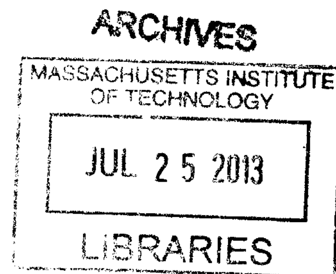
# Graphs, Matrices, and Populations: Linear Algebraic Techniques in Theoretical Computer Science and Population Genetics

by

Alex Levin


ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 25 2013
LIBRARIES

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

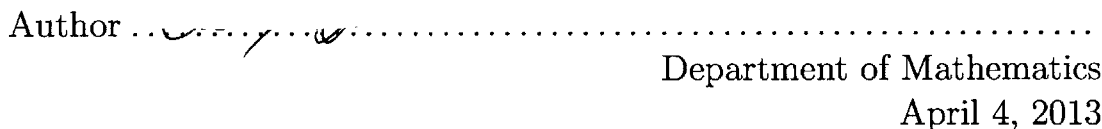
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

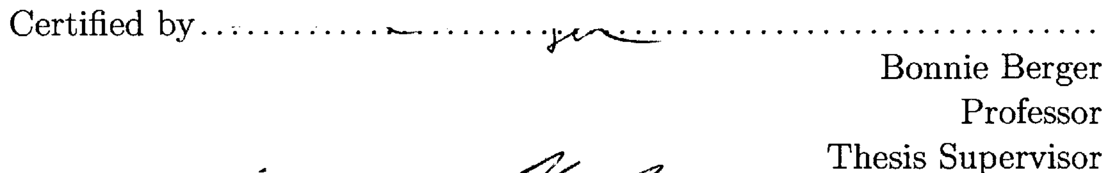
June 2013

© Alex Levin, MMXIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.



Author ..........................................................
Department of Mathematics
April 4, 2013



Certified by ..........................................................
Bonnie Berger
Professor
Thesis Supervisor

Certified by .
Jonathan Kelner
Associate Professor
Thesis Supervisor

Accepted by ..................
Michel Goemans
Chairman, Department Committee on Graduate Theses

# Graphs, Matrices, and Populations: Linear Algebraic Techniques in Theoretical Computer Science and Population Genetics

by

Alex Levin

Submitted to the Department of Mathematics
on April 4, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

In this thesis, we present several algorithmic results for problems in spectral graph theory and computational biology.

The first part concerns the problem of spectral sparsification. It is known that every dense graph can be approximated in a strong sense by a sparse subgraph, known as a *spectral sparsifier* of the graph. Furthermore, researchers have recently developed efficient algorithms for computing such approximations. We show how to make these algorithms faster, and also give a substantial improvement in space efficiency. Since sparsification is an important first step in speeding up approximation algorithms for many graph problems, our results have numerous applications.

In the second part of the thesis, we consider the problem of inferring human population history from genetic data. We give an efficient and principled algorithm for using single nucleotide polymorphism (SNP) data to infer admixture history of various populations, and apply it to show that Europeans have evidence of mixture with ancient Siberians.

Finally, we turn to the problem of RNA secondary structure design. In this problem, we want to find RNA sequences that fold to a given secondary structure. We propose a novel global sampling approach, based on the recently developed RNAmutants algorithm, and show that it has numerous desirable properties when compared to existing solutions. Our method can prove useful for developing the next generation of RNA design algorithms.

Thesis Supervisor: Bonnie Berger
Title: Professor

Thesis Supervisor: Jonathan Kelner
Title: Associate Professor

# Acknowledgments

I would like to thank my advisors Bonnie Berger and Jonathan Kelner for their help and encouragement throughout my years in graduate school. I have learned a great deal from them, and they have truly influenced my thinking about many things, among them mathematics, research, and the art of giving an effective talk.

I had the great fortune to work with some first-rate collaborators, and am grateful for the opportunity. Other than Bonnie and Jon, these people are Srinivas Devadas, Ioannis Koutis, Mark Lipson, Mieszko Lis, Po-Ru Loh, Charles O'Donnell, Nick Patterson, Richard Peng, Yann Ponty, David Reich, and Jérôme Waldispühl. It has been a true pleasure working with them. This thesis is based on research done with these individuals, and I thank them for their contributions.

Laurent Demanet introduced me to an interesting problem in compressed sensing, which I am currently pursuing. I would like to thank him for stimulating discussions, and also for serving on my thesis committee.

Many other individuals have helped me launch my mathematical career. At Harvard, my professors' enthusiasm for mathematics was infectious. Salil Vadhan got me started in theoretical computer science through his wonderful complexity theory course. From Benedict Gross I learned about the beauty of abstract algebra; Prof. Gross also taught me how to convey complex ideas in an engaging and accessible manner. I would also like to thank Cesar Silva of Williams College, where I spent a summer doing research in ergodic theory. Working with Prof. Silva and my fellow students was fun and intellectually stimulating, and gave me my first true research experience.

My friends at MIT have made my time here unforgettable. I would like to thank Eric, George, Höskuldur, Jenn, Jethro, Niels, Qinwen, Rachel, Rune, and many others.

Patrice and the rest of the staff at MIT have always been kind and professional, and have kept things running very smoothly. They have truly gone beyond the call of duty.

Finally, I would like to thank my parents, grandparents, and the rest of my family for their constant encouragement, their support, and for everything else.

For three years of my graduate career, I was sponsored by the National Science Foundation Graduate Research Fellowship. My first year at MIT was supported by an applied math fellowship and by a Liberty Mutual fellowship. I am grateful for the generous support.

# Contents


I Spectral Graph Theory 11

1 Introduction 13

2 Background 19

2.1 Positive semi-definite matrices . . . . . . . . . . . . . . . . . . . . . . . . 19

2.2 Three matrices associated to a graph . . . . . . . . . . . . . . . . . . 21

2.3 Properties of the Laplacian . . . . . . . . . . . . . . . . . . . . . . . . 23

2.4 Random walks on graphs . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.5 Electrical flows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

2.6 Electrical flows and random walks . . . . . . . . . . . . . . . . . . . . 27

2.7 Solving SDD linear systems . . . . . . . . . . . . . . . . . . . . . . . 32

2.8 A note on big-O notation . . . . . . . . . . . . . . . . . . . . . . . . 33

3 Spectral sparsification 35

3.1 Definitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.2 The Spielman-Srivastava algorithm . . . . . . . . . . . . . . . . . . . 37

3.3 Analysis of the Spielman-Srivastava algorithm . . . . . . . . . . . . . 37

3.4 A new algorithm for spectral sparsification . . . . . . . . . . . . . . . 41

3.5 Computing effective resistances . . . . . . . . . . . . . . . . . . . . . 45

3.6 A super-approximation property (optional) . . . . . . . . . . . . . . . 48

4 More background 51

4.1 Primitives for the Koutis-Miller-Peng solver . . . . . . . . . . . . . . 51

4.1.1 Low-stretch spanning trees . . . . . . . . . . . . . . . . . . . . 51
4.1.2 Incremental sparsifiers . . . . . . . . . . . . . . . . . . . . . . 52
4.2 Spine-heavy graphs . . . . . . . . . . . . . . . . . . . . . . . . . . 54

**5 Improved spectral sparsification 57**
5.1 Overview of our results . . . . . . . . . . . . . . . . . . . . . . . . 58
5.1.1 The importance of transitivity . . . . . . . . . . . . . . . . . . 58
5.1.2 The $\mathcal{O}(m \log n)$ algorithm . . . . . . . . . . . . . . . . . . . 58
5.1.3 The $\mathcal{O}(m)$ algorithm . . . . . . . . . . . . . . . . . . . . . . 59
5.2 The $\mathcal{O}(m \log n)$ algorithm . . . . . . . . . . . . . . . . . . . . . . 60
5.3 Effective resistances via very low-dimensional projections . . . . . . . 62
5.4 Improved sparsification via graph decompositions . . . . . . . . . . . 65
5.5 Getting over the Johnson-Lindenstrauss Barrier . . . . . . . . . . . . 67
5.6 Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
5.6.1 Linear system solving . . . . . . . . . . . . . . . . . . . . . . 69
5.6.2 Approximate Fiedler vectors . . . . . . . . . . . . . . . . . . . 70

**6 Semi-streaming setting 73**
6.1 Notation and conventions . . . . . . . . . . . . . . . . . . . . . . . . 76
6.2 The update algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . 76
6.2.1 Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
6.2.2 Estimating effective resistances . . . . . . . . . . . . . . . . . . 79
6.2.3 Putting it all together . . . . . . . . . . . . . . . . . . . . . . 81
6.2.4 Error-forgetfulness of the construction . . . . . . . . . . . . . . 85
6.2.5 Straightforward generalizations . . . . . . . . . . . . . . . . . . 86
6.2.6 The semi-streaming setting . . . . . . . . . . . . . . . . . . . . 86
6.3 Conclusions and future work . . . . . . . . . . . . . . . . . . . . . . 87

**II Population Genetics 88**

**7 Introduction 89**

**8 Background on population genetics** **91**

8.1 Genetic drift . . . 91

8.2 Phylogenetic trees . . . 93

8.3 Admixture graphs . . . 93

8.4 Going to multiple loci . . . 95

**9 Methods** **97**

9.1 Dataset . . . 97

9.2 The $f$-statistics and population admixture . . . 99

9.3 The MixMapper Algorithm . . . 105

9.4 Bootstrapping procedure . . . 106

9.5 Heterozygocity and drift length . . . 107

**10 Results** **111**

10.1 Constructing the pure tree . . . 111

10.2 Case study: The genetic history of European populations . . . 112

10.3 Discussion . . . 113

**III RNA secondary structure design** **118**

**11 Introduction** **119**

**12 Materials and Methods** **123**

12.1 Overview of algorithm . . . 123

12.1.1 The low-energy ensemble of a structure. . . . 123

12.1.2 Sampling from a structure's sequence ensemble. . . . 124

12.1.3 Design algorithm . . . 126

12.2 Software selection . . . 127

12.3 Dataset of random target structures and seed sequences . . . 127

12.4 Dataset of known secondary structures . . . 129

12.5 Structure and sequence analysis . . . 129

12.5.1 Characterizing sequences. . . . . . . . . . . . . . . . . . . . . . . 129
12.5.2 Characterizing structures. . . . . . . . . . . . . . . . . . . . . . . 129
12.5.3 Evaluation of performance . . . . . . . . . . . . . . . . . . . . . 130
12.5.4 The challenges of fair comparison . . . . . . . . . . . . . . . . . 130

**13 Results and Discussion 133**

13.1 Influence of the seed . . . . . . . . . . . . . . . . . . . . . . . . . . 133
13.1.1 Impact on success rate . . . . . . . . . . . . . . . . . . . . . . . 133
13.1.2 Impact on target probability. . . . . . . . . . . . . . . . . . . . . 134
13.1.3 Impact on distance between seed and solution. . . . . . . . . . . 136
13.1.4 Nucleotide composition of designed sequences. . . . . . . . . . . 137
13.2 Influence of the target structure . . . . . . . . . . . . . . . . . . . . . 137
13.2.1 Impact on success rate. . . . . . . . . . . . . . . . . . . . . . . . 138
13.2.2 Impact on target probability and base pair entropy. . . . . . . 139
13.3 Alternate stopping criterion . . . . . . . . . . . . . . . . . . . . . . . 140
13.4 Running time and multiple runs . . . . . . . . . . . . . . . . . . . . . 142

**14 Discussion 145**

# Part I

# Spectral Graph Theory

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 1

# Introduction

Recent years have seen a revolution in spectral graph theory, stemming from new and unexpected algorithmic techniques and primitives. These novel techniques have already had a great impact, and have led to the fastest known algorithms for several classic graph problems.

The new algorithms follow a rich tradition of interplay between graph theory and linear algebra. The connections between random walk properties of a graph and the spectral properties of certain matrices associated to it (e.g. the Laplacian matrix) have been known and exploited for a long time. Most prominent among these is the relationship between the first nontrivial eigenvalue of the Laplacian and quantities such as the mixing rate of a simple random walk on the graph.

One of the primary innovations of the past couple of years has involved going beyond eigenvalue analysis. Recently, researchers have shown how to very quickly solve linear systems in graph Laplacians, and these methods have in turn led to advances in graph algorithms. The fastest-known algorithm for approximating maximum flows on undirected graphs [14], for example, crucially uses these solvers. One of the concepts that has been quite fruitful is viewing a graph as an electrical network (with vertices as nodes and edges as wires), and studying quantities such as potentials and effective resistances.

Linear algebraic techniques have thus had a large impact on designing fast graph algorithms, and new graph techniques have crucially relied on advances in linear alge-

bra. In fact, the payoff has gone the other way as well, and the story is quite involved. Starting with the work of Vaidya [73], graph theoretic constructions have been used to precondition linear systems in Laplacians, and, more generally, in symmetric diagonally dominant (SDD) matrices, and have thus played a central role in speeding up solvers for these systems.

Spielman and Teng gave the first nearly-linear time algorithm for solving SDD systems; given an $n \times n$ matrix with $m$ nonzero entries, their algorithm produces a solution with accuracy $\delta$ in time $O(m \log^{O(1)} n \log(1/\delta))$. Unfortunately, the exponent of the logarithm in the original Spielman-Teng paper was quite large. This was remedied by seminal works of Koutis, Miller, and Peng [41, 42], who gave an $O(m \log n \log(1/\delta))$ algorithm. In the past year, Kelner, Orecchia, Sidford, and Zhu proposed a remarkably simple technique that runs in time $O(m \log^2 n \log(1/\delta))$ [38].

In all these cases, graph theoretic ideas, particularly notions of graph approximation, have proved crucial to the construction of the solver.

The rich and bidirectional interplay between graph theory and linear algebra offers great promise, and we explore part of the story in this thesis.

## Spectral sparsification

The tangled connections between linear algebra and graph theory are perhaps most vividly on display in the context of *spectral sparsification.* On a high level, a spectral sparsifier of a dense graph $G$ is a weighted subgraph of $G$ that is sparse, yet gives a strong approximation of $G$ in an algebraic sense. Namely, the Laplacian matrices $L_G$ and $L_H$ of $G$ and $H$ are good spectral approximations of each other.

In particular, sparsifiers seem especially well-suited for the problem of solving linear systems in graph Laplacians; if $H$ is a sparsifier of $G$, then the Laplacian of $H$ is a good preconditioner of the Laplacian of $G$. A natural question, then, is whether we could construct a sparsifier quickly enough for it to be useful for speeding up linear system solving.

A very conceptually elegant and fast construction for spectral sparsifiers was given

by Spielman and Srivastava. However, it relies on efficient algorithms for solving linear systems, and thus, in its original form, is not itself useful for linear system solving. Nevertheless, Koutis, Miller, and Peng used ideas from this algorithm in their papers [41, 42]. On a high level, their work asks how far one could push the Spielman-Srivastava construction without relying on solving linear systems. The objects they get in that manner (which they call *incremental sparsifiers*) are moderately sparse, and also provide an adequate spectral approximation. While not as good in quality or sparsity as spectral sparsifiers, they nevertheless provide preconditioners that allow them to give fast solutions to linear systems.

## Contributions of this thesis

In this thesis, we improve and extend algorithms for spectral sparsification. Because of the connections between spectral sparsification and linear system solving, we get faster algorithms for the latter problem as well.

Our first contribution involves speeding up spectral sparsification using a number of techniques. We start with a natural question of finding tradeoffs between output size and the running time of the sparsification algorithm. Specifically, we ask whether, by tolerating output graphs that are bigger (by a polylogarithmic factor) than the ones output by Spielman and Srivastava, we can make the algorithm run faster. It turns out that we can do this using the internals of the Koutis-Miller-Peng linear system solver. By running the original Spielman-Srivastava algorithm on the result of this procedure, we are able to give a faster spectral sparsification algorithm.

We then introduce a number of other techniques to push down the running time of the algorithm even further. At each step, we need a more precise understanding of the internal details of Spielman and Srivastava's algorithm. At the end, we are able to give an $O(m(\log\log n)^{O(1)})$ algorithm for spectral sparsification of graphs that are sufficiently dense.

Our faster algorithms for spectral sparsification directly go through to speed up several numerical algorithms. In particular, the first technique gives the fastest run-

ning time algorithm known for computing approximate Fiedler vectors, which has implications for numerous problems, such as graph partitioning. Additionally, the sparsifiers produced by our $O(m(\log\log n)^{O(1)})$ algorithm can be used for preconditioning linear systems, and because the algorithm is so efficient, this procedure gives a faster algorithm for linear system solving. The results hold additional theoretical appeal, since our construction is the first one that makes Spielman-Srivastava-type sparsifiers useful for linear system solving.

Our second contribution is a low-space algorithm for spectral sparsification. The algorithm given by Spielman and Srivstava first performs a computation on the entire dense graph in order to output probabilities associated with all the edges, and then uses these to sample edges in order to obtain a sparsifier. However, for dense graphs of the type one would eventually like to sparsify, this may be an unreasonable resource requirement: in particular, we may not be able to store the entire graph in memory in order to compute the probabilities.

We would like an algorithm to work when we receive a graph as a stream of edges, and only have a small amount of storage space to work with. Ideally, the work space should not be much larger than the space required to store the sparsifier we eventually output. Furthermore, we would like the procedure to have a running time that is comparable to that of the original Spielman-Srivastava algorithm.

This question has been studied in the context of cut-preserving sparsifiers by prior work, as well as by work that is concurrent with and independent from ours. We are able to give a conceptually elegant solution for the case of spectral sparsification, based on a simple "rejection sampling" technique.

# Organization

In Chapter 2, we give a basic introduction to spectral graph theory, focusing primarily on properties of the graph Laplacian. We also introduce the connection between electrical flows and random walks on graphs. Chapter 3 introduces the concept of spectral sparsification, and presents the algorithm due to Spielman and Srivastava.

We give a self-contained analysis of this algorithm, and also present and analyze a different sparsification procedure. This new algorithm for sparsification is original to our work, and also proves useful in the semi-streaming algorithm.

We continue our overview of background material in Chaper 4, where we summarize recent work on approximately solving symmetric diagonally dominant linear systems. While still expository, this material is quite cutting-edge.

In Chapter 5 we present our faster algorithms for spectral sparsification, and their applications to designing faster linear system solvers. We also give the fastest known algorithm for computing approximate Fiedler vectors.

Finally, in Chapter 6 we give our space-efficient algorithm for spectral sparsification.

Throughout, we have included a number of clearly-marked optional sections. These sections present research material that is inchoate at this stage, though still interesting, or background material that is not strictly necessary to the subsequent discussion. The optional sections can be safely skipped on first reading.

## Bibliographic notes

This part of the dissertation is based on published works with several coauthors (Jonathan A. Kelner, Ioannis Koutis, and Richard Peng). Our semi-streaming algorithm was published as [37], and a journal version is forthcoming in *Theory of Computing Systems*. Our results on improved algorithms for spectral sparsification and linear system solving were originally published as [40] and improved in a followup work [39]; an early version of these ideas was presented in the unpublished manuscript [46]. We have reorganized some of the material to improve the flow of the current document. For example, some of the background material we present is interspersed with original results from those papers.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 2

# Background

Spectral graph theory is the study of graphs using certain matrices associated to them. In particular, the Laplacian, which we define in this chapter, is the fundamental object that gives an "algebraic encoding" of the graph. Before we define the Laplacian, however, we need to review some material on symmetric positive semi-definite matrices.

## 2.1 Positive semi-definite matrices

Let $A$ be an $n \times n$ symmetric matrix. It is a standard fact that it is diagonalizable with real eigenvalues and orthonormal eigenvectors. If $\lambda_i$ are the eigenvalues and $u_i$ are the corresponding eigenvectors, then we can write $A$ as $A = \sum_{i=1}^{n} \lambda_i u_i u_i^T = U \Lambda U^T$.

**Definition 2.1.1.** We say that $A$ is *positive semi-definite* and write $A \succeq 0$ if all the eigenvalues of $A$ are non-negative.

Equivalently, $A$ is positive semi-definite if for all $x \in \mathbb{R}^n$ it is the case that $x^T A x \geq 0$.

We say that $A$ is positive definite and write $A \succ 0$ if all its eigenvalues are strictly greater than 0, or equivalently, $x^T A x > 0$ for all $x \in \mathbb{R}^n$.

Suppose that $A$ is positive semi-definite, and its eigenvalue decomposition is given as $\sum_{i=k+1}^{n} \lambda_i u_i u_i^T$, where the $\lambda_i$ in the sum are strictly greater than zero (in other

words, we let $\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0$). Then, we denote by $A^+$ the *Moore-Penrose pseudo-inverse* of $A$. One way of defining it is as $A^+ = \sum_{i=k+1}^n \lambda_i^{-1} u_i u_i^T$. From this, it is not hard to see that $AA^+ = A^+A$, and is equal to the projection onto $\mathrm{span}(u_{k+1}, \ldots, u_n)$.

Given two $n \times n$ matrices $A$ and $B$, we say that $A \preceq B$ if $B - A \succeq 0$. In other words, $A \preceq B$ if and only if for all $x \in \mathbb{R}^n$ it is the case that $x^TAx \leq x^TBx$. Note that $A$ and $B$ might have very different eigenvectors.

