{0, \ldots, n-1\}$:

$$t_{z \cdot k + i \cdot n + j} = d(v_j, u_i) - zL - p_i.$$

We define the two-way infinite sequence a for $z \in \mathbb{Z}, j \in \{0, \ldots, n-1\}$:

$$a_{z \cdot n + j} = j.$$

Fact 6.B.10 *Let $i \in \{0, \ldots, l-1\}$. Assume j is a (possibly negative) integer such that $j \leq i \cdot n + n - 1$. Then there exists a path from v_{a_j} to u_i of length $t_j + p_i$.*

Fact 6.B.11 *Let $i \in \{0, \ldots, l-1\}, q \in \{0, \ldots, n-1\}$. There exists an integer $j \leq i \cdot n + n - 1$ such that $a_j = q$ and*

$$t_j + p_i = d(v_q, u_i).$$

Recall that in CLUSTER-DIRECTED we associate with each vertex v_i an independent random variable x_i drawn from the exponential distribution with parameter β. We now define the two-way infinite sequence s as

$$s_i = t_i - x_{a_i}.$$

As in Section 6.B.1 we construct the sequence c_i by defining

$$c_i = a_j,$$

where j is the minimum q that minimizes the value of s_q among $q \leq i$.

Lemma 6.B.12 *For $i \in \{0, \ldots, l-1\}$, $c_{i \cdot n + n - 1}$ is the index of the vertex to whose cluster u_i is assigned by* CLUSTER-DIRECTED.

Proof This follows from Facts 6.B.10 and 6.B.11.

■

We are now ready to prove the main theorem.

Proof of Lemma 6.3.1: By Lemma 6.B.12, it is enough to bound the number of times $c_0, \ldots, c_k$ changes values. By Fact 6.B.3 this reduces to bounding the number of times the associated function $f : \mathbb{R} \to \{0, \ldots, n-1\}$ changes values on any interval of length L. This is shown to be $\mathcal{O}(\beta L)$ in expectation in Lemma 6.B.9.

■

Proof of Theorem 6.2.5: First note that with high probability $\max(x_1, \ldots, x_n) \leq r$, and so the radius of the computed clusters is at most r. To bound the number of cut edges, it is enough to note that by Theorem 6.2.3, we can find a set of simple cycles $C_1, \ldots, C_k$ such that their total volume is at most $\mathrm{bal}(G)\mathrm{vol}(G)$ and has weight at least $w(e)$ on every edge $e \in E$. The thesis follows by applying Lemma 6.3.1.

■

Bibliography

[ABN08] Ittai Abraham, Yair Bartal, and Ofer Neiman, *Nearly tight low stretch spanning trees*, Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '08, IEEE Computer Society, 2008, pp. 781–790. 4.1, 4.1.1, 5.4

[ACF$^+$03] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke, *Optimal oblivious routing in polynomial time*, Proc. of ACM STOC (Lawrence L. Larmore and Michel X. Goemans, eds.), ACM, 2003, pp. 383–388. 6.1

[AKPW95] N. Alon, R. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput. **24** (1995), no. 1, 78–100. 4.0.1, 4.1, 4.1.1, 4.5, 4.5.1, 5.4, 6.2.3

[AM14] Ahmed El Alaoui and Michael W Mahoney, *Fast randomized kernel methods with statistical guarantees*, arXiv preprint arXiv:1411.0306 (2014). 3.1.3, 3.2

[AN12] Ittai Abraham and Ofer Neiman, *Using petal decompositions to build a low stretch spanning tree*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 395–406. 4, 4.1, 4.1, 4.1.1, 4.3, 5.4

[Awe85] Baruch Awerbuch, *Complexity of network synchronization*, J. ACM **32** (1985), no. 4, 804–823. 6.2.3

[Axe94] Owe Axelsson, *Iterative solution methods*, Cambridge University Press, New York, NY, 1994. 5.1, 5.2

[Baj88] Chanderjit Bajaj, *The algebraic degree of geometric optimization problems*, Discrete & Computational Geometry **3** (1988), no. 2, 177–191 (English). 2.1.1

[Bar96] Y. Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, 1996, pp. 184–193. 4.0.1, 4.1, 4.1.1, 4.3, 4.3, 4.3, 6.2.3, 6.3.2