Finally, it is a standard fact that for a symmetric matrix $A$ it is the case that $\mathrm{im}(A) = \ker(A)^\perp$.

We now give a few standard results on positive semi-definite matrices. The proofs are fairly simple, but the results are fundamental to our work, so we include them for completeness.

In what follows, and throughout the work, we will use notation such as $A^{-1/2}$ even when $A$ is not invertible. The notation will mean $(A^+)^{1/2}$.

**Proposition 2.1.2.** *Suppose that $A$ and $B$ are positive semi-definite symmetric matrices that have the same kernel. Then $A \preceq B$ if and only if $B^{-1/2}AB^{-1/2} \preceq I$, where $I = A^+A = B^+B$ is the projection onto the image. Similarly, $A \succeq B$ if and only if $B^{-1/2}AB^{-1/2} \succeq I$.*

*Proof.* We will only prove the first statement, as the second one is similar.

In what follows, it is enough to prove all statements about inequality of quadratic forms for vectors in $\ker(A)^\perp = \mathrm{im}(A)$, as cross terms will cancel.

Suppose that $A \preceq B$. Then, for all $x \in \ker(A)^\perp$ it is the case that $x^TAx \leq x^TBx$. Then, for any $y$, we have $y^TB^{-1/2}AB^{-1/2}y \leq y^TB^{-1/2}BB^{-1/2}y = y^Ty$, which shows that $B^{-1/2}AB^{-1/2} \preceq I$.

Conversely, suppose that $B^{-1/2}AB^{-1/2} \preceq I$. We want to show that $x^TAx \leq x^TBx$ for all $x$. Take $y = B^{1/2}x$. Then, $x^TAx = y^TB^{-1/2}AB^{-1/2}y$, which, by the hypothesis, is at most $y^Ty = x^TBx$, as required. $\square$

In the propositions that follow, we let $I_{\mathrm{im}(A)}$ denote the projection onto the image of $A$ (and in fact, $I_{\mathrm{im}(A)} = AA^+$).

**Proposition 2.1.3.** *Suppose* $0 \preceq A \preceq I_{\mathrm{im}(A)}$. *Then,* $A^+ \succeq I_{\mathrm{im}(A)}$.

*Proof.* Let $\lambda_{\mathrm{min}}$ and $\lambda_{\mathrm{max}}$ be the smallest and biggest nonzero eigenvalues of $A$ respectively. Then, by the hypotheses, $\lambda_{\mathrm{min}} \geq 0$, and $\lambda_{\mathrm{max}} \leq 1$. Furthermore, $\lambda_{\mathrm{min}}^{-1}$ and $\lambda_{\mathrm{max}}^{-1}$ are the largest and smallest eigenvalues respectively of $A^+$. Because the smallest eigenvalue of $A^+$ is greater than 1, it follows that $A^+ \succeq I_{\mathrm{im}(A)}$, as required. □

**Theorem 2.1.4.** *Suppose* $0 \preceq A \preceq B$ *and* $A$ *and* $B$ *have the same kernel. Then, it is the case that* $B^+ \preceq A^+$.

*Proof.* Suppose that $0 \preceq A \preceq B$. Then, we have $0 \preceq B^{-1/2}AB^{-1/2} \preceq I$. By Proposition 2.1.3, we know that $B^{1/2}A^+B^{1/2} \succeq I$. But that means that $A^+ \succeq B^+$, as desired. □

**Proposition 2.1.5.** *Let* $A$, $B$ *and* $C$ *be positive semi-definite symmetric matrices, and suppose that* $A \preceq B$. *Then,* $\mathrm{Tr}(CA) \leq \mathrm{Tr}(CB)$.

*Proof.* By cyclicity of the trace, we know that $\mathrm{Tr}(CA) = \mathrm{Tr}(C^{1/2}AC^{1/2})$, which is equal to $\sum_{i=1}^n \chi_i^T C^{1/2}AC^{1/2}\chi_i$. Here, $\chi_i$ is the $n \times 1$ vector with 1 in the $i$th entry and 0's everywhere else. Because $A \preceq B$, this quantity is less than or equal to

$$\sum_{i=1}^n \chi_i^T C^{1/2}BC^{1/2}\chi_i = \mathrm{Tr}(CB).$$

This completes the proof. □

## 2.2 Three matrices associated to a graph

Let $G = (V, E)$ be an undirected graph with $n$ vertices and $m$ edges. We identify the vertex set $V$ with $\{1, 2, \ldots, n\}$. Then, $A_G$ is the adjacency matrix of $G$, which is an $n \times n$ matrix with 0's on the diagonal and a 1 at $(i, j)$ if there is an edge between $i$ and $j$. Since the graph is undirected, $A_G$ is a symmetric matrix.

Additionally, we define $D_G$ to be the diagonal matrix whose $i$th diagonal entry is the degree of vertex $i$.

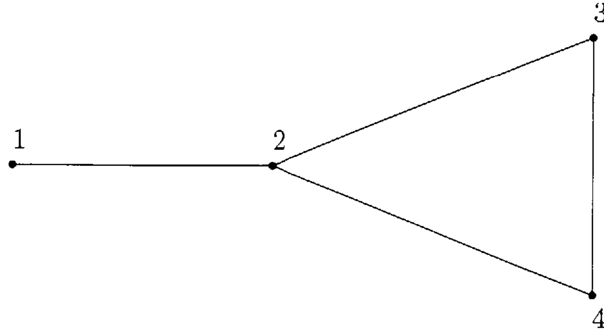



Figure 2-1: An example graph

It is easy to extend these notions to weighted graphs (we assume that all weights are nonnegative). Indeed, the degree of each vertex is now the weighted degree, i.e. the sum of weights of edges incident on the vertex. We can let $D_G$ be the diagonal matrix of weighted degrees. As for $A_G$, the $(i, j)$ entry is now given by the weight $w_{i,j}$ of the edge between $i$ and $j$.

We are now ready to define the Laplacian, which plays a crucial role in spectral graph theory, and by extension, in this thesis. The definition works equally well for unweighted and weighted graphs.

**Definition 2.2.1.** The Laplacian of $G$, written as $L_G$, is given by $L_G = D_G - A_G$.

For example, consider the (unweighted) graph $G$ in Figure 2-1. It is easy to see that

$$A_G = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \qquad D_G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

and

$$L_G = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

## 2.3 Properties of the Laplacian

The Laplacian has numerous useful properties, which we summarize below. Firstly, for an edge $e = (i, j)$ of $G$, we let $b_e \in \mathbb{R}^n$ be the vector $\chi_i - \chi_j$, i.e. the vector with 1 at $i$, a $-1$ at $j$ and 0 everywhere else. There is a degree of ambiguity in what endpoint of $e$ should take the positive sign, and we make this choice arbitrarily. In the end, it will not matter.

With this definition, it is easy to see that $L_G = \sum_{e \in G} w_e b_e b_e^T$. If we let $L_e$ be the Laplacian of a graph with vertex set $V$ and whose only edge is $e$, it is easy to see that $L_e = w_e b_e b_e^T$. Then, $L_G = \sum_{e \in G} L_e$.

From this one can show that if $x$ is a vector, then

$$x^T L_G x = \sum_{e=(i,j) \in G} w_e (x_i - x_j)^2.$$

In particular, since this sum is always non-negative, this shows that the Laplacian is a positive semi-definite matrix. Furthermore, if $G$ is connected, then $L_G$ has a one-dimensional kernel spanned by the constant vector. We define $I_{n-1}$ to be $L_G L_G^+$, the projection onto the $(n-1)$-dimensional space $\ker(L_G)^\perp$.

Let us consider a special vector $x$, namely the characteristic vector $\chi_S$ of some subset $S$ of the vertices. This $\chi_S$ is a vector that is 1 at elements of $S$ and 0 elsewhere. What edges contribute to the sum in the expression for $\chi_S^T L_G \chi_S$? These are precisely the edges $(i, j)$ such that $i \in S$ and $j \notin S$ or vice versa, i.e. the edges crossing the cut $(S : \bar{S})$. Therefore, $\chi_S^T L_G \chi_S$ is exactly the sum of weights crossing this cut, which is known as the cut value.

Let $B_G$ be an $m \times n$ matrix with rows indexed by edges of $G$ and columns indexed by vertices, and whose $e$th row $b_e^T$. Let $W_G$ be the $m \times m$ diagonal matrix indexed by edges whose $e$th diagonal entry is $w_e$, the weight of edge $e$. Then, it is not hard to see that $L_G = B_G^T W_G B_G$. We will often drop the subscripts on $L_G, B_G$, and $W_G$ when the graph we are dealing with is clear from context.

## 2.4 Random walks on graphs

Let us consider a *random walk* on the graph $G$. We start at any vertex, and if we are at vertex $i$ at a given time step, we visit a neighbor $j$ with probability proportional to the weight of the edge between $i$ and $j$. In other words, the probability of visiting $j$ is $w_{i,j}/d_i$.

This process is a Markov chain with state space given by the vertices of $G$. The transition matrix is $M_G = D_G^{-1} A_G$. For the graph in Figure 2-1, the transition matrix is

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/3 & 0 & 1/3 & 1/3 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

Suppose that $\boldsymbol{p} \in \mathbb{R}^n$ is a probability distribution on the vertices of $G$, i.e. each $p_i$ is non-negative and they all sum to 1. Then, the probability distribution after $r$ steps of the random walk is given by $(M_G^T)^r \boldsymbol{p}$.

Random walks on graphs are very natural objects of study, and, in addition, have numerous applications ranging from Markov Chain Monte Carlo methods to local graph partitioning. We refer the reader to any standard book or article for more information. One rather surprising application, which will be useful to our subsequent discussion, is the following simple algorithm for generating a uniformly random spanning tree of an unweighted graph $G$, due to Aldous [7]: We start a random walk at any vertex, and, as the random walk goes across edges, we add these edges to the tree provided they do not make a cycle with the previously added edges.

## 2.5 Electrical flows

The connection between spectral graph theory and electrical network theory has received a great deal of attention in recent years. We think of the graph as a network of nodes (vertices) and wires (edges), where an edge $e$ with weight $w_e$ has conductivity $w_e$ (equivalently, its resistivity is $w_e^{-1}$).

Let us consider driving one unit of current between vertices $i$ and $j$. Think of attaching a power source to $i$ and $j$ and dialing up the voltage until one unit of current flows between them.

In this case, what are the voltages at all the nodes? Let $\Phi \in \mathbb{R}^n$ be a vector of voltages at the nodes (so that $\Phi_r$ is the voltage at vertex $r$). Using Kirchoff's rules, we see that the net current coming into vertices $r$ other than $i$ and $j$ is 0, whereas the net current coming into $j$ and $i$ is 1 and $-1$ respectively. This means that

$$\sum_{s \sim r} w_{s,r}(\Phi_r - \Phi_s) = d_r \Phi_r - \sum_{s \sim r} w_{s,r} \Phi_s = \begin{cases} 0, \text{ if } r \neq i, j \\ 1, \text{if } r = i \\ -1, \text{if } r = j \end{cases} \tag{2.1}$$

The notation $s \sim r$ means that there is an edge between $r$ and $s$, and $w_{r,s}$ is the weight of the edge $(r, s)$ in $G$.

This equation is basically a statement of Kirkoff's law. The quantity $\Phi_r - \Phi_s$ is the potential difference between $r$ and $s$; by Ohm's Law, dividing it by the resistivity of the edge $(r, s)$, or equivalently multiplying by $w_{r,s}$ gives the current from $s$ to $r$ along the edge. Then, the first sum in (2.1) is just the net current flowing out of vertex $r$, which is given by 0 and $\pm 1$ depending on what $r$ is, as in the right hand side of the equation.

Notice also that the sum for a given $r$ is equal to the $r$th entry of $L_G \Phi$. It follows that $L_G \Phi = \chi_i - \chi_j$, the vector that is 1 at $i$, $-1$ at $j$, and 0 everywhere else. Therefore, if we could compute the pseudoinverse of $L_G$, we could compute $\Phi$ (at least up to addition of the constant vector). Note that this is good enough, since *differences* in potentials, rather than potentials themselves, are the physically meaningful quantities.

In summary, the vector of voltages is given by $L_G^+(\chi_i - \chi_j)$.

**Definition 2.5.1.** The effective resistance in $G$ between $i$ and $j$, denoted by $R^G(i, j)$, is given by the voltage difference that would result between $i$ and $j$ if we drove one unit of current between them.

From the above discussion, it is not hard to see that

$$R^G(i,j) = (\chi_i - \chi_j)^T L_G^+(\chi_i - \chi_j). \tag{2.2}$$

Indeed, $\Phi = L_G^+(\chi_i - \chi_j)$ is the vector of potentials set up when we send a unit of current between $i$ and $j$ and $(\chi_i - \chi_j)^T\Phi$ is the difference in potential between the $i$th and $j$th vertices. Therefore, effective resistances in $G$ are given by quadratic forms in $L_G^+$ evaluated at particular vectors.

Suppose that $e = (i,j)$ is an edge of the graph. Then, we write $R^G(e)$ to mean the effective resistance between the endpoints of $e$, i.e. $R^G(e) = R^G(i,j)$. When the graph $G$ is clear from context, it will sometimes be convenient to denote $R^G(e)$ by $R_e$, and we use the latter notation extensively in some parts of the thesis.

**Proposition 2.5.2.** *Let $G$ be a connected graph. Then,*

$$\sum_{e\in G} w_e R^G(e) = n - 1$$

*Proof.* We know that $R^G(e) = b_e^T L_G^+ b_e = (L_G^{-1/2} b_e)^T (L_G^{-1/2} b_e)$, which is equal to $\operatorname{Tr}\left(L_G^{-1/2} b_e b_e L_G^{-1/2}\right)$ Then

$$\begin{aligned}
\sum_{e\in G} w_e R^G(e) &= \sum_{e\in G} w_e \operatorname{Tr}\left(L_G^{-1/2} b_e b_e L_G^{-1/2}\right)\\
&= \operatorname{Tr}\left(\sum_{e\in G} w_e L_G^{-1/2} b_e b_e L_G^{-1/2}\right)\\
&= \operatorname{Tr}\left(\sum_{e\in G} L_G^{-1/2} w_e b_e b_e^T L_G^{-1/2}\right)\\
&= \operatorname{Tr}(L_G^{-1/2} L_G L_G^{-1/2})\\
&= \operatorname{Tr}(I_{n-1})\\
&= n - 1
\end{aligned}$$

as required. □

Another useful observation is the fact that if we are given a graph, and we modify it

by adding edges or raising the weights of existing edges, then the effective resistances between pairs of vertices cannot increase. This will prove very useful in our subsequent discussion, and is a crucial ingredient of our algorithm for sparsification in the semi-streaming setting.

More formally:

**Proposition 2.5.3** (Monotonicity). *Let $G_1$ be a graph, and let $G_2$ be the graph we get if we add edges to $G_1$ or raise the weights of existing edges. Then, for all vertices $i$ and $j$ it is the case that*

$$R^{G_1}(i,j) \geq R^{G_2}(i,j).$$

*Proof.* Clearly, $L_{G_1} \preceq L_{G_2}$. Therefore, $L_{G_1}^+ \succeq L_{G_2}^+$, and the proposition follows by the fact that effective resistances are just quadratic forms of Laplacians evaluated at particular vectors. □

This proposition is also fairly clear from the physical model. An alternate perspective and proof of this fact is given in [44, Lecture 9].

## 2.6 Connections between electrical flows and random walks on graphs (optional)

The concept of effective resistance has numerous connections to random walk properties of the graph. For example, for two adjacent vertices $i$ and $j$, the effective resistance between $i$ and $j$ is related to the probability that, given a random walk starting at $i$, the walk will visit $j$ before returning to $i$. Specifically, if we let $p$ be this probability, then we have $p = 1/(d_i R^G(i,j))$. Furthermore, from this property, it is not hard to see that when $G$ is unweighted and $(i,j)$ is an edge, the effective resistance between $i$ and $j$ is exactly the probability that the edge $(i,j)$ is in a uniformly random spanning tree of $G$.

We follow the presentations in [24, 44], and we assume that our graph is connected.

**Definition 2.6.1.** Let $G$ be a graph, and $i$ and $j$ be vertices. The *hitting time* of $i$ to $j$, written $\mathcal{H}_{ij}$, is the expected number of steps it takes for a random walk starting at $i$ to visit $j$.

**Definition 2.6.2.** Given a graph $G$ and two vertices $i$ and $j$, the *commute time* between $i$ and $j$, denoted by $\mathcal{C}_{ij}$, is given by

$$\mathcal{H}_{ij} + \mathcal{H}_{ji},$$

i.e. the expected number of steps it takes for a random walk on $G$ starting $i$ to travel to $j$ and then return to $i$.

Note that the commute time is the same even if we flip the indices.

Let $T_i$ be the (random) time that a random walk starting at $i$ returns to $i$. Let $T_{ij}$ be the first time that a random walk starting at $i$ returns to $i$ after visiting $j$. Clearly $T_i \leq T_{ij}$; by the time a random walk returns to $i$ after it has visited $j$ it may have already returned to $i$ previously.

**Proposition 2.6.3.** *It is the case that* $E[T_i] = 2m/d_i$.

*Proof.* The standard way to prove this fact uses the observation that in the stationary distribution $\pi$ of the random walk on $G$ the probability of vertex $i$ is $\pi_i = d_i/(2m)$. We sketch the argument. Consider starting a random walk of $N$ steps, where $N$ is effectively infinite, at the stationary distribution $\pi$. Then, the number of times the walk visits $i$ is $N\pi_i$, and the expected return time to $i$ is the average interval between visits to $i$. This is given by the length of the random walk divided by the expected number of times it visits $i$, i.e. $N/N\pi_i = 1/\pi_i$.

We give a non-standard proof here. For ease of notation, we assume without loss of generality that $i = 1$. Let $\varphi_j$ be the expected number of steps to reach 1 starting at $j$. Then, $\varphi_1 = 0$ and for $j \neq 1$ it is the case that $\varphi_j = 1 + \sum_{k \sim j}(w_{j,k}/d_j)\varphi_k$. To see this, note that $w_{j,k}/d_j$ is the probability of going to $k$ from $j$. Then, the expected time to return to 1 when starting from $j$ is the average time to return to 1 from a neighbor $k$ of $j$ (weighted by the probability of visiting $k$) plus 1 (for the initial step

to $k$). Multiplying through by $d_j$ and moving the sum to the left hand side, we see that $d_j\varphi_j - \sum_{k\sim j} w_{j,k}\varphi_k = d_j$. Therefore, $(L_G\varphi)_j = d_j$ for $j \neq 1$.

Now, $(L_G\varphi)_1 = d_1\varphi_1 - \sum_{k\sim 1} w_{1,k}\varphi_k$, and under the assumption that $\varphi_1 = 0$ this gives $-\sum_{k\sim 1} w_{1k}\varphi_k$.

Therefore,

$$L_G\varphi = \begin{pmatrix} -\sum_{k\sim 1} w_{1,k}\varphi_k \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{pmatrix}.$$

This system must be solvable, so it must be the case that the vector on the right hand side is orthogonal to the kernel of $L_G$, i.e. its entries add up to 0. Since $\sum_{j=2}^{n} d_j = 2m - d_1$, we see this amounts to $2m - d_1 - \sum_{k\sim 1} w_{1,k}\varphi_k = 0$, i.e. $1 + \sum_{k\sim 1} w_{1,k}\varphi_k = 2m/d_1$. Noting that the left hand side of this expression is the expected amount of time it takes for a random walk starting at 1 to return to 1, we see that we have shown the assertion of the proposition. □

Now, notice that $\mathcal{C}_{ij} = E[T_{ij}]$. We let $p$ be the probability that a random walk starting at $i$ visits $j$ before returning to $i$; it is not hard to see that $p = \Pr[T_i = T_{ij}]$.

Furthermore, as in [24], we can see that

$$\begin{aligned} E[T_{ij}] - E[T_i] &= E[T_{ij} - T_i] \\ &= pE[T_{ij} - T_i | T_{ij} = T_i] + (1-p)E[T_{ij} - T_i | T_i < T_{ij}] \\ &= (1-p)E[T_{ij}] \end{aligned}$$

The last line follows because in the case that $T_i < T_{ij}$, the walk has returned to $i$ without visiting $j$ first. The expected additional time beyond $T_i$ it takes to visit $j$ is exactly $E[T_{ij}]$.

The previous equation means that $p = E[T_i]/E[T_{ij}]$. Substituting in $E[T_i] = 2m/d_i$ and $E[T_{ij}] = \mathcal{C}_{ij}$, we have proved:

**Theorem 2.6.4.** *Given a random walk starting at $i$, the probability that it visits $j$ before returning to $i$ is given by*

$$\frac{2m}{d_i \mathcal{C}_{ij}}$$

*where $d_i$ is the degree of $i$.*

Let us now connect these concepts with electrical flows. Consider the vector $\varphi$, where $\varphi_k$ is the probability that a random walk starting at $k$ visits $j$ before $i$ (with the convention that the random walk visits the vertex at which it starts). Then, clearly $\varphi_i = 0$ and $\varphi_j = 1$. Furthermore, for $k \neq i, j$ it is the case that $\varphi_k = (1/d_k) \sum_{\ell \sim k} w_{k,\ell} \varphi_\ell$. Indeed, this just decomposes the relevant probability depending on the first step. The random walk visits node $\ell$ with probability $w_{k,\ell}/d_k$, and conditioned on this first step, the probability of visiting $j$ before $i$ is exactly $\varphi_\ell$ (this holds even if $\ell$ is $i$ or $j$).

It follows that $(L\varphi)_k = 0$ for $k \notin \{i, j\}$ and $\varphi_j - \varphi_i = 1$.

Consider $\tilde{\varphi} = -\Phi / R^G(i, j)$. By adding a constant to all components of $\Phi$, we can assume that $\Phi_i = 0$; doing this does not affect the electrical flow produced by $\Phi$. We know that $(L\tilde{\varphi})_k = 0$ for $k \notin \{i, j\}$ and $\tilde{\varphi}_j - \tilde{\varphi}_i = 1$. Therefore, by uniqueness of solutions to the (discrete) Laplace equation, we must have $\varphi = \tilde{\varphi}$.

Now, the probability that a random walk starting at $i$ visits $j$ before returning to $i$ is exactly

$$\frac{1}{d_i} \sum_{k \sim i} w_{i,k} \varphi_k = \frac{1}{d_i} (L\varphi)_i. \tag{2.3}$$

The quantity $(L\varphi)_i$, in turn, is the current flowing into $i$ when the vector of voltages is given by $\varphi$, which is exactly $1/R^G(i, j)$. Indeed, when the voltages are given by $\Phi$, there is one unit of current flowing out of $i$, and the statement follows since $\varphi = -\Phi / R^G(i, j)$. We conclude that the desired probability in (2.3) is $1/(d_i R^G(i, j))$.

By setting $2m/\mathcal{C}_{ij}$ equal to $1/R^G(i, j)$, we see:

**Theorem 2.6.5.** *The commute time $\mathcal{C}_{ij}$ is given by $2mR^G(i, j)$*

Furthermore, we can now easily prove a well-known result in the case that there is an edge between $i$ and $j$.

**Corollary 2.6.6.** *Suppose $G$ is unweighted. If there is an edge between $i$ and $j$, then the commute time between $i$ and $j$ is at most $2m$.*

*Proof.* Indeed, if there is an edge between $i$ and $j$, then the effective resistance between $i$ and $j$ is at most 1, hence, by the above theorem, the commute time is at most $2m$, as required. □

Finally, we relate the effective resistance of an edge $e$ to the probability that the edge will be chosen in a random spanning tree. Let $e = (i, j)$ and consider running Aldous's algorithm for generating a random spanning tree using a random walk starting at $i$. Let $q$ be the probability that $e$ will be in a random spanning tree, and $p$, as above, be the probability that a random walk starting at $i$ visits $j$ before returning to $i$.

Then, we see that

$$q = \frac{1}{d_i} + (1 - p)q.$$

Indeed, let us condition first of the events that the random walk does or does not visit $j$ before returning to $i$. In the former case, with probability $1/d_i$, it visits $j$ on its first move, and in that case the edge is taken to be in the spanning tree. On the other hand, if it visits any other vertex first but still visits $j$ before returning to $i$, then there is no way at $e$ will be taken, since it will cause a loop.

Condition now on the event that the random walk returns to $i$ before visiting $j$. Then, in terms of visiting $j$, we are back to where we started, and the edge is taken for the spanning tree in the same circumstances that it is at the beginning of the random walk. This happens, in particular, with probability exactly $q$.

Therefore, by solving the above equation for $q$, we see that $q = 1/(d_i p) = R^G(i, j)$, as required.

**Theorem 2.6.7.** *In an unweighted graph $G$, the probability that an edge $e$ is in a random spanning tree of $G$ is exactly $R^G(e)$.*

As an example, consider an edge $e$ whose removal would disconnect the graph. The effective resistance of this edge is 1, and it is also in every spanning tree of the graph, so the probability of it being in a random spanning tree is 1 as well.

This theorem also gives us another intuitive explanation for the monotonicity law (Proposition 2.5.3). Indeed, suppose that we start with an unweighted graph $G_1$ and add edges to produce a new graph $G_2$. Consider an edge in $e$ of $G_1$. Since we have added new edges to make $G_2$ it makes intuitive sense that the probability that this edge is in a random spanning tree of $G_2$ is at most the probability that it is in a random spanning tree of $G_1$. Indeed, with new edges added, there are more opportunities to make spanning trees, and any given edge of $G_1$ becomes less essential.

## 2.7 Solving SDD linear systems

In this section, we summarize some results on approximately solving symmetric diagonally dominant linear systems.

**Definition 2.7.1.** We say that a matrix $A$ is symmetric diagonally dominant (SDD) if it is symmetric and for all $i$ it is the case that $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$.

Suppose we are given an SDD matrix $A$ that is $n \times n$ with $m$ nonzero entries. Say we want to solve the system $Ax = b$, and, in a slight abuse of notation, let $x$ be the true solution. The methods developed in recent years allow us to find a vector $\tilde{x}$ such that $||x - \tilde{x}||_A \leq \delta ||x||_A$, where $|| \cdot ||_A$ is the $A$-norm (i.e. $||y||_A = y^T A y$). The running time is $O(m \log^{O(1)} n \log(1/\delta))$.

We may also try to "solve" linear systems in the case that $A$ is not invertible. In this case, the "solution" is $A^+ b$, and the results from above apply to this case as well.

In fact, Spielman and Teng found it useful for their analysis to consider the solver as a linear map. To do this, they ran the iterative steps a fixed number of times, and used Chebytschev iterations—essentially iteratively computing a polynomial of the matrix $A$—rather than the conjugate gradient method.

When this is done, we can get the following stronger result, using the algorithms of Koutis, Miller, and Peng [42]. Specifically, the solver linear map defines a linear map that approximates $A$ in the spectral sense:

**Lemma 2.7.2.** *Let $A$ be an SDD matrix. There is a symmetric operator $\tilde{A}_\delta$ such that*

$$(1-\delta)A \preceq \tilde{A}_\delta \preceq (1+\delta)A$$

*and that for any vector $b$, the vector $\tilde{A}_\delta^+ b$ can be evaluated in $\mathcal{O}(m \log n \log(1/\delta))$ time.*

The notation $\mathcal{O}(\cdot)$ hides factors polynomial in $\log\log n$ (see Section 2.8 for more details).

The condition that $\tilde{A}_\delta^+ b$ can be quickly evaluated just says that we can solve linear systems quickly; the quantity is the approximate solution to the linear system.

This statement is a little different than the analogous one in [70]; in that paper, the operator approximated $A^+$ rather than $A$. This is a minor difference, however, and the two viewpoints are equivalent. In our perspective, we only deal with the operator $\tilde{A}_\delta$ through its pseudoinverse.

## 2.8 A note on big-O notation

Before proceeding much further, we should clear up some of the notational inconsistencies in the literature. Throughout this thesis, $O(\cdot)$ will hide constant factors, $\tilde{O}(\cdot)$ will hide factors of $\log^{O(1)} n$ (i.e. polylogarithmic factors), and $\mathcal{O}(\cdot)$ will hide factors polynomial in $\log\log n$. The papers [41, 42] use $\tilde{O}(\cdot)$ where we use $\mathcal{O}(\cdot)$.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 3

# Spectral sparsification

We are now ready to introduce the concept of spectral sparsification, which plays a key role in this work. After reviewing the definitions and basic properties, we will show the algorithm due to Spielman and Srivastava for constructing a spectral sparsifier. We will provide a full analysis of this algorithm, which is somewhat different from the one presented in [69], but which resembles the argument in Srivastava's thesis [71]. We also propose and analyze an alternative algorithm, which is useful in our semi-streaming construction.

## 3.1 Definitions

Let $G$ be a graph and $H$ a weighted subgraph of $G$ (i.e. all edges of $H$ are edges of $G$, though the weights might be different).

**Definition 3.1.1.** With $G$ and $H$ as above, we say that $H$ is a $1 \pm \epsilon$ spectral sparsifier of $G$ if it is the case that

$$(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G \tag{3.1}$$

In other words, $H$ is a $1 \pm \epsilon$ sparsifier of $G$ if the Laplacians of $G$ and $H$ approximate each other well in the spectral sense.

Spectral sparsification is a very powerful notion of graph approximation. If $H$ is

a sparsifier of $G$, it approximately preserves numerous properties of $G$. For example, if $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $L_G$ and $\tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \cdots \leq \tilde{\lambda}_n$ are the eigenvalues of $L_H$, then it is the case that $(1-\epsilon)\lambda_i \leq \tilde{\lambda}_i \leq (1+\epsilon)\lambda_i$ (see, e.g. [69]).

Because values of cuts are given as quadratic forms of Laplacians evaluated at particular vectors (namely the characteristic vectors of one side of the cut), we see that a spectral sparsifier of $G$ preserves cut values to within a $1 \pm \epsilon$ factor. Cut-preserving sparsifiers (also known as combinatorial sparsifiers) were introduced by Benczúr and Karger [12] in the context of cut and flow problems, and our prior observation shows us that the notion of spectral sparsification is at least as strong as the notion of combinatorial sparsification. In fact, one can prove that it is strictly stronger. In particular, there exist graphs $G$ and cut-preserving sparsifiers $H$ such that the $H$ do not satisfy the properties of spectral sparsification.

By taking pseudoinverses, we also have the following inequality.

$$\frac{1}{1+\epsilon} L_G^+ \preceq L_H^+ \preceq \frac{1}{1-\epsilon} L_G^+ \tag{3.2}$$

Since effective resistances are given by the quadratic form defined by a Laplacian pseudoinverse evaluated at at certain vectors, it follows that $H$ preserves effective resistances between vertices in $G$ to within a $1/(1 \mp \epsilon)$ factor. This observation will prove crucial in our semi-streaming algorithm.

Of course, in order to be useful for speeding up approximation algorithms, we would like the graph $H$ to be as sparse as possible. Since $G$ can have up to $O(n^2)$ edges, an edge count that is $O(n \log^{O(1)} n)$ is considered fairly good. In fact, the algorithm due to Spielman and Srivastava, which we review and extend in this work, gives us a graph of $O(n \log n/\epsilon^2)$ edges. Batson, Spielman, and Srivastava [11] gave a deterministic polynomial-time algorithm for constructing a sparsifier with $O(n/\epsilon^2)$ edges. However, their algorithm, while still polynomial time, is significantly slower.

## 3.2 The Spielman-Srivastava algorithm

The first constructions of spectral sparsifiers were quite complicated. In 2008, Spielman and Srivastava gave a conceptually elegant procedure that also produced sparser results. Like the algorithm of Benczúr and Karger, theirs is based on a random sampling procedure, where we take edge samples with replacement and add them into the sparsifier with certain weights.

We sample edges with probabilities proportional to their "importance" in the graph. In particular, at least for an unweighted graph, one intuitive measure of the importance of an edge for problems related to cuts and connectivity is how likely that edge is to appear in a randomly generated spanning tree of the graph. As we saw before (Theorem 2.6.7), this quantity is given by to the edge's effective resistance.

In the Spielman-Srivastava algorithm, the probability of picking an edge $e$ at each step is proportional to its weight multiplied by its effective resistance $R^G(e)$. In other words, the probability of sampling an edge $e$ is given by $w_e R^G(e)/(n-1)$.

**Algorithm 1** Sparsify

**Input:** $G = (V, E, w)$
  Set $H$ to be the empty graph on $V$.
  **for** $i = 1$ to $N = O(n \log n/\epsilon^2)$ **do**
    Sample edge $e \in G$ with probability $p_e$ proportional to $w_e R^G(e)$ (i.e. $p_e = w_e R^G(e)/(n-1)$) and add it in with weight $w_e/(Np_e)$ to $H$
  **end for**

## 3.3 Analysis of the Spielman-Srivastava algorithm

The result we get on the Spielman-Srivastava algorithm is a little stronger than what the authors gave in their paper. This is because we use a stronger concentration of measure theorem [74, 32].

In order to simplify notation in the analysis, we will assume that $G$ is unweighted. It is straightforward to adapt the proof to the general case.

**Theorem 3.3.1.** *The Spielman-Srivastava sampling algorithm (Algorithm 1) produces a* $1 \pm \epsilon$ *sparsifier* $H$ *of* $G$ *with high probability.*

We do this by reducing the sparsification problem to that of approximating the projection matrix $I_{n-1} = L_G L_G^+$. Recall, as before, $I_{n-1}$ is the projection onto $\ker(L_G)^\perp$.

We know that $L_G = \sum_{e \in G} b_e b_e^T$. Multiplying both sides of the equation by $L_G^{-1/2}$ on the right and left, and letting $v_e = L_G^{-1/2} b_e$, we see that $\sum_{e \in G} v_e v_e^T = I_{n-1}$. The Spielman-Srivastava sparsification scheme is essentially a way of sampling the $v_e$ to get something close to the matrix $I_{n-1}$. This will in turn be enough, by the following lemma.

**Lemma 3.3.2.** *Suppose that we have* $\left\| \sum_{i=1}^M w_i v_{e_i} v_{e_i}^T - I_{n-1} \right\| \leq \epsilon$ *for some* $e_i$ *and weights* $w_i$. *(Here,* $\|\cdot\|$ *is the* $L^2$ *operator norm.) Then, taking the graph* $H$ *obtained by adding edges* $e_i$ *with weights* $w_i$ *gives us a* $1 \pm \epsilon$ *spectral sparsifier of* $G$.

*Proof.* We note that to prove that $H$ is a $1 \pm \epsilon$ sparsifier of $G$ it is enough to show the inequality (3.1) for all $x \in \ker(L_G)^\perp$, rather than all $x \in \mathbb{R}^n$. Indeed, writing a given $x \in \mathbb{R}^n$ as $x = y + z$ for $y \in \ker(L_G)^\perp$ and $z \in \ker(L_G)$, we will see that all terms involving $z$ will be 0.

For a matrix $A$, the condition $\|A - I_{n-1}\| \leq \epsilon$ means that $|x^T A x - x^T x| \leq \epsilon x^T x$ for all $x$. This amounts to saying that

$$(1-\epsilon)x^T x \leq x^T A x \leq (1+\epsilon) x^T x$$

for all $x$.

Now, for a given $y \in \ker(L_G)^\perp$, we can write it as $L_G^+ x$ for some $x$, and $y^T L_G y = x^T x$. Furthermore, if $A = \sum w_i v_{e_i} v_{e_i}^T$, we have $x^T A x = y^T L_G A L_G y$, which is equal to $y^T \left(\sum_i w_i b_{e_i} b_{e_i}^T\right) y$, which equals $y^T L_H y$ for $H$ as defined in the theorem statement. From this, it is easy to see that $(1-\epsilon)L_G \preceq L_H \preceq (1+\epsilon)L_G$, as required.

□

*Proof of Theorem 3.3.1.* For $i$ from 1 to $N$ let $e_i$ be an edge sampled with probability proportional to its effective resistance. We will show that

$$\left\| \frac{1}{N} \sum_{i=1}^N (1/p_{e_i}) v_{e_i} v_{e_i}^T - I_{n-1} \right\| \leq \epsilon \tag{3.3}$$

holds with high probability. If the inequality holds, then Lemma 3.3.2 it is easy to see that the graph $H$ obtained by adding in edges $e_i$ with weight $1/(Np_{e_i})$ is a $1 \pm \epsilon$ sparsifier of $G$.

Note that $\|v_e v_e^T\| = \|v_e\|^2 = b_e^T L_G^+ b_e = R^G(e)$. Write $v_e v_e^T = p_e \tau_e \tau_e^T$, where, as in the algorithm $p_e = R^G(e)/(n-1)$. Note that each $\tau_e$ has norm $\sqrt{n-1}$ (and hence $\tau_e \tau_e^T$ has operator norm $n-1$). Further, we see that if $\boldsymbol{p}$ is the probability distribution on edges assigning $e$ a probability of $p_e$ then the expected value of $\tau_e \tau_e^T$ with respect to $\boldsymbol{p}$ is $I_{n-1}$. Because $\tau_e \tau_e^T = (1/p_e) v_e v_e^T$, we see that the probability in (3.3) is exactly

$$\Pr\left[\left\|\frac{1}{N}\sum \tau_{e_i}\tau_{e_i}^T - I_{n-1}\right\| \leq \epsilon\right]. \tag{3.4}$$

The Ahlswede-Winter Theorem (a matrix version of the Chernoff bound) allows us to analyze the deviations from the expected value that we get when we independently sample. Using [74, Corollary 3] (also proved in [32]), we see that the probability in (3.3) is at least $1 - n\exp\left(\frac{-\epsilon^2 n}{4(n-1)}\right)$. The $n-1$ in the denominator of the exponential function comes from the fact that each $\tau_e \tau_e^T$ has operator norm at most $n-1$ (it is exactly $n-1$ in this case).

By taking $N = O(n \log n/\epsilon^2)$ samples, we can make the probability larger than $1 - n^{-d}$; we can get the claim for any given $d$ by taking a big enough constant in the expression for $N$.

Therefore, we see that by sampling the $\tau_e \tau_e^T$ with the appropriate distribution, we get an approximation to the identity with high probability. Noting that $\tau_e \tau_e^T = (1/p_e) v_e v_e^T$, we see that we get a $1 \pm \epsilon$ sparsifier with high probability. □

The argument in [69] also required showing that by estimating the effective resistances, one could still get a high quality sparsifier. This is easy to prove given our formulation.

**Corollary 3.3.3.** *Suppose that we have estimates $\tilde{R}_e$ for the effective resistances. Consider sampling edges with probability $\tilde{p}_e$ that is proportional to $\tilde{R}_e$, i.e. $\tilde{p}_e = \tilde{R}_e / \left(\sum_{f \in G} \tilde{R}_f\right)$. If all the $\tilde{p}_e$ are at least $(1/\alpha) p_e$, then by running the algorithm for*

*generating $H$ with $N$ replaced by $\alpha N$ we will have the same high probability guarantee on $H$ being a $1 \pm \epsilon$ sparsifier of $G$ as in the original algorithm.*

*Proof.* Write $v_e v_e^T = \tilde{p}_e \tilde{\tau}_e \tilde{\tau}_e^T$, where $\tilde{\tau}_e = \sqrt{p_e/\tilde{p}_e}\tau_e$, and therefore $\|\tilde{\tau}_e \tilde{\tau}_e^T\| \leq (n-1)p_e/\tilde{p}_e$, which is at most $\alpha(n-1)$. Then, analogously to the above proof, we see that the probability in (3.4) with the $\tilde{\tau}_{e_i}$ replacing the $\tau_{e_i}$ is now at least

$$1 - n\exp\left(\frac{\epsilon^2 N'}{4\alpha(n-1)}\right),$$

where $N'$ is the number of samples we take. Therefore, taking $N' = \alpha N$ gives us the desired high probability claim. □

Koutis, Miller, and Peng [41] had a slightly different perspective on sparsifying using approximate values for effective resistances. Their result is essentially the same as the corollary proved above. In fact, the proof we give simplifies the one in [41].

**Corollary 3.3.4.** *Given a graph $G$, let the $S_e$ for each edge $e$ be numbers such that $S_e \geq w_e R^G(e)$ for all $e$. Let $S = \sum_{e \in G} S_e$. Then, by taking $O(S \log n)$ samples in the Spielman-Srivastava sampling procedure we obtain a subgraph $G'$ of $G$ which, with high probability, satisfies $G \preceq 2G' \preceq 3G$.*

*Proof.* The probability with which we sample edge $e$ is $S_e/S$ which is at least $w_e R^G(e)/S$. The latter quantity is equal to $p_e/(S/(n-1))$. Therefore, by Corollary 3.3.3, taking

$$(S/(n-1))O(n \log n) = O(S \log n)$$

samples is enough to get the approximation guarantee with high probability. (We remark that the desired approximation guarantee in the theorem has a constant $\epsilon$, which we absorb into the $O(\cdot)$ notation for the number of samples.) □

Note, in particular, that if $S_e = w_e R^G(e)$, then the above proposition tells us to take $O(S \log n)$ edges, where $S = \sum_{e \in G} S_e = n - 1$. This matches the result that we need $O(n \log n)$ edges to get a sparsifier when we use the correct effective resistances to compute the sampling probabilities.

## 3.4 A new algorithm for spectral sparsification

Instead of sampling edges with replacement, we can also run a procedure where we consider each edge, and accept or reject it with a certain probability. In this case, the number of edges in our sparsifier will be a random variable (though one tightly concentrated around its mean).

Consider the following algorithm:

**Algorithm 2** Alternative sparsify

**Input:** $G$

**Output:** $H$, a $1 \pm \epsilon$ sparsifier of $G$ (with high probability)

**for all** edges $e$ of $G$ **do**

  **for** $i$ from 1 to $N = O(n \log^2 n/\epsilon^2)$ **do**   *%Run this loop implicitly*

    With probability $p_e = w_e R^G(e)/(n-1)$ add $e$ to $H$ with weight $w_e/(Np_e)$

  **end for**

**end for**

Naïvely, it looks like the sampling should take $O(mN)$ time. However, we run the inner loop implicitly: the number of times we pick edge $e$ follows a binomial distribution, and we can sample this quickly to drastically reduce the running time.