[Bar98] Yair Bartal, *On approximating arbitrary metrices by tree metrics*, Proceedings of the thirtieth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '98, ACM, 1998, pp. 161–168. 4.1.1

[BB05] Antoine Bordes and Léon Bottou, *The huller: a simple and efficient online svm*, Machine Learning: ECML 2005, Springer, 2005, pp. 505–512. 3.1.2

[BGK$^+$13] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan, *Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs*, Theory of Computing Systems (2013), 1–34 (English). 4.1.2, 4.5.3

[BGKL15] Christos Boutsidis, Dan Garber, Zohar Karnin, and Edo Liberty, *Online principal components analysis*, Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2015, pp. 887–901. 3.1.2

[BHI02] Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk, *Approximate clustering via core-sets*, Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada, 2002, pp. 250–257. 2.1, 2.1.1

[BK96] András A. Benczúr and David R. Karger, *Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time*, Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '96, ACM, 1996, pp. 47–55. 5.1

[BKR03] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke, *A practical algorithm for constructing oblivious routing schemes*, SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003), ACM, 2003, pp. 24–33. 6.1

[BL99] Yair Bartal and Stefano Leonardi, *On-line routing in all-optical networks*, Theoretical Computer Science **221** (1999), no. 1-2, 19–39. 6.1, 6.1.1

[BMM03] Prosenjit Bose, Anil Maheshwari, and Pat Morin, *Fast approximations for sums of distances, clustering and the Fermat-Weber problem*, Computational Geometry **24** (2003), no. 3, 135 – 146. 2.1.1

[Bot04] Léon Bottou, *Stochastic learning*, Advanced Lectures on Machine Learning (Olivier Bousquet and Ulrike von Luxburg, eds.), Lecture Notes in Artificial Intelligence, LNAI 3176, Springer Verlag, Berlin, 2004, pp. 146–168. 5.1

[BSS09] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava, *Twice-ramanujan sparsifiers*, Proceedings of the 41st annual ACM symposium on Theory of computing (New York, NY, USA), STOC '09, ACM, 2009, pp. 255–262. 5.1

[BSS12] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava, *Twice-ramanujan sparsifiers*, SIAM Journal on Computing **41** (2012), no. 6, 1704–1721. 3.1.3, 3.4, 3.5

[BSST13] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng, *Spectral sparsification of graphs: theory and algorithms*, Commun. ACM **56** (2013), no. 8, 87–94. 5.1

[Bub14] Sébastien Bubeck, *Theory of convex optimization for machine learning*, arXiv preprint arXiv:1405.4980 (2014). 2.5.2

[BY82] Egon Balas and Chang-Sung Yu, *A note on the weiszfeld-kuhn algorithm for the general fermat problem*, Managme Sci Res Report (1982), no. 484, 1–6. 2.1.1

[Can06] J. Candés, E, *Compressive sampling*, Proceedings of the International Congress of Mathematicians (2006). 1.1.2

[CCG$^+$98] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin, *Approximating a finite metric by a small number of tree metrics*, Proceedings of the 39th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '98, IEEE Computer Society, 1998, pp. 379–388. 4.1

[CDK$^+$06] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer, *Online passive-aggressive algorithms*, The Journal of Machine Learning Research **7** (2006), 551–585. 3.1.2

[CFM$^+$14] Michael Cohen, Brittany Terese Fasy, Gary L. Miller, Amir Nayyeri, Richard Peng, and Noel Walkington, *Solving* 1*-laplacians of convex simplicial complexes in nearly linear time: Collapsing and expanding a topological ball*, 2014. 4.1

[CK81] Leon Cooper and I.Norman Katz, *The weber problem revisited*, Computers and Mathematics with Applications **7** (1981), no. 3, 225 – 234. 2.1.1

[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*, Proceedings of the 43rd annual ACM symposium on Theory of computing (New York, NY, USA), STOC '11, ACM, 2011, pp. 273–282. 2, 4.1, 5.B

[CKM$^+$14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup Rao, and Shen Chen Xu, *Solving sdd linear systems in nearly mlog$^{1/2}$n time*, STOC, 2014, pp. 343–352. 1.3.3, 4, 4.0.1, 5, 6.3.2

[CKP$^+$14] Michael B. Cohen, Rasmus Kyng, Jakub W. Pachocki, Richard Peng, and Anup Rao, *Preconditioning in expectation*, CoRR **abs/1401.6236** (2014). 1.3.3, 4.1.2, 5

[CLM$^+$15] Michael B Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford, *Uniform sampling for matrix approximation*, Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ACM, 2015, pp. 181–190. 3.1.1, 3.1.3, 3.2

[CLM$^+$16] Michael Cohen, Yin-Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford, *Geometric median in nearly linear time*, to appear in STOC 2016, 2016. 1.3.1, 2

[CMMP13] Hui Han Chin, Aleksander Madry, Gary L. Miller, and Richard Peng, *Runtime guarantees for regression problems*, Proceedings of the 4th conference on Innovations in Theoretical Computer Science (New York, NY, USA), ITCS '13, ACM, 2013, pp. 269–282. 2.1, 2.1.1, 2

[CMP$^+$14] Michael B. Cohen, Gary L. Miller, Jakub W. Pachocki, Richard Peng, and Shen Chen Xu, *Stretching stretch*, CoRR **abs/1401.2454** (2014). 1.3.3, 4.0.1, 4.0.1, 5.4, 6.3.2

[CMP15] M. Cohen, C. Musco, and J. Pachocki, *Online row sampling*, Unpublished manuscript, 2015. 1.3.2, 3

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson, *Introduction to algorithms*, 2nd ed., McGraw-Hill Higher Education, 2001. 4.4.1

[CT89] R. Chandrasekaran and A. Tamir, *Open questions concerning weiszfeld's algorithm for the fermat-weber location problem*, Mathematical Programming **44** (1989), no. 1-3, 293–295 (English). 2.1.1

[CW13] Kenneth L. Clarkson and David P. Woodruff, *Low rank approximation and regression in input sparsity time*, Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC), 2013, pp. 81–90. 3.1.1

[Dia69] Robert B. Dial, *Algorithm 360: shortest-path forest with topological ordering [h]*, Commun. ACM **12** (1969), no. 11, 632–633. 4.5

[DKSW02] Zvi Drezner, Kathrin Klamroth, Anita SchÃűbel, and George Wesolowsky, *Facility location*, ch. The Weber problem, pp. 1–36, Springer, 2002. 2.1.1

[DS84] Peter G. Doyle and J. Laurie Snell, *Random walks and electric networks*, Carus Mathematical Monographs, vol. 22, Mathematical Association of America, 1984. 4.A

[DS08] Samuel I. Daitch and Daniel A. Spielman, *Faster approximate lossy generalized flow via interior point algorithms*, Proceedings of the 40th annual ACM symposium on Theory of computing (New York, NY, USA), STOC '08, ACM, 2008, pp. 451–460. 4.1

[EEST08] Michael Elkin, Yuval Emek, Daniel A Spielman, and Shang-Hua Teng, *Lower-stretch spanning trees*, SIAM Journal on Computing **38** (2008), no. 2, 608–628. 4.1, 4.1.1, 4.3, 5.4

[EMPS16] Alina Ene, Gary Miller, Jakub Pachocki, and Aaron Sidford, *Routing under balance*, to appear in STOC 2016, 2016. 1.3.4, 6

[ER09] Matthias Englert and Harald Räcke, *Oblivious routing for the lp-norm*, Proc. of IEEE FOCS, IEEE Computer Society, 2009, pp. 32–40. 6.1

[FL11] Dan Feldman and Michael Langberg, *A unified framework for approximating and clustering data*, Proceedings of the forty-third annual ACM symposium on Theory of computing, ACM, 2011, pp. 569–578. 2.1, 2.1.1

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. Syst. Sci. **69** (2004), no. 3, 485–497. 4.1.1

[FT87] Michael L. Fredman and Robert Endre Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM **34** (1987), 596–615. 4.4.1

[Gon92] Clovis C Gonzaga, *Path-following methods for linear programming*, SIAM review **34** (1992), no. 2, 167–224. 2.1.2

[GR98] Andrew V. Goldberg and Satish Rao, *Beyond the flow decomposition barrier*, J. ACM **45** (1998), 783–797. 6.1, 6.1.1