Indeed, the probability that we get $j$ copies of edge $e$ is $\beta_j := \binom{N}{j} p_e^j (1-p_e)^{N-j}$. Therefore, to sample the inner loop for a given edge $e$, we pick a uniformly random $r$ from $[0,1]$. We compute $\beta_0$, and if $r \leq \beta_0$, then we do not include $e$ in the sparsifier. Otherwise, we generate $\beta_1, \beta_2$, etc. in turn, until we have found a $j$ such that $\sum_{k=0}^{j-1} \beta_k \leq r < \sum_{k=0}^{j} \beta_k$; we add $j$ copies of $e$ to $H$. For a particular edge $e$, the total running time is $O(1)$ (to generate $r$, compute $\beta_0$, and compare the two) plus a constant times the (random) number of samples of $e$ we take.

Thus, because the total number of samples we take throughout the algorithm is $O(N)$ with high probability, we have:

**Proposition 3.4.1.** *The total running time of Algorithm 2 (if we know all the $p_e$) is $O(m+N) = O(m)$.*

We know that $O(m+N) = O(m)$ in the regime of interest because sparsification

only makes sense when the number of edges in the sparsifier, $O(N)$, is much smaller than that in the original graph.

Now we are ready to prove the algorithm's correctness.

**Theorem 3.4.2.** *The algorithm above produces a $1 \pm \epsilon$ sparsifier of $G$ with high probability. Further, with high probability, the number of edges in the resulting graph is $O(N) = O(n \log^2 n/\epsilon^2)$.*

Again, to simplify notation we assume that $G$ is unweighted.

*Proof.* For $i$ from 1 to $N$, let $X_{i,e}$ be the random variable that is $\tau_e \tau_e^T = (1/p_e) v_e v_e^T$ with probability $p_e$ and 0 otherwise. Let $A_{i,e}$ be the random variable that is 1 if $X_{i,e}$ is nonzero, and 0 otherwise. Then, $E[X_{i,e}] = v_e v_e^T$. Further, for a given $i$, we know that $E\left[\sum_{e \in G} A_{i,e}\right]$, the expected number of nonzero $X_{i,e}$ (over all the $e$ in $G$), is exactly $\sum_{e \in G} p_e = 1$, and, by a Chernoff bound, is at most $O(\log n)$ with high probability.

Let $Y_i = \sum_{e \in G} X_{i,e}$. We have that $E[Y_i] = \sum_e E[X_{i,e}] = I_{n-1}$. Further, by the triangle inequality, we have that $\|Y_i\|$ is at most $n - 1$ times the number of nonzero $X_{i,e}$ (since the $\tau_e \tau_e^T$ all have norm $n - 1$), and is thus at most $O(n \log n)$ with high probability. Let $Y = (1/N) \sum_{i=1}^N Y_i$. Then $E[Y] = I_{n-1}$.

We claim that

$$\Pr[\|Y - I_{n-1}\| > \epsilon] < ne^{-O\left(\frac{\epsilon^2 N}{n \log n}\right)} \tag{3.5}$$

This is once again proved by matrix-valued Chernoff bounds, but the proof is a little more complicated since $\|Y_i\|$ can be large. However, this happens rarely, and does not affect the overall result.

Consider the auxiliary random variable $\tilde{Y}_i$, which is the same as $Y_i$ when at most $O(\log n)$ of the $X_{i,e}$ (over $e \in G$) are nonzero, and 0 otherwise. We will prove that the expected value of $\tilde{Y}_i$ is very close to $I_{n-1}$. Further, since $\|\tilde{Y}_i\| = O(n \log n)$, we will be able to apply matrix Chernoff bounds to prove that $\sum_{i=1}^N \tilde{Y}_i$ is tightly concentrated around its expectation, hence to $NI_{n-1}$ (since $E[\tilde{Y}_i] \approx E[Y_i]$). Finally, because $Y_i$ and $\tilde{Y}_i$ coincide overwhelmingly often, we will be able give a strong bound the probability of $\sum_i Y_i$ deviating from its expectation.

We know that $Y_i = \tilde{Y}_i + (Y_i - \tilde{Y}_i)$, hence

$$E[Y_i] = E[\tilde{Y}_i] + E[Y_i - \tilde{Y}_i | Y_i = \tilde{Y}_i] \Pr[Y_i = \tilde{Y}_i] + E[\tilde{Y}_i - Y_i | \tilde{Y}_i \neq Y_i] \Pr[\tilde{Y}_i \neq Y_i].$$

The second term is 0, and the third term is $E[Y_i | \tilde{Y}_i \neq Y_i] \Pr[\tilde{Y}_i \neq Y_i]$ since the only time that $Y_i$ can differ from $\tilde{Y}_i$ is when $\tilde{Y}_i$ is 0.

Thus, we have $I_{n-1} = E[Y_i] = E[\tilde{Y}_i] + \Pr[Y_i \neq \tilde{Y}_i] E[Y_i | \tilde{Y}_i \neq Y_i]$. Now, each draw of $Y_i$ has norm at most $O(n^3)$, since there are at most $O(n^2)$ of the $\tau_e \tau_e^T$, each of norm $n-1$. Therefore, the norm of $\Pr[Y_i \neq \tilde{Y}_i] E[Y_i | \tilde{Y}_i \neq Y_i]$ can be made smaller than $1/n^d$ for some $d$, since the probability $\Pr[Y_i \neq \tilde{Y}_i]$ can be made small.

Therefore, $\left\| I_{n-1} - E[\tilde{Y}_i] \right\| \leq 1/n^d$.

Now, $E\left[\sum_i \tilde{Y}_i\right] = NE[\tilde{Y}_i]$, and

$$\Pr\left[ \left\| \sum_i \tilde{Y}_i - NE[\tilde{Y}_i] \right\| > \epsilon N \|E[\tilde{Y}_i]\| \right] \leq e^{-O\left(\frac{\epsilon^2 N \|E[\tilde{Y}_i]\|}{n \log n}\right)}. \tag{3.6}$$

This follows by matrix Chernoff bounds, specifically [74, Corollary 3]. We need the $n \log n$ in the denominator of the exponent because $O(n \log n)$ is an upper bound on the norm of the $\tilde{Y}_i$.

Furthermore,

$$\left\| \sum_i \tilde{Y}_i - NI_{n-1} \right\| \leq \left\| \sum \tilde{Y}_i - NE[\tilde{Y}_i] \right\| + N \|E[\tilde{Y}_i] - I_{n-1}\|.$$

Because $E[\tilde{Y}_i]$ is very close to $I_{n-1}$, the second quantity is tiny; with appropriate constants it can be made smaller than any inverse polynomial in $n$. It is certainly smaller than $\epsilon N/2$. Therefore,

$$\Pr\left[ \left\| \sum \tilde{Y}_i - NI_{n-1} \right\| > \epsilon N \right] \leq \Pr\left[ \left\| \sum \tilde{Y}_i - NE[\tilde{Y}_i] \right\| > \epsilon N/2 \right],$$

which is equal to $\Pr\left[ \left\| \sum \tilde{Y}_i - NE[\tilde{Y}_i] \right\| > (\epsilon/2\|E[\tilde{Y}_i]\|) N \|E[\tilde{Y}_i]\| \right]$. Using (3.6), we see

that this quantity is is at most

$$ne^{-O\left(\frac{\epsilon^2 N}{\|E[\tilde{Y}_i]\| n \log n}\right)} = ne^{-O\left(\frac{\epsilon^2 N}{n \log n}\right)}$$

(recall that $\|E[\tilde{Y}_i]\|$ is very close to 1, so we can absorb its contribution into the $O(\cdot)$ notation). For $N = O(n \log^2 n/\epsilon^2)$ as in the algorithm, this quantity is very small.

Let $\mathcal{A}$ be the event that $\tilde{Y}_i = Y_i$ for all $i$, which occurs with high probability. Then, for any event $\mathcal{B}$, it is the case that $\Pr[\mathcal{B}] = \Pr[\mathcal{B}, \mathcal{A}] + \Pr[\mathcal{B}, \bar{\mathcal{A}}] \leq \Pr[\mathcal{B}, \mathcal{A}] + \Pr[\bar{\mathcal{A}}]$. In particular,

$$\begin{aligned}
\Pr\left[\left\|\sum Y_i - NI_{n-1}\right\| > \epsilon N\right] &\leq \Pr\left[\left\|\sum Y_i - NI_{n-1}\right\| > \epsilon N, \mathcal{A}\right] + \Pr[\bar{\mathcal{A}}] \\
&= \Pr\left[\left\|\sum \tilde{Y}_i - NI_{n-1}\right\| > \epsilon N, \mathcal{A}\right] + \Pr[\bar{\mathcal{A}}] \\
&\leq \Pr\left[\left\|\sum \tilde{Y}_i - NI_{n-1}\right\| > \epsilon N\right] + \Pr[\bar{\mathcal{A}}]
\end{aligned}$$

(The second line follows because when $\mathcal{A}$ is true, $Y_i = \tilde{Y}_i$, by definition.)

The final quantity is very small for $N$ as above. This shows that $Y$ is tightly concentrated around its expectations.

Because

$$Y = \frac{1}{N}\left(\sum_{e_i}(1/p_{e_i})v_{e_i}v_{e_i}^T\right),$$

where the $e_i$ are edges that were selected (with multiplicities), we see (again by Lemma 3.3.2) that Algorithm 2 puts in edges correctly to give a $1 \pm \epsilon$ sparsifier with high probability.

To prove that the size of the sparsifier is $O(N)$ with high probability, we consider the $A_{i,e}$. Then, the size of the sparsifier is $\sum_{i=1}^{N}\sum_{e\in G} A_{i,e}$, whose expectation is $N$. Further, it is a sum of independent $\{0,1\}$-valued random variables, hence by a Chernoff bound, it is tightly concentrated around its expected value.

□

Notice that our sparsifiers require $O(\log n)$ more edges than the Spielman-Srivastava sparsifiers. This is because of the stochasticity in the number of edges we select at

each $i$. This quantity is 1 in expectation, and is at most $O(\log n)$ with high probability.

By the same argument as the one used to prove Corollary 3.3.3, we see that if we use estimates $\tilde{p}_e = \tilde{R}_e / \left(\sum_{f \in G} \tilde{R}_f\right)$ such that $\tilde{p}_e \geq (1/\alpha) p_e$ for all $e$, then we will need to increase $N$ by a factor of $\alpha$ in the algorithm to match the high probability guarantee of the algorithm that makes use of the correct effective resistances.

The result is even more flexible. For example, if we can guarantee that $p_e/\alpha \leq \tilde{p}_e \leq \alpha p_e$, for some small $\alpha \geq 1$ (it will be the case that $\alpha \ll 2$ in our setting) then by sampling using the probabilities $\tilde{p}_e$ we still get a $1 \pm \epsilon$ sparsifier with high probability. Indeed, the number of nonzero $X_{i,e}$ is still $O(\log n)$, and the norm of the $\tau_e \tau_e^T$ increases by at most an $\alpha$ factor, as in Corollary 3.3.3. We can thus carry the proof through almost unchanged. As before, we can increase $N$ by a constant factor to overcome the increased error probability caused by our estimates.

Interestingly, the result holds even when $\sum_{e \in G} \tilde{p}_e$ is something other than 1. The added flexibility that the method offers is useful to our analysis.

## 3.5 Computing effective resistances

The algorithms we give above rely on knowing the effective resistances, as these give us the required probabilities. The effective resistances can be computed exactly by first computing $L_G^+$, and then using the formula $R^G(i,j) = (\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j)$.

Unfortunately, while doing this is polynomial time, it is not efficient enough in certain contexts. For example, many algorithms that are sped up by sparsifiers, including Benczúr and Karger's original application, proceed in two stages. First, we compute a sparsifier of the input graph, and then we run the original algorithm on the sparsifier. For this two-stage process to give a speedup, the sparsification algorithm has to be extremely efficient (and in particular, it should take less time to sparsify the input graph than to run the original algorithm on it).

As we noted, it is enough to use good estimates of the effective resistances, rather than the exact quantities. This suggests a simple speedup: instead of using $L_G^+$, we could approximately solve linear systems in $L_G$ with one of the fast solvers. This

is not fast enough, however, since we still need to compute $m$ quantities, each one requiring an application of the solver.

An ingenious trick in the paper allows for estimating *all* the effective resistances very quickly. First we can obtain a different expression for the effective resistance, via a simple algebraic manipulation:

$$\begin{aligned} R^G(i,j) &= (\chi_i - \chi_j)^T L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ L L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ B^T W^{1/2} W^{1/2} B L^+ (\chi_i - \chi_j) \\ &= \|W^{1/2} B L^+ (\chi_i - \chi_j)\|^2 \end{aligned}$$

The advantage of this definition is that it expresses the effective resistance as the squared Euclidean distance of two points, given by the $i$th and $j$th column of the matrix $W^{1/2}BL^+$.

This new expression still involves the solution of a linear system in $L$. The natural idea is to replace $L$ with an approximation $\tilde{L}$ satisfying the properties described in Lemma 2.7.2. So instead of $R^G(i,j)$ we compute the quantities $\hat{R}^G(i,j) = \|W^{1/2}B\tilde{L}_\delta^+(\chi_i - \chi_j)\|^2$.

How big does the $\delta$ have to be in order to give us a constant approximation guarantee (i.e. to approximate the quantities to within a constant factor)? In the original construction, Spielman and Srivastava took it to be inverse polynomial in $n$. In our work [40], we showed that in fact a constant $\delta$ is good enough. This directly improves the running time of the procedure by a logarithmic factor.

**Lemma 3.5.1.** *For a given $\eta$, if $\tilde{L}$ satisfies $(1-\delta)L \preceq \tilde{L} \preceq (1+\delta)L$ where $\delta = \eta/8$, then the approximate effective resistance values $\hat{R}^G(u,v) = \|W^{1/2}B\tilde{L}^+(\chi_u - \chi_v)\|^2$ satisfy:*

$$(1-\eta)R^G(u,v) \leq \hat{R}^G(u,v) \leq (1+\eta)R^G(u,v).$$

*Proof.* We only show the first half of the inequality, as the other half follows similarly.

Since $L$ and $\tilde{L}$ have the same null space, by (3.2) the given condition is equivalent to:

$$\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+ \preceq \frac{1}{1-\delta}L.$$

Since $\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+$, we have

$$\begin{aligned} R^G(u,v) &= (\chi_u - \chi_v)^T L^+ (\chi_u - \chi_v) \\ &\leq (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ \tilde{L} \tilde{L}^+ (\chi_u - \chi_v). \end{aligned}$$

Applying the fact that $\tilde{L} \preceq (1+\delta)L$ to the vector $\tilde{L}^+(\chi_u - \chi_v)$ in turn gives:

$$\begin{aligned} R^G(u,v) &\leq (1+\delta)^2 (\chi_u - \chi_v)^T \tilde{L}^+ L \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)^2 \|W^{1/2} B \tilde{L}^+ (\chi_u - \chi_v)\|^2 = \hat{R}^G(u,v) \end{aligned}$$

The rest of the proof follows from $\frac{1}{(1+\delta)^2} \leq 1 - \eta/4$ by choice of $\delta$. □

Of course, even though we can now quickly solve the linear systems using an approximate solver, there are still $m$ systems to be solved. To work around this hurdle, Spielman and Srivastava observe that projecting the vectors to an $O(\log n)$-dimensional space preserves the Euclidean distances within a constant factor (with high probability), by the Johnson- Lindenstrauss theorem. Algebraically this amounts to computing the quantities $\|QW^{1/2}B\tilde{L}_\delta^+(\chi_i - \chi_j)\|^2$, where $Q$ is a properly defined random matrix of dimension $k \times m$ for $k = O(\log n)$. The authors invoke the result of Achlioptas [2], which states that one can use a matrix $Q$ each of whose entries is randomly chosen in $\{\pm 1/\sqrt{k}\}$.

Since each $\|W^{1/2}BL_\delta^+(\chi_i - \chi_j)\|^2$ is within a constant factor of $R^G(i,j)$, and multiplying by $Q$ preserves the lengths of all of these vectors up to a constant factor (with high probability), we see that the $\|QW^{1/2}BL_\delta^+(\chi_i - \chi_j)\|^2$ are also within a constant factor of $R^G(i,j)$.

The construction of the sparsifiers can can thus be broken up into three steps.

1. Compute $QW^{1/2}B$. This takes time $O(km)$, since $B$ has only two non-zero entries per row.

2. Apply the linear operator $\tilde{L}_\delta^+$ to the $k$ columns of the matrix $(QW^{1/2}B)^T$, using Lemma 2.7.2. This gives the matrix $Z = QW^{1/2}B\tilde{L}_\delta^+$. The running time of this operation is $\mathcal{O}(m \log^2 n \log(1/\delta))$. As we showed before, we can take $\delta$ to be constant, so the running time is $\mathcal{O}(m \log^2 n)$.

3. Compute all the (approximate) effective resistances (time $O(km)$) via the square norm of the differences between columns of the matrix $Z$. Then sample the edges (in time $O(m + n \log^2 n/\epsilon^2)$).

Throughout, we will consider running the sparsification algorithm on graphs that are large enough so that the $O(n \log^2 n/\epsilon^2)$ in the sampling time is dominated by the other factors in the running time of the algorithm. Indeed, for it to make sense to sparsify a graph in the first place, the graph should be sufficiently dense (namely $m$ should be big enough relative to $n$, e.g. $m > n \log^2 n/\epsilon^2$). For such graphs, we have proved:

**Theorem 3.5.2.** *Let $G$ be a graph that is sufficiently dense. We can find a $1 \pm \epsilon$ sparsifier $H$ of $G$ in time $O(m \log^2 n)$.*

## 3.6 A super-approximation property (optional)

Let us once again consider the Spielman-Srivastava algorithm run with the exact effective resistances. We know that if $G$ is a graph and $H$ is its sparsifier produced by that algorithm, then $H$ approximates certain graph quantities to within a $1 \pm \epsilon$ factor.

One could expect to do better than the $1 \pm \epsilon$ worst-case approximation guarantee in certain instances. We consider a particularly interesting example. For unweighted and connected $G$, the sum of the effective resistances of all the edges is $n - 1$. Let us consider calculating the sum of the effective resistances of edges of $G$ using $H$. In

other words, what is $\sum_{e \in G} b_e^T L_H^+ b_e$? Of course, this is merely a theoretical exercise; since we know the correct answer, there is no algorithmic gain to performing this calculation using the graph $H$. However, it is a natural question to try to analyze how this invariant behaves when we use the graph approximation $H$.

We have

$$\begin{aligned}\sum_{e \in G} b_e^T L_H^+ b_e &= \sum_{e \in G} (b_e^T L_H^{-1/2})(L_H^{-1/2} b_e) \\ &= \sum_{e \in G} \operatorname{Tr}\left(L_H^{-1/2} b_e b_e^T L_H^{-1/2}\right) \\ &= \operatorname{Tr}\left(L_H^{-1/2} L_G L_H^{-1/2}\right)\end{aligned}$$

We will study this trace by considering the trace of the inverse matrix. Consider

$$\operatorname{Tr}\left(L_H^{1/2} L_G^+ L_H^{1/2}\right) = \operatorname{Tr}\left(L_G^{-1/2} L_H L_G^{-1/2}\right). \tag{3.7}$$

We know that $L_H$ is the sum of $N = O(n \log n/\epsilon^2)$ terms of the form $w_e b_e b_e^T$ for $e \in G$ and $w_e = \frac{n-1}{N R^G(e)}$. Thus, the trace on the right hand side of (3.7) consists of a sum of $N$ terms of the form $w_e b_e^T L_G^+ b_e$, so each term is $(n-1)/N$, meaning that the trace is $n-1$. Moreover, if $H$ is a $1 \pm \epsilon$ sparsifier of $G$, then we know that each eigenvalue of $L_G^{-1/2} L_H L_G^{-1/2}$ is of the form $\lambda_i = 1 + \mu_i$ where $|\mu_i| \leq O(\epsilon)$. Now $\operatorname{Tr}\left(L_G^{-1/2} L_H L_G^{-1/2}\right) = \sum \lambda_i = n - 1$, hence $\sum \mu_i = 0$. The inverse, $L_G^{1/2} L_H^+ L_G^{1/2}$ has eigenvalues $1/\lambda_i = 1 - \mu_i + O(\epsilon^2)$. Therefore, $\operatorname{Tr}(L_G^{1/2} L_H^+ L_G^{1/2}) = (n-1) - \sum \mu_i + O((n-1)\epsilon^2)$. The sum of the $\mu_i$ is zero, so this is equal to $(n-1)(1 \pm O(\epsilon^2))$.

Therefore, for the sum of effective resistances, the distortion caused by evaluating using $H$ is substantially smaller than what we would naively expect ($1 \pm O(\epsilon^2)$ vs. $1 \pm O(\epsilon)$).

It would be interesting to find other examples where better-than-expected approximation properties hold, or to prove that certain randomized constructions yield them. What can we say, for example, about sums of quadratic forms of $L_G$ versus $L_H$ evaluated at random vectors?