[Gre96] Keith D. Gremban, *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems*, Ph.D. thesis, Carnegie Mellon University, 1996. 4.2, 5.4

[GT83] Harold N. Gabow and Robert Endre Tarjan, *A linear-time algorithm for a special case of disjoint set union*, Proceedings of the 15th annual ACM symposium on Theory of computing (STOC) (New York, NY, USA), ACM, 1983, pp. 246–251. 4.5.5

[Han04] Yijie Han, *Deterministic sorting in o(nlog logn) time and linear space*, J. Algorithms **50** (2004), no. 1, 96–105. 4.5.1

[HKLR05] Mohammad Taghi Hajiaghayi, Jeong Han Kim, Tom Leighton, and Harald Räcke, *Oblivious routing in directed graphs with random demands*, Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005 (Harold N. Gabow and Ronald Fagin, eds.), ACM, 2005, pp. 193–201. 6.1

[HKLR06] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Frank Thomson Leighton, and Harald Räcke, *New lower bounds for oblivious routing in undirected graphs*, Proc. of ACM-SIAM SODA, ACM Press, 2006, pp. 918–927. 6.1

[HKRL07] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Harald Räcke, and Tom Leighton, *Oblivious routing on node-capacitated and directed graphs*, ACM Transactions on Algorithms **3** (2007), no. 4. 6.1, 6.1.1

[HPK05] Sariel Har-Peled and Akash Kushal, *Smaller coresets for k-median and k-means clustering*, Proceedings of the twenty-first annual symposium on Computational geometry, ACM, 2005, pp. 126–134. 2.1.1

[ID00] P. Indyk and Stanford University. Computer Science Dept, *High-dimensional computational geometry*, Stanford University, 2000. 2.1.1

[KL13] Jonathan A Kelner and Alex Levin, *Spectral sparsification in the semi-streaming setting*, Theory of Computing Systems **53** (2013), no. 2, 243–262. 3, 3.1.2, 3.1.3

[KLMS14] Michael Kapralov, Yin Tat Lee, Christopher Musco, and Aaron Sidford, *Single pass spectral sparsification in dynamic streams*, Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on, IEEE, 2014, pp. 561–570. 3, 3.1.2

[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*, Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, 2014, pp. 217–226. 4.1, 6, 6.1, 6.1.1, 6.5.1

[KLP12] Ioannis Koutis, Alex Levin, and Richard Peng, *Improved Spectral Sparsification and Numerical Algorithms for SDD Matrices*, 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012) (Dagstuhl, Germany) (Christoph Dürr and Thomas Wilke, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 14, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 266–277. 4.1.2

[KMP10] Ioannis Koutis, Gary L. Miller, and Richard Peng, *Approaching optimality for solving SDD linear systems*, Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '10, IEEE Computer Society, 2010, pp. 235–244. 1.3.3, 3.2, 5.1, 5.4

[KMP11] ______, *A nearly-m log n time solver for SDD linear systems*, Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '11, IEEE Computer Society, 2011, pp. 590–598. 1.3.3, 4.1, 4.1.2, 4.4.1, 4.5, 5, 5.1, 5.4, 5.4, 5.4

[KMST10] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng, *Subgraph sparsification and nearly optimal ultrasparsifiers*, Proceedings of the 42nd ACM symposium on Theory of computing (New York, NY, USA), STOC '10, ACM, 2010, pp. 57–66. 5.1, 5.2

[KOSZ13] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu, *A simple, combinatorial algorithm for solving SDD systems in nearly-linear time*, Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (New York, NY, USA), STOC '13, ACM, 2013, pp. 911–920. 4.1, 5, 5.1, 5.1, 5.4, 5.B

[Kou07] Ioannis Koutis, *Combinatorial and algebraic tools for optimal multilevel algorithms*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, May 2007, CMU CS Tech Report CMU-CS-07-131. 5.D

[KP79] Richard A. Kronmal and Arthur V. Peterson, *The alias and alias-rejection-mixture methods for generating random variables from probability distributions*, Proceedings of the 11th Conference on Winter Simulation - Volume 1 (Piscataway, NJ, USA), WSC '79, IEEE Press, 1979, pp. 269–280. 2.6

[Kuh73] HaroldW. Kuhn, *A note on fermat's problem*, Mathematical Programming **4** (1973), no. 1, 98–107 (English). 2.1.1

[KV97] Jakob Krarup and Steven Vajda, *On torricelli's geometrical solution to a problem of fermat*, IMA Journal of Management Mathematics **8** (1997), no. 3, 215–224. 2.1.1

[Lei92] F. Thomson Leighton, *Introduction to parallel algorithms and architectures: Array, trees, hypercubes*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992. 4.2

[LMP13] Mu Li, Gary L Miller, and Richard Peng, *Iterative row sampling*, Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on, IEEE, 2013, pp. 127–136. 3.1.1

[LMR94] Frank Thomson Leighton, Bruce M Maggs, and Satish B Rao, *Packet routing and job-shop scheduling in O(congestion+ dilation) steps*, Combinatorica **14** (1994), no. 2, 167–186. 4.2

[Lou10] Anand Louis, *Cut-matching games on directed graphs*, CoRR **abs/1010.1047** (2010). 6.5.3, 3

[LR91] Hendrik P. Lopuhaa and Peter J. Rousseeuw, *Breakdown points of affine equivariant estimators of multivariate location and covariance matrices*, Ann. Statist. **19** (1991), no. 1, 229–248. 2.1, 2.5, 2.5.1

[LRS13] Yin Tat Lee, Satish Rao, and Nikhil Srivastava, *A new approach to computing maximum flows using electrical flows*, Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (New York, NY, USA), STOC '13, ACM, 2013, pp. 755–764. 4.1

[LS13] Yin Tat Lee and Aaron Sidford, *Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems*, Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '13, IEEE Computer Society, 2013, pp. 147–156. 4.1.2, 5.1, 5.1, 5.B

[LS14] Yin Tat Lee and Aaron Sidford, *Path-finding methods for linear programming : Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow*, 55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 18-21 October, 2014, Philadelphia, PA, USA, 2014, pp. 424–433. 1.2.1, 2.1, 2.1.2, 6, 6.1, 6.1.1

[LS15] Yin Tat Lee and He Sun, *Constructing linear-sized spectral sparsification in almost-linear time*, Unpublished manuscript; to appear in FOCS 2015, 2015. 3.4

[LSS14] Edo Liberty, Ram Sriharsha, and Maxim Sviridenko, *An algorithm for online k-means clustering*, arXiv preprint arXiv:1412.5721 (2014). 3.1.2

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong, *A faster cutting plane method and its implications for combinatorial and convex optimization*, Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on, IEEE, 2015. 1.1.2

[Mad10] Aleksander Madry, *Fast approximation algorithms for cut-based problems in undirected graphs.*, FOCS, IEEE Computer Society, 2010, pp. 245–254. 4.1, 4.1.2, 6, 6.5.1

[Mad13] Aleksander Madry, *Navigating central path with electrical flows: From flows to matchings, and back*, 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, 2013, pp. 253–262. 1.2.1, 2.1, 2.1.2, 4.1, 6, 6.1, 6.1.1

[Mah11] Michael W. Mahoney, *Randomized algorithms for matrices and data.*, Foundations and Trends in Machine Learning **3** (2011), no. 2, 123–224. 5.1

[MM13] Michael W. Mahoney and Xiangrui Meng, *Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression*, Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC), 2013, pp. 91–100. 3.1.1

[MMP$^+$05] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo, *Finding effective support-tree preconditioners*, Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures (New York, NY, USA), SPAA '05, ACM, 2005, pp. 176–185. 5.D

[MMVW97] Bruce M. Maggs, Friedhelm Meyer auf der Heide, Berthold Vöcking, and Matthias Westermann, *Exploiting locality for data management in systems of limited bandwidth*, Proc. of IEEE FOCS, IEEE Computer Society, 1997, pp. 284–293. 6.1, 6.1.1

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu, *Parallel graph decompositions using random shifts*, Proceedings of the 25th ACM symposium on Parallelism in algorithms and architectures (New York, NY, USA), SPAA '13, ACM, 2013, pp. 196–203. (document), 4.0.1, 4.4.1, 4.5.3, 6.2.3, 6.3.1, 6.3.1, 6.2