There are potential algorithmic applications if such properties can be shown to hold. For example, if we have an algorithm that only needs to evaluate quantities which satisfy certain super-approximation properties, then we can argue that we will need a worse-quality (and faster-to-construct) sparsifier than one would naively assume.

Finding quantities for which we have super-approximation properties, and seeing if we can leverage this to construct faster algorithms, is thus an intriguing avenue of investigation.

# Chapter 4

# More background

This chapter presents more advanced background material. Most of the content consists of primitives that are important for the solvers of Koutis, Miller, and Peng [41, 42].

## 4.1 Primitives for the Koutis-Miller-Peng solver

### 4.1.1 Low-stretch spanning trees

Let $G$ be a graph, and let $T$ be a subgraph of $G$ that is a spanning tree of $G$. Consider some edge $e$ of $G$. Then, there is a unique path $P_e$ in $T$ that joins the endpoints of $e$.

The *stretch* of an edge $e \in G$ is defined in reference to this path. It is given by:

$$\mathrm{st}_T(e) = w_e \left( \sum_{f \in P_e} w_f^{-1} \right) \tag{4.1}$$

Let us consider, for example, the case of unweighted graphs. Then, the stretch of edge $e$ is given by the length of the path in $T$ joining the endpoints of $e$.

Notice that the quantity $\sum_{f \in P_e} w_f^{-1}$ is just the effective resistance in $T$ between the endpoints of edge $e$. Indeed, $w_f^{-1}$ is the resistivity of the edge $f$, and since $T$ is a tree, the effective resistance between any two points is just the sum of resistivities on the path between the two points (in the physical model, we can think of resistors in series).

The *total stretch* of $G$ through $T$, which we write as $\mathrm{st}_T(G)$, is given by the sum of the stretches of each edge of $G$. In other words,

$$\mathrm{st}_T(G) = \sum_{e \in G} \mathrm{st}_T(e).$$

**Proposition 4.1.1.** *Given a graph $G$ and a tree $T$, the total stretch $\mathrm{st}_T(G)$ is given by $\mathrm{Tr}(L_T^+ L_G)$.*

*Proof.* We have

$$\begin{aligned}
\mathrm{Tr}(L_T^+ L_G) &= \mathrm{Tr}\left(L_T^{-1/2} L_G L_T^{-1/2}\right) \\
&= \sum_{e \in G} w_e \,\mathrm{Tr}\left(L_T^{-1/2} b_e b_e^T L_T^{-1/2}\right) \\
&= \sum_{e \in G} w_e b_e^T L_T^+ b_e \\
&= \sum_{e \in G} w_e R^T(e) \\
&= \sum_{e \in G} \mathrm{st}_T(G)
\end{aligned}$$

□

It is an important problem to design spanning trees with low average stretch. The best algorithm is the one by Abraham, Bartal, and Nieman [1], as improved by Koutis, Miller, and Peng [42]. We summarize the statement in the following theorem:

**Theorem 4.1.2.** *Given a graph $G$ with $n$ vertices and $m$ edges, there is a spanning tree $T$ of $G$ such that $\mathrm{st}_T(G) = \mathcal{O}(m \log n)$. Moreover, this $T$ can be found by an algorithm running in time $\mathcal{O}(m \log n)$.*

### 4.1.2 Incremental sparsifiers

In [41], Koutis, Miller, and Peng asked whether they could get anything useful out of the Spielman-Srivastava construction, without having to rely on linear system solvers. Given a tree spanning tree $T$ of $G$, we know that for any edge $e$, it is the case that

$\mathrm{st}_T(e) \geq w_e R^G(e)$. Since it is easy to compute the stretches of all the edges in $O(m)$ time (using an offline lowest common ancestor algorithm [31]), it is possible to sample with probabilities proportional to $\mathrm{st}_T(e)$. One can use the results of Corollary 3.3.4 to determine how many edges to take in order to get a high-quality approximation of $G$.

Of course, such an approximation would not be of any use unless the number of edges was significantly smaller than $m$. For this, Koutis, Miller, and Peng take the tree $T$ to be a low-stretch spanning tree of $G$. The intuition is that the lower we get $\mathrm{st}_T(G)$ the fewer samples we need to take, since the sum of the overestimates of the probabilities defines the number of samples. Unfortunately, taking the low stretch spanning tree does not yield a small enough number of samples, so instead, the authors consider the graph $G'$ that is the same as $G$ except that the weights of the edges of $T$ are scaled up by a suitably chosen factor $\kappa$.

We then run the procedure described above on $G'$. We lose a factor of $\kappa$ in the approximation guarantee. However, by scaling up tree weights we in fact lowered the total stretch by a $\kappa$ factor, and, for suitable $\kappa$, we will have have a small enough edge count in the graph we output. We call this graph the *incremental sparsifier.* For the analysis, it also turns out to be useful to exercise care and count multiple copies of the same edge as only one edge; this helps us get an edge reduction even for sparse graphs, which is required for preconditioning very sparse systems of equations.

We summarize the result below; for details, see [41]. The $\kappa$ factor is the factor by which we increase the weights of the tree edges. In Koutis, Miller, and Peng's paper [41], they set $\kappa$ to be $O(\log^4 n)$.

**Theorem 4.1.3.** *Let $G$ be a graph. Then, there is an algorithm for constructing a subgraph $K$ of $G$ such that, given $\kappa$:*

- $G \preceq K \preceq 3\kappa G$
- *$K$ has $n - 1 + \mathcal{O}((m/\kappa)\log^2 n)$ edges*

*The algorithm succeeds with high probability and runs in time*

$$\mathcal{O}(m \log n + m \log^3 n/\kappa).$$

The $\mathcal{O}(m \log n)$ in the running time comes from the computation of the low-stretch spanning tree, and the computation of the stretches. The $\mathcal{O}(m \log^3 n/\kappa)$ is the time required to sample.

If the tree is given, then we only need to compute the stretches (which takes $O(m)$ time [41]) and sample. This gives a running time of $\mathcal{O}(m + m \log^3 n/\kappa)$.

Note that the incremental sparsifiers that we construct have a significantly higher distortion than spectral sparsifiers, and also have more edges (at least for large $m$). This is the tradeoff we have to accept in order to avoid using linear system solvers. Despite this tradeoff, the incremental sparsifiers make good preconditioners for solving linear systems.

Interestingly, incremental sparsifiers have also played a role in our algorithms for speeding up spectral sparsification; see [46] and Section 5.5.

## 4.2 Spine-heavy graphs

Koutis, Miller, and Peng [42] show that it is often useful to consider graphs that have spanning trees of extremely low stretch.

**Definition 4.2.1.** Let $G$ be a graph with $n$ vertices and $m$ edges. We say that it is *spine-heavy* if it has a low-stretch tree of stretch at most $O(m/\log n)$.

In other words, the stretch of this tree is an $\mathcal{O}(\log^2 n)$ factor better than the worst-case guarantee of Theorem 4.1.2.

Among other improvements to the results of [41], the article [42] showed that systems of equations in spine-heavy graphs can be very efficiently solved, provided we have the low-stretch spanning tree available. In particular:

**Theorem 4.2.2.** *Let $G$ be a spine-heavy graph and let $T$ be its tree such that* $\mathrm{st}_T(G) = O(m/\log n)$. *Then, given $T$, we can solve linear systems in $L_G$ to precision $\delta$ in time* $\mathcal{O}(m \log(1/\delta))$.

The stronger statement of Lemma 2.7.2 applies here as well: we can consider the solver as defining a linear map that spectrally approximates $L_G^+$.

Note that the approximation to $G$ we get in the previous sections, where we take a low-stretch spanning tree (e.g. output by the algorithm of Theorem 4.1.2) and scale up the edge weights by a factor of $\kappa = \mathcal{O}(\log^2 n)$, is a spine-heavy graph.

**Proposition 4.2.3.** *There is an $\mathcal{O}(m \log n)$ algorithm that produces an approximation $G'$ of a graph $G$ and a tree $T'$ in $G'$ such that:*

- *$G'$ is spine-heavy, and namely* $\mathrm{st}_{T'}(G') = O(m/\log n)$.
- $G \preceq G' \preceq \mathcal{O}(\log^2 n) G$.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 5

# Improved spectral sparsification

This chapter presents some of our improvements on the spectral sparsification algorithm of Spielman and Srivastava. The work here arose out of a simple question: can we trade off the running time of the sparsification algorithm for the size of the sparsifier we output? In other words, can we find an algorithm that is faster than Spielman and Srivastava's, but that potentially produces a sparsifier with a higher edge count? Such tradeoffs and improvements were considered in the case of cut-preserving sparsifiers in a recent paper by Fung *et al.* [27]. We thus sought to produce similar results for spectral sparsifiers.

Given that solving linear systems in order to estimate the effective resistances is the bottleneck of our algorithm, we looked for ways of speeding up that step. In fact, we found a simple way of solving linear systems on a modified graph, which introduced some distortions into our estimates of the effective resistances. In order to overcome these distortions, we were, in turn, required to take more samples to produce our sparsifier. In short, this exactly gave us the tradeoff we sought.

Pushing these ideas further, we were able to substantially speed up constructions of spectral sparsifiers almost to linear time. The construction is fast enough to allow us to use the resulting sparsifier as a preconditioner for solving linear systems in $L_G$, giving an improved running time for that problem. Another theoretical consequence is that ours is the first construction of Spielman-Srivastava type sparsifiers that is useful for linear system solving. (The situation is not quite as satisfying as we would

like, as we still need to use linear systems, albeit more specialized ones that are fast to solve, in the construction of the sparsifier.)

All of these results are very much in the original spirit of sparsification. Specifically, our results show that for the applications cited above, we get a running time savings by first creating a sparsifier. Given that a sparsifier is a very strong approximation of a graph, the fact that we can construct it quickly enough to speed up already-fast algorithms is indeed very surprising. This was also the case in Benczúr and Karger's original paper on cut problems [12], where the key idea of speeding up the algorithm was to (very quickly) construct a sparsifier first.

## 5.1 Overview of our results

### 5.1.1 The importance of transitivity

In several of our results, we will make use of the transitivity property of the spectral sparsification construction. If we have a $1 \pm \epsilon_1$ sparsifier $H_1$ of a graph $G$ and we run use the Spielman-Srivastava algorithm to produce a $1 \pm \epsilon_2$ sparsifier of $H_1$, the result is clearly a $(1 \pm \epsilon_1)(1 \pm \epsilon_2)$ sparsifier of $G$ (with high probability).

In our applications, $H_1$ will be a graph of "intermediate" size: the number of edges will be significantly smaller than $m = |E(G)|$ but bigger than $O(n \log n/\epsilon^2)$. If we have a fast algorithm for producing $H_1$ (as will be the case in the instances where we apply the construction), then we can subsequently sparsify $H_1$ down to a graph of $O(n \log n/\epsilon^2)$ edges, and moreover, if $|E(H_1)|$ is much smaller than $m$, this step will take less than $O(m)$ time. In many important parameter regimes this two-step process gives us a faster algorithm for spectral sparsification.

### 5.1.2 The $\mathcal{O}(m \log n)$ algorithm

The original algorithm of Spielman and Srivastava already includes a number of ingenious techniques that make it run very fast. In order to speed up the algorithm further, we need to break the central bottleneck, which comes from having to solve

$O(\log n)$ linear systems each of which takes $\mathcal{O}(m \log n)$ time. We improve the running time of this step by allowing for cruder, but more easily-computable, approximations of the effective resistances. It was shown in [41] that if we estimate the effective resistances, the Spielman-Srivastava scheme still goes through, but we may need to sample more edges to compensate for the loss of accuracy.

In particular, we estimate the effective resistances by using a *spine-heavy* approximation to $G$. This is a graph that has an extremely good low stretch spanning tree. In [42] it was shown that linear equations in Laplacians of spine-heavy graphs can be solved to precision $\delta$ in $\mathcal{O}(m \log(1/\delta))$ time (see Theorem 4.2.2). Further any graph can be easily transformed into a spine-heavy approximation while distorting the effective resistances by at most an $\mathcal{O}(\log^2 n)$ factor. Using this spine-heavy approximation in order to quickly estimate effective resistances, and then sampling with respect to these estimates, allows us to get a sparsifier with $\mathcal{O}(n \log^3 n/\epsilon^2)$ edges in $\mathcal{O}(m \log n)$ time. The details are given in Section 5.2.

### 5.1.3 The $\mathcal{O}(m)$ algorithm

Several more obstacles needs to be circumvented for an even faster algorithm. Even assuming a computationally free SDD solver, estimating the effective resistances via the Johnson-Lindenstrauss projection requires operating on $m$ vectors of dimension $O(\log n)$, which is too expensive. This forces us to try to decrease (hopefully down to a constant) the dimension of the projections. Of course this introduces higher distortions in the estimates for the effective resistances, but as we noted above the algorithm can compensate by taking more samples. The second key to our result comes into play here: transitivity. We observe that it is enough to produce a sparsifier with $m' = O(m/\log^2 n)$ edges since we can then run our original sparsification algorithm in time $\mathcal{O}(m' \log^2 n) = \mathcal{O}(m)$ and get the final sparsifier. This trick allows us to reduce the dimension of the JL projection to a constant, for large enough $m$. The details are given in Section 5.3.

However to get these severely distorted estimates for the effective resistances, it is not enough to just take our $\mathcal{O}(m \log n)$ algorithm and replace the Johnson-

Lindenstrauss projection by a constant-dimensional one. The remaining bottleneck is the running time of the solver; its construction requires at the minimum the computation of a low-stretch tree which takes $\mathcal{O}(m \log n)$ time [1]. The solver steps after the construction of the low-stretch tree take $\mathcal{O}(m)$ time on a spine-heavy graph. This implies that we would be able to sparsify in $\mathcal{O}(m)$ time if the computation of the low-stretch tree were not an issue.

To solve this problem, we show that every graph can be decomposed into graphs of diameter $O(\log n)$ with relatively few edges between the pieces. Spanning trees with $O(\log n)$ average stretch can be easily computed for each of these pieces, and thus we sparsify them separately and then put the results together. The details are given in Section 5.4.

## 5.2 The $\mathcal{O}(m \log n)$ algorithm

We have seen in Corollary 3.3.3 that if we use estimates to the effective resistances, rather than the true values, the Spielman-Srivastava scheme still works, but in order to produce the sparsifier we have to compensate by taking more samples. Specifically, for $\alpha > 1$, if the probabilities with which we sample all edges are at least $1/\alpha$ of the true values, then we have to take $\alpha$ times as many samples. An equivalent way of expressing this is the following lemma:

**Lemma 5.2.1.** *Suppose that we run the Spielman-Srivastava algorithm and sample edges with probabilities proportional to $\tilde{q}_e$ such that*

$$\frac{w_e R^G(e)}{\alpha} \leq \tilde{q}_e \leq w_e R^G(e)$$

*for all edges $e$. Then, taking $\alpha$ times as many samples gets us a $1 \pm \epsilon$ sparsifier with the same high probability guarantee as the Spielman-Srivastava algorithm run with probabilities proportional to $w_e R^G(e)$.*

Indeed, it is not hard to see that the bounds on $\tilde{q}_e$ imply that the probability with which we sample edge $e$ is at least $p_e/\alpha$ for all $e$.

We are now ready to state our first theorem.

**Theorem 5.2.2.** *There is a* $1 \pm \epsilon$ *sparsification algorithm for graphs with* $m > n \log^3 n$ *edges that runs in time* $\mathcal{O}(m \log n)$. *The output sparsifier contains* $\mathcal{O}(n \log^3 n/\epsilon^2)$ *edges.*

*Proof.* Given the input graph $G$ we construct a spine-heavy graph $H$ satisfying the properties of Proposition 4.2.3. The construction can be done in time $\mathcal{O}(m \log n)$. Then, we have

$$\frac{1}{\mathcal{O}(\log^2 n)} R^G(i,j) \leq R^H(i,j) \leq R^G(i,j).$$

We run the procedure for estimating effective resistances (Section 3.5) on $H$ to approximate the effective resistances $R^H(i,j)$ within a constant factor. Step 2 of the process runs in $\mathcal{O}(m \log n)$ time on $H$, by Lemma 2.7.2, since each linear system takes $\mathcal{O}(m)$ time.

The output is a set of estimates that are correct to within a constant factor with high probability. To make these estimates conform to the statement of Lemma 5.2.1, we divide them by a constant factor, which does not change the sampling probabilities. Then, the calculated approximate effective resistances, $\hat{R}^H(i,j)$ satisfy

$$\frac{1}{\mathcal{O}(\log^2 n)} R^G(i,j) \leq \hat{R}^H(i,j) \leq R^G(i,j).$$

Finally we let $\tilde{q}_e = w_e \hat{R}^H(i,j)$ for all edges $e = (i,j)$ of $G$ and sample the edges of $G$ with probabilities proportional to $\tilde{q}_e$. By Lemma 5.2.1 with $\alpha = \mathcal{O}(\log^2 n)$ we see that we get a $1 \pm \epsilon$ sparsifier with $\mathcal{O}(n \log^3 n/\epsilon^2)$ edges. □

We should note that in the first version of this work [46], we used an incremental sparsifier to estimate the effective resistances instead of a spine-heavy approximation as above. An incremental sparsifier of a graph $G$ has poly-logarithmically fewer edges than $G$, and approximates it to within a poly-logarithmic factor. This, in turn, allows us to quickly compute the required approximations to the effective resistances, and then proceed as in the algorithm above. Subsequent improvements in the linear

system solver [42], and in particular the observation that systems arising from spine-heavy graphs can be solved extremely quickly, allowed us to come up with the simpler version we presented here.

## 5.3 Effective resistances via very low-dimensional projections

With the improvement of the last section, all three steps of the Spielman-Srivastava algorithm take $\mathcal{O}(m \log n)$ time; our goal now is to reduce this to $\mathcal{O}(m)$. The extra logarithm in the current implementation is due to the dimension $k = O(\log n)$ of the projection matrix $Q$, and we address this issue here.

It is worth noting that once we have a sparsifier $H$ with $O(\frac{m}{\log^2 n})$ edges such that

$$\left(1 - \frac{\epsilon}{3}\right) G \preceq H \preceq \left(1 + \frac{\epsilon}{3}\right) G,$$

we can afford to fully $(1 \pm \frac{\epsilon}{3})$-sparsify that $H$ using our $\mathcal{O}(m \log^2 n)$ algorithm. The sparsifier of $H$ (with $O(n \log n/\epsilon^2)$ edges) will then be a $1 \pm \epsilon$-sparsifier for $G$.

Since we can take more samples, we are able to underestimate probabilities more aggressively by decreasing the dimension we project onto, and still get a good approximation to $G$ with high probability. In order to show that we do not underestimate effective resistances by too much, we need a more detailed understanding of the relationship between the dimension $k$ and the approximation guarantee. This is provided by the version of the Johnson-Lindenstrauss theorem stated as Lemma 7 of [34]:

**Lemma 5.3.1.** *Let $u$ be a unit vector in $\mathbb{R}^\nu$. For any given positive integers $k$, let $U_1, \ldots, U_k$ be random vectors chosen independently from the $\nu$-dimensional Gaussian distribution, which we call $N^\nu(0,1)$. For $X_i = u^T U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = \|W\|^2$. Then for any $\beta > 1$:*

1. $E(L) = k$,

2. $\Pr[L \geq \beta k] < O(k) \exp(-\frac{k}{2}(\beta - (1 + \ln \beta)))$,

*3.* $\Pr[L \leq k/\beta] < O(k) \exp(-\frac{k}{2}(\beta^{-1} - (1 - \ln \beta)))$.

This basically quantifies the distortions in the length of a vector when projecting it on random vectors, as is done in standard analysis of the Johnson-Lindenstrauss theorem. (In the lemma, the vector is normalized to have unit length; the expected square length of a unit vector is $k$, but we can make it 1 by dividing the matrix entries by $1/\sqrt{k}$.) Using this viewpoint we see that this lemma essentially gives us the probability of increasing or decreasing sizes of a given vector by a certain factor when we multiply the vector by a random matrix of Gaussian entries.[1] Roughly, the third part states that for a given small constant $r \ll 1$, the probability of underestimating distances (and hence effective resistances in our application) by an $n^r$ factor is around $O(n^{-rk/2})$. By setting $k$ sufficiently large and applying a union bound, we obtain that with high probability all estimates are at least $\Omega(n^{-r})$ of the true quantities required by the Spielman-Srivastava algorithm.

Combining this with the fact that weight times effective resistance is upper bounded by 1, one can show by concentration of measure theorems that the normalizing factor (i.e. the weighted sum of the estimated effective resistances) stays within a constant factor of its true value with high probability. Therefore, with high probability we underestimate the edge selection probabilities by at most a factor of $O(n^r)$. The number of samples we need to take as a result is $n^{1+r} \log n$. As long as this is smaller than $m/\log^2 n$ we can sparsify in $\mathcal{O}(m)$ time. This shows that as long as $m$ is big enough relative to $n$, we can sparsify in linear time, as we claimed in the introduction.

We formalize this argument below. In fact, we integrate the results of the previous subsection, where we sped up the linear system solving by using a spine-heavy approximation.

**Lemma 5.3.2.** *There is an algorithm that, on input a graph $G$ with $n$ vertices, $m$ edges, a low-stretch spanning tree for $G$ with total stretch $\mathcal{O}(m \log n)$, and a parameter $t$, generates a $1 \pm \epsilon$ sparsifier with $\mathcal{O}(\frac{m}{t} \log n/\epsilon^2)$ edges in $\mathcal{O}(m \log_{\frac{m}{3nt \log^2 n}} n)$ time.*

[1]This is a minor difference from previous parts, where we use matrices with entries randomly chosen in $\pm 1/\sqrt{k}$

*Proof.* We first construct in $O(m)$ time the spine-heavy graph $G'$ that $\mathcal{O}(\log^2 n)$-approximates $G$ (i.e. such that $G \preceq G' \preceq O(\log^2 n)G$). We then apply the Spielman-Srivastava procedure in order to estimate the effective resistances in $G'$.

Invoking Part 3 of Lemma 5.3.1 with $\beta = \frac{m}{nt\log^2 n}$ shows us that when we project onto $k$ dimensions, the probability of underestimating by a factor of $\beta$ is at most:

$$O(k)\exp\left(\frac{k}{2}(1-\beta^{-1}-\ln\beta)\right) \leq O(k)\exp\left(\frac{k}{2}(1-\ln\beta)\right) \leq O(k)(3/\beta)^{\frac{k}{2}}$$

where the first inequality follows from $k/2 \geq 0$ and $1-\beta^{-1} \leq 1$. So when $(3/\beta)^{\frac{k}{2}} = n^{-d}$, taking a union bound over all $m \leq n^2$ edges gives that no edge's effective resistance is underestimated by more than a factor of $\beta$. The requirement on $k$ imposed by this is:

$$O(k)(3/\beta)^{\frac{k}{2}} \leq n^{-d}$$
$$k \geq 2d\log_{\beta/3} n + \log_{\beta/3} k + O(1)$$

Setting $d$ to be some constant and taking the value of $\beta$ as before we see that taking $k = O(\log_{\frac{m}{3nt\log^2 n}} n)$ will give us the required high probability claims.

This shows that projecting in order to estimate effective resistances and using these to estimate edge selection probabilities will give us values that are at least an $nt\log^2 n/m$ factor of the true value (for $\beta$ as above). Following the proof of Lemma 3.5.1 we can see that using an approximate solver introduces a small multiplicative error. Using the fact that $G'$ is a graph that $O(\log^2 n)$-approximates $G$, we see that this method produces approximates probabilities in $G$ that are at least a factor of $\frac{nt}{m}$ of the true values.

Consider sampling with these estimated probabilities. Then, by the discussion at the beginning of Section 5.2 with $\alpha = m/(nt)$, we see that to sparsify we need to take $O(\frac{m}{t}\log n\epsilon^{-2})$ samples.

The running time of this process is dominated by amount of time it takes to do $k$ solves in $L_{G'}$, namely $O(km)$ by Lemma 2.7.2. For the choice of $k$ as before this is $\mathcal{O}(m \log_{\frac{m}{3nt \log^2 n}} n)$, as required.

□

**Theorem 5.3.3.** *Given a graph $G$ with $n$ vertices, $m$ edges such that $m > n \log^5 n$, and a low-stretch spanning tree with stretch $\mathcal{O}(m \log n)$, we can generate a $1 \pm \epsilon$-sparsifier $H$ of $G$ with $O(n \log n/\epsilon^2)$ edges in $\mathcal{O}(m \log_{\frac{m\epsilon^2}{n \log^5 n}} n)$ time.*

*Proof.* Applying Lemma 5.3.2 with $t = O(\log^3 n/\epsilon^2)$ gives a graph with $\mathcal{O}(\frac{m}{\log^2 n})$ edges that is a $1 \pm \epsilon/3$-sparsifier. This graph can in turn be $1 \pm \epsilon/3$ sparsified in $\mathcal{O}(\frac{m}{\log^2 n} \log^2 n) = \mathcal{O}(m)$ time, by Theorem 5.2.2. □

## 5.4 Improved sparsification via graph decompositions

Theorem 5.3.3 reveals that the computation of the low-stretch tree of the input graph is the final bottleneck on our way to getting the faster algorithms. In order to solve this problem, we no longer compute a low-stretch spanning tree for the entire graph. Instead, we decompose the graph into subgraphs for which we can trivially find low-stretch spanning trees and we sparsify each subgraph separately. The decomposition is based on the following simple fact about low diameter graphs:

**Lemma 5.4.1.** *Given an unweighted graph with $n$ vertices, $m$ edges, and diameter $O(\log n)$, finding a breadth-first search (BFS) tree in $O(m)$ time gives low stretch spanning tree with average stretch $O(\log n)$.*

*Proof.* It takes $O(m)$ time to construct the BFS tree. Suppose that $i$ is the vertex we start at. Because the graph has diameter $O(\log n)$, each vertex $j$ will be at a distance of $O(\log n)$ from $i$, and by properties of BFS trees, there will be a path of length $O(\log n)$ to $i$ using tree edges. From here, it is clear that the endpoints of any edge in $G$ can be connected by a path of length $O(\log n)$ in the tree, hence the claim about stretch follows. □

We can now apply low diameter decomposition to extend this to arbitrary undirected graphs losing an extra factor of $\log \log n$. The variant of low diameter decomposition that we use can be best described using the following lemma (see, e.g., [72, Lemma 4]).

**Lemma 5.4.2.** *Given an undirected, unweighted graph with $n$ vertices and $m$ edges, we can partition it into pieces of $O(\log n)$ diameter so that at most $m/2$ edges are between the pieces. This process can be performed in $O(m)$ time.*

Applying this $O(\log \log n)$ times and sparsifying the edges between pieces each time gives the claim for arbitrary unweighted graphs:

**Theorem 5.4.3.** *Given an undirected, unweighted graph $G$ with $n$ vertices and $m$ edges such that $m > \Omega(n \log^5 n)$, we can output a sparsifier $H$ with $\mathcal{O}(n \log n/\epsilon^2)$ edges in $\mathcal{O}(m \log_{\frac{m\epsilon^2}{3n \log^5 n}} n)$ time.*

*Proof.* We create $G_1, \ldots G_l$ where $l = 4 \log \log n$ as follows. Given $G_1, \ldots, G_i$, we partition $E(G) \setminus E(G_1) \ldots \setminus E(G_i)$ into low diameter pieces using Lemma 5.4.2 and let $G_{i+1}$ be edges with both endpoints in the same piece that's not in some $G_j$ with $j \leq i$. Applying guarantees of Lemma 5.4.2 inductively gives $|E(G_i)| \leq 2^{-i}|E(G)| = 2^{-i}m$, and specifically $|E(G_l)| \leq \frac{m}{\log^2 n}$. Therefore $G_l$ can be sparsified to $H_l$ via the original Spielman-Srivastava sparsification algorithm algorithm in time $\mathcal{O}(m)$.

We now turn our attention to $G_1, \ldots, G_{l-1}$. If a particular $G_i$ contains fewer than $O(m/\log^2 n)$ edges, it can be left unsparsified (such $G_i$ will contribute a sufficiently small number of edges, and we will take care of this at the end). Otherwise, since a low-stretch tree can be obtained trivially, we can sparsify it by means of Lemma 5.3.2. Concretely, by letting $t = \log^3 n/\epsilon^2$ and using the same $\beta$ as in the proof of that Lemma, we get graphs $H_1, \ldots, H_{l-1}$ (the $1 \pm \epsilon$-sparsifiers of the corresponding $G_i$) such that

$$(1-\epsilon)G_i \preceq H_i \preceq (1+\epsilon)G_i,$$

in total time $\mathcal{O}(m \log_{\frac{m\epsilon^2}{n \log^5 n}} n)$. Letting $\hat{H} = H_l + \sum_{i<l} H_i$ gives a sparsifier with $\mathcal{O}(\frac{m}{\log^2 n})$ edges, which can in turn be sparsified in $\mathcal{O}(m)$ time to generate $H$ with $O(n \log n/\epsilon^2)$ edges. ◻

For weighted graphs, with polynomially bounded edge weights, we partition edges by weight into buckets and sparsify each subgraph. More concretely, let $G_i$ be the subgraph of $G$ consisting of edges whose weights are in the interval $[(1+\epsilon)^i w_{\min}, (1+\epsilon)^{i+1} w_{\min}]$, with the weight of each edge rounded down to $(1+\epsilon)^i w_{\min}$. (Here, $w_{\min}$ is the minimum weight of an edge in the graph.) Since the edge weights are assumed to be polynomially bounded, we have only $O(\log n/\epsilon)$ of the $G_i$. Furthermore, note that $\sum G_i \preceq G \preceq (1+\epsilon)\sum G_i$.

Since each $G_i$ is a multiple of an unweighted graph, we can sparsify it down to a graph $\hat{G}_i$ of $O(m\epsilon/\log^3 n)$ edges using the techniques above. Summing up all the $\hat{G}_i$ gives us a graph with $O(m/\log^2 n)$ edges, which we can sparsify using the Spielman-Srivastava algorithm in $O(m)$ time. This gives us a $1 \pm O(\epsilon)$ approximation.

## 5.5 Getting over the Johnson-Lindenstrauss Barrier

Thus far, we have made substantial progress on improving the running time of the spectral sparsification algorithm, and pushing it as far as we can towards linear time. Unfortunately, because of the constraints of Section 5.3, our best running time guarantee holds only for $m > n^{1+r}$ for a small constant $r$. We would like to strengthen our result, and eliminate this constraint to the greatest extent possible. In particular, we will be happy if we can get an algorithm with $\mathcal{O}(m)$ running time in the regime $m > n\log^{O(1)} n$.

It turns out that we can do this, but we need to set aside our modifications of the Johnson-Lindenstrauss step, and do something completely different.

We consider an *unweighted* graph $G$. Assume that a low-stretch tree of $G$ is given (or computable in $O(m)$ time; this happens if $G$ is of $O(\log n)$ diameter).

Let us first construct an incremental sparsifier $H$ of $G$ with $O(m/\log n)$ edges. We have $G \preceq H \preceq O(\log^3 n)G$. Then, we can construct a sparsifier $K$ of $H$ that gives a constant relative condition number and that has $O(n \log n)$ edges. After scaling

weights of edges in $K$ by constant factors as necessary, we have $G \preceq K \preceq O(\log^3 n)G$.

Furthermore, since $K$ has $O(n \log n)$ edges, it has a spanning $T$ tree with total stretch $\text{st}_T(K) = O(n \log^2 n)$ by Theorem 4.1.2.

Let $\{w_e\}_{e \in K}$ represent the weights of the edges of $K$. For vertices $i$ and $j$, let $P_{i,j}$ be the path between them in $T$. Define the *pseudo-stretch* between $i$ and $j$ through $T$ as follows:

$$\text{pst}_T(i,j) = \sum_{f \in P_{i,j}} w_f^{-1} \tag{5.1}$$

This definition is very similar to that of the stretch, except it is missing a term for the weight of the edge between $i$ and $j$. Indeed, $(i,j)$ might not be an edge of $K$, so it would not make sense to have a weight.

It is the case that $\text{pst}_T(i,j) \geq R^K(i,j)$.

This, together with the fact that $O(\log^3 n)R^K(i,j) \geq R^G(i,j)$ means that

$$O(\log^3 n)\,\text{pst}_T(i,j) \geq R^G(i,j).$$

Therefore, as in Proposition 3.3.4 we can sample edges $e$ of $G$ with probability proportional to $s_e = O(\log^3 n)\,\text{pst}_T(e)$, where, if $e = (i,j)$ we let $\text{pst}_T(e) := \text{pst}_T(i,j)$. To figure out the number of samples we need to take, we sum these quantities over all the edges $e$ of $G$. Note that $\sum_{e \in G} \text{pst}_T(e)$ is equal to $\text{Tr}(L_T^+ L_G)$ by the same argument used to prove Proposition 4.1.1.

Indeed,

$$\begin{aligned}
\sum_{e \in G} \text{pst}_T(e) &= \sum_{e \in G} R^T(e) \\
&= \sum_{e \in G} b_e^T L_T^+ b_e \\
&= \sum_{e \in G} \text{Tr}(L_T^+ b_e b_e^T) \\
&= \text{Tr}(L_T^+ L_G)
\end{aligned}$$

The equation in the first line follows because $\text{pst}_T(e)$ is just the effective resistance in $T$ between the endpoints of $e$, being the sum of resistivities. The last line follows

because $L_G = \sum_{e \in G} b_e b_e^T$, where we use the fact that $G$ is an unweighted graph.

Now, by Proposition 2.1.5 the quantity $\mathrm{Tr}(L_T^+ L_G)$ is at most $\mathrm{Tr}(L_T^+ L_K)$ since $G \preceq K$, and therefore is less than or equal to $O(n \log^2 n)$. Therefore, the sum of the $s_e$ is $O(\log^5 n)$, which means that we need to take $O(n \log^6 n/\epsilon^2)$ samples in order to get a $1 \pm \epsilon$ sparsifier with high probability.

**Theorem 5.5.1.** *There is an algorithm running in time $\tilde{\mathcal{O}}(m)$ that, given a graph $G$ with $\Omega(n \log^3 n)$ edges and its low-stretch spanning tree produces a $1 \pm \epsilon$ sparsifier $H$ of $G$ with $O(n \log^6 n/\epsilon^2)$ edges.*

*Proof.* We have proved everything except the running time claim. To see that, note that the incremental sparsifier can be constructed in $\tilde{\mathcal{O}}(m)$ time, and the running times of the other operations are at most $\tilde{\mathcal{O}}(m)$. □

We can extend this to weighted case, as well as to the case where we do not have a handy low stretch spanning tree, using the graph decomposition and weight bucketing tricks of Section 5.4. We do not include the full details here, but refer the reader to our paper [39], where we further refine this technique.

## 5.6 Applications

### 5.6.1 Linear system solving

The $\tilde{\mathcal{O}}(m)$ algorithm for graph sparsification immediately allows us to use the output as a preconditioner for solving linear systems. Specifically, if $G$ is a graph and we want to solve linear systems of the form $L_G x = b$, then we construct a sparsifier $H$ of $G$ in $\tilde{\mathcal{O}}(m)$ time as above. We can then use $L_H$ as a preconditioner for $L_G$ with constant relative condition number. While $L_G$ is not invertible, the system will have a solution provided $b \in \ker(L_G)^\perp$; for $G$ connected, this just means that the sum of the entries of $b$ is 0.

Consider the preconditioned system $L_H^+ L_G x = L_H^+ b$.[2] Then, since the condition number of $L_H^+ L_G$ is constant, we need only take constantly many iterations. In each

[2]Technically, this is not quite the system we need to consider, since we must make the appropriate

iteration, we need to multiply $L_G$ by a vector ($O(m)$ time), and then apply $L_H^+$ to the result, which we do by solving a linear system in $L_H$ to constant precision. By considering the solver as an efficiently-computable linear operator, we see that this procedure is equivalent to applying an operator $\tilde{L}_H^+$ instead of $L_H^+$, where the relative condition number of $\tilde{L}_H$ and $L_H$ is constant, and hence $\tilde{L}_H^+ L_G$ has constant condition number. Since $H$ has $O(n \log n)$ edges, approximately solving equations in $L_H$, or equivalently, evaluating the map $\tilde{L}_H^+$, takes $O(n \log^2 n)$ time. Therefore, taking $\log(1/\delta)$ many iterations of this gives $O(m + n \log^2 n) \log(1/\delta) = O(m \log(1/\delta))$ time. It follows that the total amount of time to solve, including the construction of the preconditioner, is $\mathcal{O}(m \log(1/\delta))$, as desired.

It is also worth noting that even our simpler $\mathcal{O}(m \log n)$ running time sparsification algorithm gives a moderate running time advantage for solving linear systems. Constructing the sparsifier with $O(n \log n)$ edges takes $\mathcal{O}(m \log n)$ time. Then, solving the preconditioned system as before takes $O(m + n \log^2 n) \log(1/\delta)$ time. Assuming that $n \log^2 n < m$, we see that the total running time is $\mathcal{O}(m \log n + m \log(1/\delta))$, rather than $\mathcal{O}(m \log n \log(1/\delta))$, and thus the method gives a slight running time advantage. This advantage is especially visible when $\delta$ is small (e.g. inverse polynomial in $n$), where it effectively shaves a $O(\log n)$ factor from the running time.

### 5.6.2 Approximate Fiedler vectors

We now show how we can use our techniques to give the fastest known algorithm for computing approximate Fiedler vectors.

As before, let $G$ be a connected graph with Laplacian $L_G$, and let $\lambda_2$ be the second lowest eigenvalue of $L_G$. Then

$$\lambda_2 = \min_{v \in \ker(L_G)^\perp} \frac{v^T L_G v}{v^T v}.$$

An approximate Fiedler vector is a vector in $\ker(L_G)^\perp$ that gets a value for the

matrix symmetric and positive definite. Instead, when solving the system $Ax = b$ with preconditioner $B$, we actually solve the system $(B^{-1/2} A B^{-1/2}) B^{1/2} x = B^{-1/2} b$ for $B^{1/2} x$. However, for purposes of exposition, we will ignore those issues.

quadratic form that is close to $\lambda_2$. More formally:

**Definition 5.6.1.** Let $G$ be a graph, and let $\lambda_2$ be the second eigenvalue. Then, a vector $v \in \ker(L_G)^\perp$ is an $\epsilon$-approximate Fiedler vector if $v^T L_G v / v^T v \leq (1+\epsilon)\lambda_2$.

Approximate Fiedler vectors play an important role in numerous algorithms. For example, graph partitioning by Fiedler vectors works if we supply an approximate one instead, which is what is done in practice.

There is a very natural algorithm based on the power method for producing approximate Fiedler vectors. Notice that $\lambda_2^{-1}$ is the biggest eigenvalue of $L_G^+$. Further, writing a vector $x \in \ker(L_G)^\perp$ in the eigenbasis for $L_G$, e.g. $x = x_2 u_2 + \cdots + x_n u_n$, we see that

$$(L_G^+)^r x = \lambda_2^{-r} x_2 u_2 + \cdots + \lambda_n^{-r} x_n u_n,$$

so the first term in the sum will dominate.

The procedure is further analyzed by Spielman and Teng, and algorithmically, it is run by using the solver to simulate multiplication by $L_G^+$.

Their result, [70, Theorem 6.2], depends on the speed of the linear system solver, which they do not explicitly give. Using the running time of [42], we can recast it as:

**Theorem 5.6.2.** *There is an algorithm that, on input a graph $G$, an approximation guarantee $\epsilon$, and a positive constant $p$, computes an approximate Fiedler vector of $G$ with probability $1 - 1/p$. The running time of the algorithm is*

$$\mathcal{O}(m \log^2 n \log(1/p) \log(1/\epsilon)/\epsilon).$$

For the running time claim, the algorithm calls the solver $O(\log n \log(1/p)/\epsilon)$ times and solves to precision $\epsilon$.

In our algorithm, we first obtain a $1 \pm \epsilon$ sparsifier of $G$. Notice that an $\epsilon$-approximate Fiedler vector of $H$ will give an $O(\epsilon)$-approximate Fiedler vector of $G$. We will then find an approximate Fiedler vector, using Spielman and Teng's method.

Interestingly, it will be fast enough to first compute the sparsifier, and then run the approximate Fiedler vector procedure on it. Even our $\mathcal{O}(m \log n)$ sparsification

algorithm gives an improved running time for the computation. For concreteness, the running time when using our $\mathcal{O}(m \log n)$ sparsification algorithm before computing the approximate Fiedler vector is

$$\mathcal{O}(m \log n + n \log^5 n \log(1/p) \log(1/\epsilon)/\epsilon^3).$$

The first part of the sum is to compute the sparsifier, and the second part is to compute an approximate Fiedler vector of the $O(n \log^3 n/\epsilon^2)$-edge sparsifier. This is an improvement over the original algorithm as long as $m > O(n \log^3 n/\epsilon^2)$. Of course, using the more advanced $\mathcal{O}(m)$ running time methods for sparsification that we introduced earlier can give us an even faster algorithms for this problem.

# Chapter 6

# Spectral sparsification in the semi-streaming setting

As we noted in the introduction, spectral sparsification produces a sparse approximation of a graph, which we can then use to say useful things about the original graph. However, being able to manipulate the original dense graph in order to construct the sparsifier in the first place is perhaps an unreasonable assumption. We would like to reduce the space requirement of the Spielman-Srivastava procedure, and in particular give an algorithm that works in the *semi-streaming* setting. In this setting, the amount of space we get is $\tilde{O}(n)$, which is comparable to the size of the final output. We think of receiving the graph as a stream of edges: at each step, we get to see an edge of the graph.

Our work gives a conceptually simple algorithm for producing a sparsifier, which works in the semi-streaming setting and takes only one pass over the edges of $G$. The latter statement means that once we see an edge and decide what to do with it, we never need to see it again.