[MR89] Gary L. Miller and John H. Reif, *Parallel tree contraction part 1: Fundamentals*, Randomness and Computation (Silvio Micali, ed.), JAI Press, Greenwich, Connecticut, 1989, Vol. 5, pp. 47–72. 4.5.3

[Nes03] Yu Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, vol. I, 2003. 2.1.2, 2.D.5

[Nes13] Yurii Nesterov, *Gradient methods for minimizing composite functions*, Math. Program. **140** (2013), no. 1, 125–161. 6.5.4

[NN94] Yurii Nesterov and Arkadii Semenovich Nemirovskii, *Interior-point polynomial algorithms in convex programming*, vol. 13, Society for Industrial and Applied Mathematics, 1994. 2.1.2, 2.1.2

[NN13] Jelani Nelson and Huy L. Nguyen, *OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings*, Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2013, pp. 117–126. 3.1.1

[NT13] Deanna Needell and Joel A Tropp, *Paved with good intentions: Analysis of a randomized block kaczmarz method*, Linear Algebra and its Applications (2013). 5.1

[Ost78] Lawrence M. Ostresh, *On the convergence of a class of iterative methods for solving the weber location problem*, Operations Research **26** (1978), no. 4, 597–609. 2.1.1

[OSV12] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi, *Approximating the exponential, the Lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator*, Proceedings of the 44th symposium on Theory of Computing (New York, NY, USA), STOC '12, ACM, 2012, pp. 1141–1160. 4.1

[PE08] Frank Plastria and Mohamed Elosmani, *On the convergence of the weiszfeld algorithm forăcontinuous single facility location allocation problems*, TOP **16** (2008), no. 2, 388–406 (English). 2.1.1

[Pen13] Richard Peng, *Algorithm design using spectral graph theory*, Ph.D. thesis, Carnegie Mellon University, September 2013. 5.1, 5.4

[Pen14] ______, *A note on cut-approximators and approximating undirected max flows*, CoRR **abs/1411.7631** (2014). 6.1, 6.1.1, 6.5.1

[PS01] Pablo A. Parrilo and Bernd Sturmfels, *Minimizing polynomial functions*, DIMACS Workshop on Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Mathematics and Computer Science, March 12-16, 2001, DIMACS Center, Rutgers University, Piscataway, NJ, USA, 2001, pp. 83–100. 2.1, 2.1.1

[PS13] Richard Peng and Daniel A. Spielman, *An efficient parallel solver for SDD linear systems*, CoRR **abs/1311.3286** (2013). 4.1.2

[Räc02] Harald Räcke, *Minimizing congestion in general networks*, Proceedings of the 43rd Symposium on Foundations of Computer Science, IEEE, 2002, pp. 43–52. 6.1

[Räc08] Harald Räcke, *Optimal hierarchical decompositions for congestion minimization in networks*, Proceedings of the 40th annual ACM symposium on Theory of computing (New York, NY, USA), STOC '08, ACM, 2008, pp. 255–264. (document), 6.1, 6.1.1, 6.2.3, 6.3.2, 6.3.3, 6.3.8, 6.3.3, 6.5

[Räc09] Harald Räcke, *Survey on oblivious routing strategies*, Mathematical Theory and Computational Practice, 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009. Proceedings (Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, eds.), Lecture Notes in Computer Science, vol. 5635, Springer, 2009, pp. 419–429. 6.1.1

[Ren88] James Renegar, *A polynomial-time algorithm, based on newton's method, for linear programming*, Mathematical Programming **40** (1988), no. 1-3, 59–93. 2.1.2

[RST14] Harald Räcke, Chintan Shah, and Hanjo Täubig, *Computing cut-based hierarchical decompositions in almost linear time*, Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, 2014, pp. 227–238. 6, 6.1

[RT14] Peter Richtárik and Martin Takác, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Math. Program. **144** (2014), no. 1-2, 1–38. 6.5.4

[She09] Jonah Sherman, *Breaking the multicommodity flow barrier for $O(\sqrt{logn})$-approximations to sparsest cut*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (Washington, DC, USA), FOCS '09, IEEE Computer Society, 2009, pp. 363–372. 4.1

[She13] Jonah Sherman, *Nearly maximum flows in nearly linear time*, CoRR **abs/1304.2077** (2013). 4.1, 4.1.2, 6, 6.1, 6.1.1, 6.5.1, 6.5.1, 6.5.1