In our analysis, we will consider a slightly more general setting, where we start with a graph $G$ and its sparsifier $H$, and, as we keep adding edges to $G$, we want to maintain a $1 \pm \epsilon$ approximation to the current graph. (Setting the initial graphs $G$ and $H$ to be empty graphs on the vertex set $V$, we get the original problem of sparsification in the semi-streaming setting). It is not hard to see that as we add

edges to $G$, by adding in those same edges to $H$, we get the desired approximation of $G$. Unfortunately, as we keep doing this, our sparsifier will contain increasingly many edges and may eventually become too large. Thus we will need to resample, to produce a sparsifier of smaller size. We show how to periodically do this resampling very fast, leading to amortized poly-logarithmic update time per edge added to $G$. More importantly, the resampling requires us to know only $H$ and the additional edges, without having to know all of $G'$. The resampling algorithm relies on two main insights:

1. As we add new edges to $G$ to produce a graph $G'$, the effective resistances of the edges of $G$ do not increase, and thus, neither does their probability of being selected for a sparsifier. Thus, if we can compute their new probabilities, we can rejection sample the edges in $H$ and also appropriately sample the new edges to produce edges selected with the probability distributions from $G'$, and hence a sparsifier of $G'$. Thus, we need not consider all the probabilities in $G'$, but only those of edges in $H$ and the added edges.

2. Since $H$ with the new edges well-approximates $G'$, we can use it to quickly estimate the effective resistances for the edges we need; this estimate turns out to be good enough.

On a high level, the key idea of our construction is that the original sparsifier already contains a great deal of information, which we can reuse to save time instead of building a sparsifier from scratch.

The problem of updating a sparsifier of a growing graph was what originally brought us to this field. We asked whether it would be possible for the update to take time that is nearly linear in the number of edges we add, rather than the total number of edges in the graph. The latter running time could be achieved by sparsifying the updated graph from scratch, and is thus not algorithmically interesting. The fact that our algorithm was in fact very space efficient was a nice side effect, and an important one, as we later realized.

## Related work

The problem of graph sparsification in the semi-streaming setting was introduced by Ahn and Guha [5], and it was then further studied by Goel, Kapralov, and Khanna [29] (the latter of which is concurrent to and independent of the present paper). Ahn and Guha constructed combinatorial sparsifiers in the semi-streaming model. However, while the space complexity of their algorithm was $\tilde{O}(n)$, the running time was $\tilde{O}(mn)$, which is often too slow when the graphs are large. This is remedied by the present work, as well as by Goel, Kapralov, and Khanna, who obtain results that are similar to ours when one aims to construct combinatorial sparsifiers.

However, the graphs that we produce obey the strictly stronger constraints imposed by spectral sparsification. To our knowledge, ours is the first work to do this in the semi-streaming setting.

Furthermore, we believe that our algorithm is conceptually cleaner and simpler than that of [29], and our techniques are quite different from theirs. The algorithm set forth by Goel *et al.* inherently requires a logarithmic number of passes through the data, and they maintain a multi-level collection of graphs and partitions of graphs. Then, using an ingenious construction and careful analysis, they find a way to implement this in a single pass. This results in a graph that has logarithmically more edges than necessary, which they then clean up at the end.

Our algorithm, on the other hand, operates inherently in a single pass. We simply add edges to our graph until it becomes large. When this occurs, we replace our graph with a sparser version still preserving the approximation guarantee and continue. By taking advantage of the stronger notion of sparsification that we are employing, and properly sparsifying and analyzing the probabilities, we are able to show that this simple algorithm produces the desired sparsifiers while requiring a poly-logarithmic amount of amortized work per edge and maintaining at all times a graph with $\tilde{O}(n/\epsilon^2)$ edges.

The algorithm we propose is, however, several log factors slower than that of [29]. In addition, like the algorithm of Spielman and Srivastava, it is not completely self-

contained, as it crucially relies on the (non-elementary) fast solvers for symmetric diagonally dominant linear systems (e.g. [42]).

## 6.1 Notation and conventions

Before proceeding, we make a few remarks about notation. Let $G$ be a graph with $n$ vertices. Let $\Gamma$ be another graph on the same vertex set as $G$. Then, $G + \Gamma$ is the graph given by adding the weights of the edges of $\Gamma$ to the corresponding edges of $G$.

In this chapter, for the most part $G$ and $\Gamma$ will be unweighted graphs, and $\Gamma$ will be edge-disjoint from $G$. In this case, $G+\Gamma$ represents the graph we get when we add the edges of $\Gamma$ to $G$. The definition agrees with the previous one if we regard missing edges as having a weight of 0, and those that are in the graph as having a weight of 1.

For an edge $e$ not in $G$, we denote $G + e$ the graph obtained by adding $e$ to $G$.

As noted before, the notation $\tilde{O}(\cdot)$ hides factors of $\log^{O(1)} n$.

## 6.2 The update algorithm

We are now ready to present the main part of our work, where we show how to continually maintain a sparsifier of a growing graph. Throughout, for notational convenience, we will consider the setting of adding new edges to an unweighted graph $G$ without adding new vertices. It is straightforward to generalize to the case where we add vertices, or where the graph is weighted and we may increase the weights of existing edges as well as add new ones, provided that the weights are polynomially bounded.

### 6.2.1 Setup

Initially, we assume that we are given access to the exact effective resistances when we need them to sample. We will later relax this requirement.

Suppose that $G$ is a graph on $n$ vertices and $H$ is a $1 \pm \epsilon$ sparsifier of $G$ generated by Algorithm 2. For conceptual convenience, we assume that if the sampling process of Algorithm 2 adds several copies of a given edge into $H$, it adds them as parallel edges; we will sometimes refer to edges of $H$ as *samples*, as they are indeed random samples output by the algorithm. The number of samples we put into $H$ is tightly concentrated around $N$.

Let $e$ be an edge not in $G$; then it is clear that $H + e$ is a $1 \pm \epsilon$ sparsifier of $G + e$. Indeed, we have

$$L_{G+e} = L_G + b_e b_e^T, \qquad L_{H+e} = L_H + b_e b_e^T,$$

whence the desired statement follows.

As we add edges to $G$, we can add those same edges to $H$, until the sparsifier gets too large, forcing us to resample. In this work, we say that this happens when it is of size $CN$ for some constant $C$ that we can choose at will.

We will formalize this situation as follows. Let $G = (V, E)$ be a graph, and let $H$ be its $1 \pm \epsilon$ sparsifier with around $N$ edges. Let $\Gamma$ represent the added edges (i.e., it is a graph on $V$ with edges exactly those that are added to $G$) such that $H + \Gamma$ has $CN$ edges. (Note that $H + \Gamma$ is a $1 \pm \epsilon$ sparsifier of $G' := G + \Gamma$.)

Because $H + \Gamma$ is large, we want to construct a sparsifier $H'$ of $G'$ such that $H'$ has around $N$ edges (i.e. we want to reduce the size of the sparsifier of $G'$ by some constant factor). We call this procedure *resparsification*. We would like this resparsification to take much less time than it would take to sample from scratch, namely $\tilde{O}(m/\epsilon^2)$. Sparsifying $G'$ from scratch gives us an average update time of $\tilde{O}(m/n)$ per operation, which is $\tilde{O}(n)$ when $G'$ is dense. We want a $\tilde{O}(1)$ amortized time instead. The key insight is to use the information already contained in $H$, which will allow us to sample edges from the correct distribution in time $\tilde{O}(n/\epsilon^2)$, leading to the desired bound.

The main observation is that when we add a new edge to $G$, the effective resistances of the other edges cannot increase, as we proved in Proposition 2.5.3. Thus, since effective resistances are given by evaluating the quadratic form defined by Laplacian pseudoinverses at particular vectors, we see that indeed they cannot increase.

Further, the sum of the effective resistances of all of the edges cannot decrease. (If adding the edge reduces the number of connected components, this quantity increases, otherwise it stays the same.) Thus, the probabilities of choosing the edges in the sparsification procedure cannot increase.

In what follows, we let $R_e = R^G(e)$ and $R'_e = R^{G'}(e)$ for notational simplicity. Let $p_e$ (resp. $p'_e$) be the probabilities of selecting that edge (i.e. $p_e = R_e / \sum_{f \in G} R_f$, and $p'_e = R'_e / \sum_{f \in G'} R'_f$). We will denote the collections of the $p_e$ and $p'_e$ by $\boldsymbol{p}$ and $\boldsymbol{p}'$ respectively.

The prior observations make it easy to sample according to the probabilities $\boldsymbol{p}'$ *only having to consider edges in $H$ and $\Gamma$!* Indeed, we can run Algorithm 2 on $\Gamma$ to get proper samples of those edges. As for edges of $G$, for each sample $e$ in $H$, with probability $p'_e/p_e$ we add it into $H$ with weight $1/(Np'_e)$, with $N$ being the number of iterations in the inner loop of Algorithm 2. To see that this gives the correct distribution (over all the randomness of the algorithm, including that used to generate $H$) we note that when considering a copy of an edge $e$ that is in $H$, we know that it was placed into $H$ with probability $p_e$ at one iteration of the inner loop. Now, instead of thinking about generating $H$ and then generating $H'$, we can imagine a two-step process that decides whether to include a given edge of $G$ in $H$ and $H'$. When considering an edge $e$ of $G$ at a given iteration, we:

1. Accept and add it in to $H$ with probability $p_e$.

2. If we accepted in Step 1, we add it to $H'$ with probability $p'_e/p_e$

This process adds edge $e$ to $H'$ with probability exactly $p'_e$.

The algorithm presented above basically implements this process for all edges of $G$. Notice that it only considers edges of $H$. This is because we need not worry about edges outside of $H$, since they were already rejected in Step 1, and thus, are irrelevant at Step 2.

This is an overview of the algorithm, if we have access to the true probabilities $\boldsymbol{p}$ and $\boldsymbol{p}'$. The details are given as Algorithm 3.

**Algorithm 3** Resparsification (knowing the correct probabilities)

**Input:** $H, \Gamma$

**Output:** $H'$, a $1 \pm \epsilon$ sparsifier of $G'$ with $O(n \log^2 n/\epsilon^2)$ edges with high probability,

1: **for all** edges $e$ of $H$ **do**
2: Keep $e$ with probability $p'_e/p_e$ and add it to $H'$ with weight $1/(p'_e N)$.
3: **end for**
4: *%The next loop runs Algorithm 2 on* $\Gamma$
5: **for all** edges $e$ of $\Gamma$ **do**
6: **for** $i$ from 1 to $N$ **do** *%Do this loop implicitly*
7: With probability $p'_e$ put $e$ into $H'$ with weight $1/(p'_e N)$
8: **end for**
9: **end for**
10: **return** $H'$

**Proposition 6.2.1.** *Algorithm 3 produces a $1 \pm \epsilon$ sparsifier $H'$ of $G'$ with high probability (over all the randomness used so far, including the randomness used to sample $H$). The number of edges in this $H'$ is tightly concentrated around $N$. Furthermore, the running time of this algorithm is $O(N)$.*

*Proof.* The claims about the sparsifier quality and size are true because the algorithm is simulating a sampling process from the proper probability distribution on edges of $G'$.

To see the running time claim, we note that since $H$ has $O(N)$ edges, rejection sampling them takes $O(N)$ time. Furthermore, the second part of the algorithm, where we properly sample edges of $\Gamma$, will give us at most $O(N)$ samples with high probability, and since $\Gamma$ consists of $O(N)$ edges, the analysis preceding Proposition 3.4.1 shows that this can be done in $O(N)$ time with high probability.

□

Thus, to complete our construction, we will need a quick way of estimating the $R_e$ and $R'_e$, and from them the $p_e$ and $p'_e$.

### 6.2.2 Estimating effective resistances

Unfortunately, we are not able to exactly compute the effective resistances (and hence selection probabilities) quickly enough, so we will have to estimate them. As we have

discussed, it is enough to provide estimates of the probabilities that are within a constant factor of the true quantities and this is what we will do.

The best known result for estimating effective resistances is given by the following theorem:

**Theorem 6.2.2.** *There is an algorithm that, given a graph $G$ with $m$ edges, with high probability outputs an estimate of the effective resistance along all edges of $G$ to within a constant factor. The algorithm runs in $O(m \log^2 n (\log \log n)^3)$ time.*

This theorem follows by the analysis in [69]. The crucial step is computing an $O(\log n) \times n$ matrix $Z_G$ such that (with high probability) for any vertices $i$ and $j$, the quantity $||Z_G \chi_i - Z_G \chi_j||^2$ is within a $[1/\alpha, \alpha]$ factor of the true effective resistance in $G$ between $i$ and $j$ for some fixed small $\alpha \geq 1$. (We say that $Z_G$ encodes the effective resistances between vertices in $G$ to within a $[1/\alpha, \alpha]$ factor.) Recall from before that the bottleneck involves approximately solving $O(\log n)$ linear systems in $L_G$, each of which takes $O(m \log n (\log \log n)^3)$ time, using the recent result of Koutis, Miller, and Peng [42]. Recall further that by a result of Koutis, Levin, and Peng [40] running the solver to get a constant error guarantee (rather than the inverse polynomial one required by Spielman and Srivastava) is enough to provide the desired estimate of the effective resistances. An inverse polynomial error guarantee would require an extra $O(\log n)$ factor in the running time. Once the $Z_G$ matrix is computed, it takes $O(m \log n)$ time to calculate the effective resistances along all edges of $G$.

For our purposes, we need to estimate the effective resistances in $G'$ of all edges in $H + \Gamma$, of which there are $\tilde{O}(n/\epsilon^2)$, and we need to do this in $\tilde{O}(n/\epsilon^2)$ time. Now, because $H + \Gamma$ is a $1 \pm \epsilon$ approximation of $G'$, the effective resistance in $H + \Gamma$ between any two vertices is very close to the effective resistance in $G'$ between those same vertices. Thus, to give a good estimate of the effective resistances of all edges in $H$ and $\Gamma$ in $G'$ we can compute their effective resistances in $H + \Gamma$. By the above theorem, this takes time $\tilde{O}(n/\epsilon^2)$, since $H + \Gamma$ has $\tilde{O}(n/\epsilon^2)$ edges. In particular, we compute a matrix $Z_{H+\Gamma}$ such that $Z_{H+\Gamma}$ encodes the effective resistances in $H + \Gamma$ between all pairs of vertices to within a $[1/\alpha, \alpha]$ factor, and use it to evaluate the

estimated effective resistances along edges of $H$ and $\Gamma$. Note, however, that if running time were not an issue, we could in principle use $Z_{H+\Gamma}$ to estimate the effective resistance in $G'$ between any pair of vertices. With high probability, the result would be within a $[1/(\alpha(1+\epsilon)), \alpha/(1-\epsilon)]$ factor of the true values.

### 6.2.3 Putting it all together

Now we are ready to show the final algorithm. Before we do, however, it will be convenient to have a few definitions.

**Definition 6.2.3.** Given a graph $G$, with true edge probabilities $\boldsymbol{p} = \{p_e\}_{e \in G}$, and given a constant $\alpha \geq 1$, we say that a collection of probabilities $\tilde{\boldsymbol{p}} = \{\tilde{p}_e\}_{e \in G}$ is $\alpha$-good with respect to $G$ if for all $e \in G$ it is the case that

$$(1/\alpha)\, p_e \leq \tilde{p}_e \leq \alpha p_e.$$

**Definition 6.2.4.** With notation as in the above lemma, we say that a graph $H$ is $\alpha$-good with respect to $G$ if it is generated by Algorithm 2 applied to $G$, with selection probabilities $\tilde{\boldsymbol{p}}$ for some $\alpha$-good $\tilde{\boldsymbol{p}}$.

We know that there exists a small $\alpha \geq 1$ such that, given any graph $G$ and a $1 \pm \epsilon$ sparsifier $H$, we can use $H$ to compute probabilities $\tilde{\boldsymbol{p}}$ that are $\alpha$-good for $G$. Specifically, we do this by first computing a matrix $Z_H$ encoding the effective resistances in $H$ between vertices to within a good approximation factor, and then noticing that the effective resistances in $G$ will also be well-approximated, since $H$ is a sparsifier. This gives us estimates of the probabilities that are within a $[1/\alpha, \alpha]$ factor of the true quantities, for a small fixed $\alpha \geq 1$. In what follows, we will focus on this $\alpha$. For the purposes of the algorithm, we will use $Z_H$ to only compute the probabilities of edges we need. In the analysis, however, it will be useful to think about the probabilities of all the edges.

So, let $G$ be a graph and $H$ be $\alpha$-good with respect to $G$ for $\alpha$ as above. Then, it is a $1 \pm \epsilon$ sparsifier of $G$ with high probability. Further $H$ has $O(N)$ edges with

Table 6.1: Notation used in description and analysis of resparsification algorithm

| | |
|---|---|
| $G$ | The original graph |
| $H$ | Sparsifier of $G$, generated using Algorithm 2 and probabilities $\tilde{\boldsymbol{p}}$ |
| $\Gamma$ | Edges added to $G$ |
| $G'$ | $G + \Gamma$, the new graph |
| $R_e$ | True effective resistance along edge $e$ in $G$ |
| $\tilde{R}_e$ | Estimate of effective resistance along edge $e$ in $G$ |
| $R'_e$ | True effective resistance along edge $e$ in $G'$ |
| $\tilde{R}'_e$ | Estimate of effective resistance along edge $e$ in $G'$ |
| $p_e$ | $R_e/(n-1)$, the true probability of selecting edge $e$ when sparsifying $G$ |
| $\tilde{p}_e$ | $\tilde{R}_e/(n-1)$, an estimate to this probability |
| $p'_e$ | $R'_e/(n-1)$, the true probability of selecting edge $e$ when sparsifying $G'$ |
| $\tilde{p}'_e$ | $\tilde{R}'_e/(n-1)$, an estimate to this probability |

high probability. Let the $\tilde{p}_e$ be the estimates of the probabilities of edges in $G$, which were used to generate $H$. As before, $\Gamma$ will represent the new edges (of which there are $O(N) = O(n \log^2 n/\epsilon^2)$), and $G' := G + \Gamma$. Denote by $\tilde{p}'_e$ the probabilities of edges in $G'$, computed using $H + \Gamma$, as described previously; they are $\alpha$-good for $G'$ with high probability.

We have summarized the relevant notation in Table 6.1.

Consider Algorithm 4. For conceptual convenience, we will assume that we input the probabilities $\tilde{p}_e$ used to generate $H$, and we will output the probabilities used to generate $H'$ so that they are available in the next resparsification step.

It is not hard to see that this algorithm simulates the random process for sparsifying $G'$ using Algorithm 2. (Again, we reuse the randomness used to generate $H$.) The procedure is almost identical to the one in Algorithm 3, with two changes. Firstly, we now sample with probabilities that are not exactly the true ones, but are good approximations. Secondly, we perform the modification in Step 4.

We remark that we need Step 4 since approximation errors might cause the estimate of the probability of an edge to go up after we have added $\Gamma$, even though the true probabilities should go down. For rejection sampling to simulate the proper probability distributions, the probabilities have to be non-increasing. If it is the case that $\tilde{p}_e \leq \tilde{p}'_e$, then we can sample $e$ for $H'$ with probability at most $\tilde{p}_e$, which is what our algorithm does. We show that the change does not in fact hurt our construction.

**Algorithm 4** Resparsification

**Input:** $H, \Gamma$, as well as the $\tilde{p}_e$ for every edge $e \in H$.

**Output:** $H'$, a $1 \pm \epsilon$ sparsifier of $G'$ with $O(n \log^2 n/\epsilon^2)$ edges with high probability, as well as $\tilde{p}'_e$ for every edge $e \in H'$.

1: Estimate the effective resistances in $G'$ of all the edges of $H + \Gamma$.
2: For $e \in H + \Gamma$, let $\tilde{p}'_e = \tilde{R}'_e/(n-1)$ *%Good approximation to true $p_e$*
3: **for** each edge $e$ of $H$ **do**
4: $\quad \tilde{p}'_e \leftarrow \min(\tilde{p}_e, \tilde{p}'_e)$
5: **end for**
6: **for all** edges $e$ of $H$ **do**
7: $\quad$ Keep $e$ with probability $\tilde{p}'_e/\tilde{p}_e$ and add it to $H'$ with weight $1/(\tilde{p}'_e N)$.
8: **end for**
9: **for all** edges $e$ of $\Gamma$ **do**
10: $\quad$ **for** $i$ from 1 to $N$ **do** *%Do this loop implicitly*
11: $\quad\quad$ With probability $\tilde{p}'_e$ put $e$ into $H'$ with weight $1/(\tilde{p}'_e N)$
12: $\quad$ **end for**
13: **end for**
14: **return** $H'$ and the $\tilde{p}'_e$ for $e \in H'$.

**Lemma 6.2.5.** *Let $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{p}}'$ be collections of probabilities that are $\alpha$-good for $G$ and $G'$ respectively, and denote by $\mathbf{p}$ and $\mathbf{p}'$ the collection of true probabilities of edges of $G$ and $G'$. Then, for all $e \in G$, it is the case that $\min(\tilde{p}_e, \tilde{p}'_e) \geq p'_e/\alpha$.*

*Proof.* Indeed, suppose $p_e$ and $p'_e$ are the true probabilities of $e$ in $G$ and $G'$ respectively, and assume that $\tilde{p}_e \geq p_e/\alpha$ and $\tilde{p}'_e \geq p'_e/\alpha$ for some $\alpha$. As we noted previously, because effective resistances cannot increase as we add edges to a graph, we must have $p_e \geq p'_e$, hence $\tilde{p}_e \geq p_e/\alpha \geq p'_e/\alpha$, and hence $\min(\tilde{p}_e, \tilde{p}'_e)$ is at least as big as $p'_e/\alpha$. The desired claim follows. □

We can conclude that if the hypotheses of the lemma hold, the collection of probabilities $\tilde{\mathbf{p}}'$, after the modification in Step 4 of the algorithm, is $\alpha$-good for $G'$. Algorithmically, we only do the modification for edges of $H$, and only have access to edges in $H$ and $\Gamma$. For the purpose of analysis we can think of modifying the probabilities of all edges in $G$ (only the edges of $H$ matter for the purposes of the algorithm, however).