[Spi10] Daniel A. Spielman, *Algorithms, Graph Theory, and Linear Equations in Laplacian Matrices*, Proceedings of the International Congress of Mathematicians, 2010. 5.1

[SS08] Daniel A. Spielman and Nikhil Srivastava, *Graph sparsification by effective resistances*, Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC), 2008, pp. 563–568. 4.1.2

[SS11] Daniel A Spielman and Nikhil Srivastava, *Graph sparsification by effective resistances*, SIAM Journal on Computing **40** (2011), no. 6, 1913–1926. 3.2, 3.2

[ST94] Masahiko Sagae and Kunio Tanabe, *Upper and lower bounds for the arithmetic-geometric-harmonic means of positive definite matrices*, Linear and Multilinear Algebra **37** (1994), no. 4, 279–282. 5.3, 5.3.4

[ST04a] Daniel A. Spielman and Shang-Hua Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.*, Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), June 2004, pp. 81–90. 1.3.3, 4, 4.1, 5.1

[ST04b] Daniel A Spielman and Shang-Hua Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, ACM, 2004, pp. 81–90. 3.1.1

[ST06] Daniel A. Spielman and Shang-Hua Teng, *Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems*, CoRR **abs/cs/0607105** (2006). 5.4, 5.D, 5.D.1

[ST11] ———, *Spectral sparsification of graphs*, SIAM Journal on Computing **40** (2011), no. 4, 981–1025. 5.1

[SW09] Daniel A. Spielman and Jaeoh Woo, *A note on preconditioning by low-stretch spanning trees*, CoRR **abs/0903.2816** (2009). 5.4

[Tar79] Robert Endre Tarjan, *Applications of path compression on balanced trees*, J. ACM **26** (1979), no. 4, 690–715. 4.5.5

[TB97] L.N. Trefethen and D.I. Bau, *Numerical linear algebra*, Society for Industrial and Applied Mathematics, 1997. 5.A

[Ten10] Shang-Hua Teng, *The Laplacian Paradigm: Emerging Algorithms for Massive Graphs*, Theory and Applications of Models of Computation, 2010, pp. 2–14. 5.1

[Tho00] Mikkel Thorup, *Floats, integers, and single source shortest paths*, Journal of Algorithms **35** (2000), 189–201. 4.4.1, 4.5

[Tro11] Joel Tropp, *Freedman's inequality for matrix martingales*, Electron. Commun. Probab. **16** (2011), no. 25, 262–270. 3.3.2

[Tro12] Joel A. Tropp, *User-friendly tail bounds for sums of random matrices*, Found. Comput. Math. **12** (2012), no. 4, 389–434. 5.2, 5.C.1

[Vai91] Pravin M. Vaidya, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners*, A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991. 4.2, 5.4

[Viv59] Vincenzo Viviani, *De maximis et minimis geometrica divinatio liber 2*, De Maximis et Minimis Geometrica Divinatio (1659). 2.1.1

[Vui80] Jean Vuillemin, *A unifying look at data structures*, Commun. ACM **23** (1980), no. 4, 229–239. 2

[VZ00] Yehuda Vardi and Cun-Hui Zhang, *The multivariate l1-median and associated data depth*, Proceedings of the National Academy of Sciences **97** (2000), no. 4, 1423–1426. 2.1.1

[Web09] Alfred Weber, *The theory of the location of industries*, Chicago University Press, 1909, Aber den I der Industrien. 2.1

[Wei37] E. Weiszfeld, *Sur le point pour lequel la somme des distances de n points donnes est minimum*, Tohoku Mathematical Journal (1937), 355–386. 2.1.1

[XY97] Guoliang Xue and Yinyu Ye, *An efficient algorithm for minimizing a sum of euclidean norms with applications*, SIAM Journal on Optimization **7** (1997), 1017–1036. 2.1, 2.1.1

[Ye11] Yinyu Ye, *Interior point algorithms: theory and analysis*, vol. 44, John Wiley & Sons, 2011. 2.1.2

[Zou12] Anastasios Zouzias, *A matrix hyperbolic cosine algorithm and applications*, Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming - Volume Part I (Berlin, Heidelberg), ICALP'12, Springer-Verlag, 2012, pp. 846–858. 5.1