Now, fix $\alpha$ as in the text following Definition 6.2.4. We will show that if Algorithm 4 gets as input a graph that is $\alpha$-good with respect to $G$ it will produce,

with high probability, a graph that is $\alpha$-good for $G'$. Therefore, the property of being $\alpha$-good for the current graph is an invariant of the resparsification procedure.

**Theorem 6.2.6.** *Let notation be as above, and consider running Algorithm 4 on input $H$ and $\Gamma$. Suppose that $H$ is $\alpha$-good with respect to $G$. Then, with high probability, the graph $H'$ output by Algorithm 4 is $\alpha$-good with respect to $G'$.*

*Proof.* Since $H$ is $\alpha$-good with respect to $G$, with high probability it is a $1 \pm \epsilon$ sparsifier of $G$. In this case, $H + \Gamma$ is a $1 \pm \epsilon$ sparsifier of $G'$, and thus we can use it to give estimates of effective resistances and hence probabilities $\tilde{\boldsymbol{p}}'$ that are $\alpha$-good for $G'$. (For the algorithm, we only need to compute the probabilities for edges in $H$ and $\Gamma$; the matrix $Z_{H+\Gamma}$ does, however, encode good estimates of all the effective resistances, and hence probabilities.) For each $e \in H$, the algorithm replaces $\tilde{p}'_e$ by $\min(\tilde{p}_e, \tilde{p}'_e)$, which still gives us a collection of $\alpha$-good probabilities for $G'$, by the above lemma. Then, the rejection sampling step effectively samples edges of $G$ with these probabilities $\tilde{\boldsymbol{p}}'$. This gives us an $\alpha$-good graph $H'$. □

Finally, consider running the full update algorithm, where we add edges to the original graph and its sparsifier, and resparsify every $O(N)$ steps. Let $G$ and $G'$ be the graphs at consecutive resparsification steps. If $H$ is $\alpha$-good for $G$, then the previous theorem tells us that with high probability, resparsifying $G'$ will give us an $\alpha$-good graph $H'$ and associated probabilities $\tilde{\boldsymbol{p}}'$. Moreover, with high probability, $H'$ will have $O(N)$ edges. Provided $H'$ is $\alpha$-good (which happens overwhelmingly often), we will be able to continue the procedure. We can union bound the probability of failure over all the resparsification steps to see that with high probability, at all times we maintain a sparsifier of the subgraph received thus far. By another union bound argument, we see that with high probability all our sparsifiers have $O(N)$ edges.

To compute the running time, we note that estimating the relevant effective resistances takes $\tilde{O}(n/\epsilon^2)$ times since $H + \Gamma$ has $O(N)$ edges with high probability. We only need to compute $\tilde{O}(n/\epsilon^2)$ effective resistances (since we do this only for edges in $H$ and those in $\Gamma$). Determining the probabilities and sampling also takes $\tilde{O}(n/\epsilon^2)$

time. We resparsify every $\tilde{O}(n/\epsilon^2)$ steps, so we conclude that the update procedure takes $\tilde{O}(1)$ steps per added edge.

By keeping careful track of the running times of the construction, we can prove:

**Theorem 6.2.7.** *Our update algorithm takes $O(\log^2 n(\log\log n)^3)$ operations per added edge.*

*Proof.* The bottleneck of the algorithms is estimating the effective resistances. This takes $O(n\log^4 n(\log\log n)^3/\epsilon^2)$ time for a graph with $O(n\log^2 n/\epsilon^2)$ edges. Since we resparsify after adding $O(n\log^2 n/\epsilon^2)$ edges, the amortized cost is $O(\log^2 n(\log\log n)^3)$ per added edge. □

### 6.2.4 Error-forgetfulness of the construction

Before concluding this section, we note one interesting property of our construction in Algorithm 4. Using $H$ and $H+\Gamma$, which are approximations to $G$ and $G'$ respectively, we obtain estimates on effective resistances, which are slightly worse than those we would get had we used the full graphs $G$ and $G'$ (but allow us to do the computation much faster). Despite the approximations that we make, by resparsifying using our algorithm we once again obtain a high-quality sparsifier (with high probability), allowing us to make the approximation all over. In other words, because we take enough samples, and do so intelligently, the errors we make in approximating the effective resistances do not propagate; the procedure has no memory for the approximations we made in the past.

Compare this to a more naïve approach to the problem of resparsifying. If we have $G$, $G'$, $H$ and $H+\Gamma$, defined as before, it is tempting to use Algorithm 1 or Algorithm 2 to sparsify $H+\Gamma$ directly to a smaller graph. Unfortunately, the resulting graph $\bar{H}$ is a $1\pm\epsilon$ approximation of $H+\Gamma$, which is a $1\pm\epsilon$ approximation of $G'$, so $\bar{H}$ is only guaranteed to be a $(1\pm\epsilon)^2 \approx 1\pm 2\epsilon$ sparsifier of $G'$. In other words, the error propagates.

### 6.2.5 Straightforward generalizations

It is easy to generalize the above construction to the following cases. First, the construction goes through almost directly for the case of weighted graphs, where we are allowed to add weighted edges. For example, the probability of selecting an edge becomes the weight of that edge times its effective resistance. The weights with which we add sampled edges depend on their weights in $G$, so in order to do this properly, we should store the weights of the edges in the current sparsifier.

We can also consider operations where we increase the weight of an edge $e$ of $G$ by some amount $w$. In this case, we imagine adding an edge parallel to $e$ and with weight $w$ to $G$, and proceed as before (we add $e'$ with weight $w$ to $H$, and resparsify after some number of steps). The reason for considering parallel edges here is that while increasing the weight of an edge decreases the probabilities of other edges, it may increase the probability of that edge, which our construction would not be able to handle. If we instead add an independent copy of the edge, all the arguments go through.

Secondly, we can envision adding vertices as well as edges to $G$. Adding a vertex and connecting it by an edge to some existing vertex does not affect the effective resistances of the other edges, and it does not increase the number of connected components in the graph. Hence, once again, the probability of existing edges can only decrease, and we can use the same arguments. Here, by adding vertices, we increase the number of times we need to sample in the inner loop of Algorithm 2 in order to get a $1 \pm \epsilon$ approximation guarantee. If we have an upper bound on the number of vertices we will end up with, we can ensure that we take enough samples from the outset.

### 6.2.6 The semi-streaming setting

The update algorithm described above goes through almost unchanged in the semi-streaming case (where we start with the empty graph). After adding the first $CN$ edges (where $N = O(n \log^2 n/\epsilon^2)$), we use Algorithm 4 (with $H$ set to the empty

graph and $\Gamma$ set to the current graph), giving us a $1 \pm \epsilon$ approximation to the current graph, containing around $N$ edges in expectation. The number of edges is in fact tightly concentrated around this expectation, and is almost certainly $O(N)$. Then we continue as before, adding edges and resparsifying when needed.

For our algorithm to be valid in the semi-streaming model, we only need to prove that it requires $\tilde{O}(n/\epsilon^2)$ work space. But this is immediate, since, with high probability, we will only deal with graphs of $\tilde{O}(n/\epsilon^2)$ edges throughout the run.

If we would like to end up with a sparsifier containing $O(n \log n/\epsilon^2)$ edges, we can run Algorithm 1 on the output, which will change the final error guarantee from $1 \pm \epsilon$ to $(1 \pm \epsilon)^2$. This one-time amplification in error should be acceptable for most applications. If we need to end up with a $1 \pm \epsilon$ sparsifier, we just change the error requirement of our procedures to give us $1 \pm \epsilon/3$ sparsifiers at intermediate steps, and find a $1 \pm \epsilon/3$ sparsifier of the output (using Algorithm 1); this increases space requirements by a constant factor.

## 6.3 Conclusions and future work

We have presented an algorithm for maintaining a sparsifier of a growing graph, such that the average time is $\tilde{O}(1)$ for each added edge. The main idea is a resampling procedure that uses information in the existing sparsifier to construct a new one very quickly. Our construction is robust and holds relatively unchanged for several natural variants. An interesting question left open by our work is whether similar results could be obtained in a dynamic model that permits the removal of edges as well. While this is a somewhat unnatural notion in the semi-streaming setting, it is a very reasonable goal in the dynamic setting where one aims to maintain a sparsifier for a graph that is changing over time.

In fact, recently Goel, Kapralov, and Post [30] and independently Ahn, Guha, and McGregor [4, 6] proposed an update algorithm for combinatorial sparsification in the dynamic setting, with edge deletions allowed. It would be interesting to see if similar ideas apply in the case of spectral sparsification.

# Part II

# Population Genetics

# Chapter 7

# Introduction

The recent explosion in the availability of genetic data has led to significant advances in understanding human history. These advances have, in addition, benefited from emerging algorithmic and statistical techniques that have made it possible to efficiently analyze the deluge of genetic information. Yet, despite recent developments, much computational work remains to be done.

In this part of the dissertation, we study a fundamental question in population genetics: how are the various human populations interrelated, and, in particular, what is the history of mixture between them? Using a simple model for admixture and a novel algorithm we are able to deduce numerous plausible admixture scenarios, in many cases gaining new insights into human history, or reproducing recent discoveries. Our algorithm, MixMapper, is principled and efficient, and can be applied in various settings.

Specifically, MixMapper is a fast method for constructing phylogenetic trees including admixture events using single nucleotide polymorphism (SNP) genotype data. The MixMapper algorithm determines the best-fit positions of individual admixed populations relative to an initial pure tree. Said another way, if we have a number of populations whose relationship is modeled well by a pure phylogenetic tree (without admixture) and a new population, we try to determine a likely admixture scenario among tree populations that produced the new population.

Mathematically, our approach is based on previously developed theoretical rela-

tionships between allele frequency ("$f$-") statistics under an instantaneous admixture model. MixMapper makes use of certain structural features of these statistics to optimize the admixture parameters and provides estimates of statistical uncertainty. Finally, the results can be expressed using a new method to convert all genetic distances to absolute drift length units. We apply the method to recently published data from a SNP array designed especially for use in population genetics studies, with a simple ascertainment scheme that eliminates bias in the estimation of allele frequency changes and heterozygosity in modern and ancestral populations. In all, we obtain confident results for 33 HGDP populations, 22 of them admixed. Notably, we confirm a robust signal of ancient admixture in all surveyed European populations, involving a proportion of 20-45% Siberian or Central Asian ancestry, and fit six populations as second-order admixtures using an admixed European group as one ancestor. Overall, MixMapper can help shed light on fine-scale aspects of population relationships and is a useful tool for future investigations into human demographic history.

# Bibliographic notes

The work in this part of the thesis is largely from an upcoming paper [49]. I would like to thank Nick Patterson for introducing us to the topic, and the rest of my co-authors for a stimulating research experience.

# Chapter 8

# Background on population genetics

## 8.1 Genetic drift

Consider a population of $N$ individuals. Let us focus on a particular genetic locus. A locus could be a gene, or, in the case of single-nucleotide polymorphism (SNP) data, a particular nucleotide in the genome. Since each individual has 2 copies of each chromosome, there are $2N$ copies of the locus in the population.

Suppose that in the population there are two possible *alleles* at the locus, one of which is (arbitrarily) assigned to be the "standard" allele, while the other allele is the "variant" allele. Suppose that the standard allele has frequency $p$ over the $2N$ copies of the locus.

We consider a very simplistic synchronous model of reproduction. The population is composed of $N$ individuals, and the standard allele has a certain frequency $p$. At the next time step, $N$ children are born through a random mating process, at which point all the parents die. This leaves a population of $N$ individuals, and we are interested in the standard allele frequency in this new population (and, by extension, how it changes throughout generations).

If $N$ is large (and effectively infinite), then the frequencies of the alleles stay roughly constant, and moreover, the frequencies of homozygous and heterozygous individuals will settle at constant values; these values are given by the *Hardy-Weinberg equilibrium.* However, stochastic effects can cause the frequencies to change, and this

phenomenon becomes more visible as population size decreases. The process is known as *genetic drift.*

Genetic drift tends to push populations to be homozygous. Note that in the absence of new mutations, once a population becomes homozygous at a locus it stays that way.

To model genetic drift, one often uses the Wright-Fisher process. In this process, we have $2N$ alleles of two potential variants. At each generation, we choose $2N$ alleles with replacement from the previous set. This is a crude model of what happens in genetic drift, and bears little relation to the biology of reproduction. Nevertheless, its mathematical properties encompass the dynamics of genetic drift fairly well.

As before, suppose we have two possible variants and let $p$ be the frequency of the standard allele at a given time step. Then, by considering the Wright-Fisher process, it is not hard to see that at the next time step, the frequency will be $p'$, a random quantity whose mean is $p$ and whose variance is $p(1-p)/(2N)$.

As we mentioned, over time genetic drift reduces a population's heterozygocity. Mathematically, suppose that the heterozygocity at some generation is $H_0$. Then, it is not hard to show that the expected heterozygocity after after $n$ generations of genetic drift via the Wright-Fisher process is $H_0(1-1/(2N))^n$.

Throughout this discussion, a key assumption we make is *neutrality*: we posit that neither one of the two genetic variants at each locus offers a selective advantage over the other one. Of course, in many cases this assumption is demonstratively false: the introduction of a new mutation is often deleterious, though occasionally it confers a selective advantage. Nevertheless, the neutrality assumption is a good first approximation, especially if we deal with non-gene regions, as we primarily do in this work. Additionally, given the large number of sites we consider, it is a reasonable assumption that neutrality holds "on average," and even if the assumption is violated at some sites, the signal of selection will be washed out overall.

For more information on genetic drift, the reader can consult any standard text on population genetics, e.g. [28].

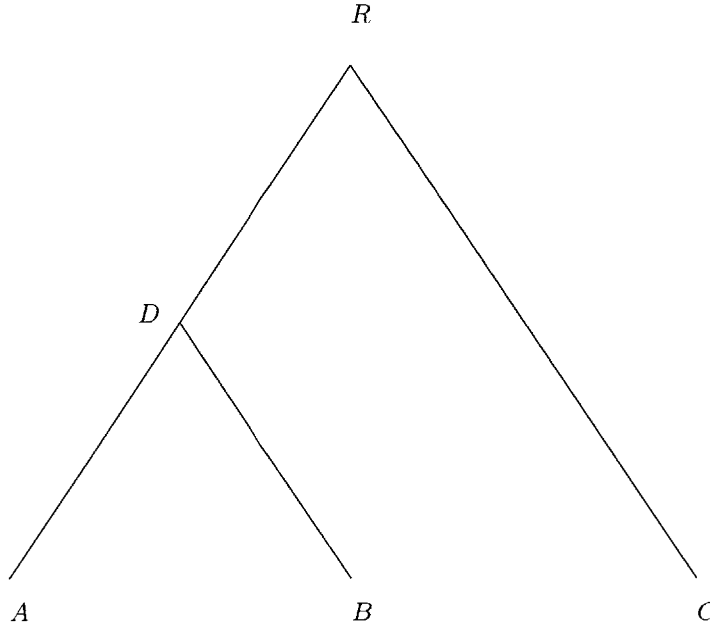



Figure 8-1: An example phyologenetic tree

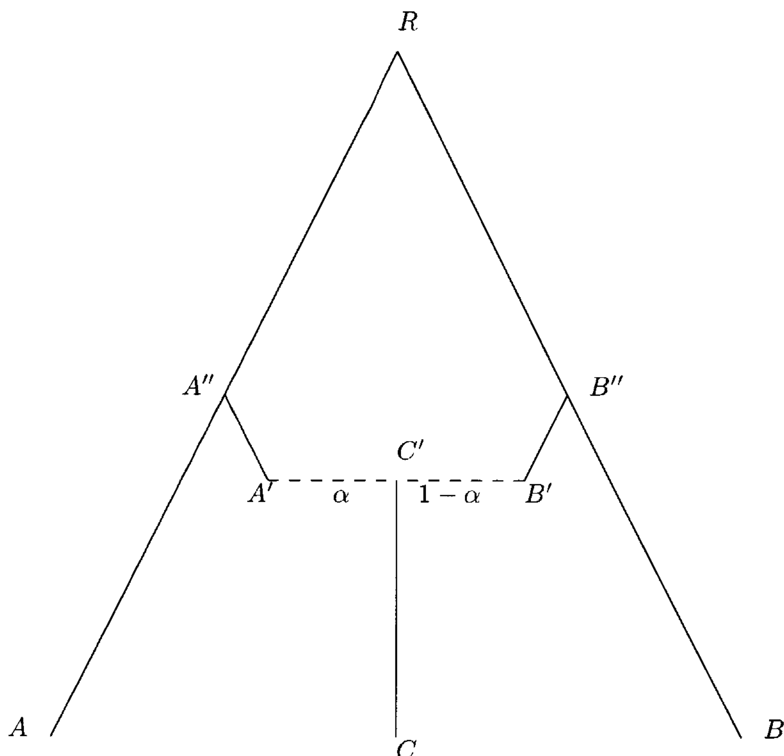
## 8.2 Phylogenetic trees

Phylogenetic trees are simple models of the relationships between populations. Consider the tree in Figure 8-1. The model here is that there is a root population $R$ that splits into two non-interacting branches. The branches then undergo independent genetic drift, potentially with different population sizes (so one of the branches can undergo significantly more drift in this model). After some amount of drift, the population $D$ in the left-hand branch splits further, and produces populations $A$ and $B$ by independent genetic drift. The right-hand branch at $R$ produces population $C$ at the end of genetic drift. We assume that $A$, $B$, and $C$ are the present-day populations and we are able to get genetic data from them. However, we have no direct access to genetic data of the non-leaf populations.

## 8.3 Admixture graphs

Phylogenetic trees are useful models, but they do not capture the complex mixing that can occur between populations even after a split. A simple and useful model for

Figure 8-2: An illustration of a simple admixture tree. In this case, $C'$ is produced by a point admixture process (dashed line) from $A'$ and $B'$ with admixture proportions $\alpha$ and $1-\alpha$ respectively. Then, $C$ is produced via a genetic drift process from $C'$.




that is the *point-admixture* process [63, 59].

Suppose we have populations $A$ and $B$, which have standard allele frequencies $p_A$ and $p_B$ respectively. A point admixture process creates a population $C$ whose standard allele frequency is $\alpha p_A + (1-\alpha)p_B$. This models an instantaneous, one-time mixture, where $C$ inherits $\alpha$ of its genome from $A$ and $1-\alpha$ from $B$.

One typical example of an admixture graph we will consider is given in Figure 8-2. In this figure, the dotted lines connect the locations where admixture events take place. Importantly, populations $A'$ and $B'$, and not $A$ and $B$, are the ones that undergo the admixture event.

## 8.4 Going to multiple loci

Thus far, we have been talking about genetic drift and admixture for a single locus. Of course, the genome consists of many loci. In this work, we will study single-nucleotide polymorphism (SNP) data. In other words, we have a set of genetic locations where every human has one of two possible nucleotides (i.e. we are dealing with *bi-allelic* SNPs). We will consider each SNP location as a separate locus, and, in our data, we have hundreds of thousands of loci to consider. As before we model the loci as following independent drift processes. If we have loci $L_1$ and $L_2$, we will assume that the frequencies of the standard allele at each one evolve via independent genetic drift (though with the same population size). This is a reasonable assumption, as long as the loci are far enough apart for recombination to have destroyed the effects of linkage disequilibrium.

For populations $A$ and $B$ and locus $i$, define $p_A^i$ (resp. $p_B^i$) to be the frequency of the standard allele at position $i$ in population $A$ (resp. $B$). We will omit the superscripts $i$ when the locus is clear from context.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 9

# Methods

We are now ready to present our techniques for inferring admixture history using SNP data. We should, however, say a few words about the dataset we use. This dataset is based on samples from the Human Genetic Diversity Project (HGDP), which genotyped about 1000 individuals from various human populations. Such genotyping is often done on SNP arrays that choose SNPs by some complicated criterion, for example, their possible association with disease. Thus, there is a potential for ascertainment bias, which could affect the results.

To overcome the effects of ascertainment bias, we base our work on a new dataset where the HGDP samples were re-genotyped using an array whose SNPs are carefully selected for population genetics applications [51].

Following a more detailed description of this dataset, we proceed to overview the theoretical underpinnings of the MixMapper algorithm.

## 9.1 Dataset

We used SNP data from 934 HGDP samples [65, 47], which were re-genotyped on the new Affymetrix Axiom Human Origins Array [51, 50, 36]. In particular, we computed statistics based on Panel 4 from the array, which consists of 163,313 SNPs ascertained as heterozygous in the complete genome sequence of a San individual. As pointed out in [51], this SNP panel attempts to overcome problems with ascertainment bias

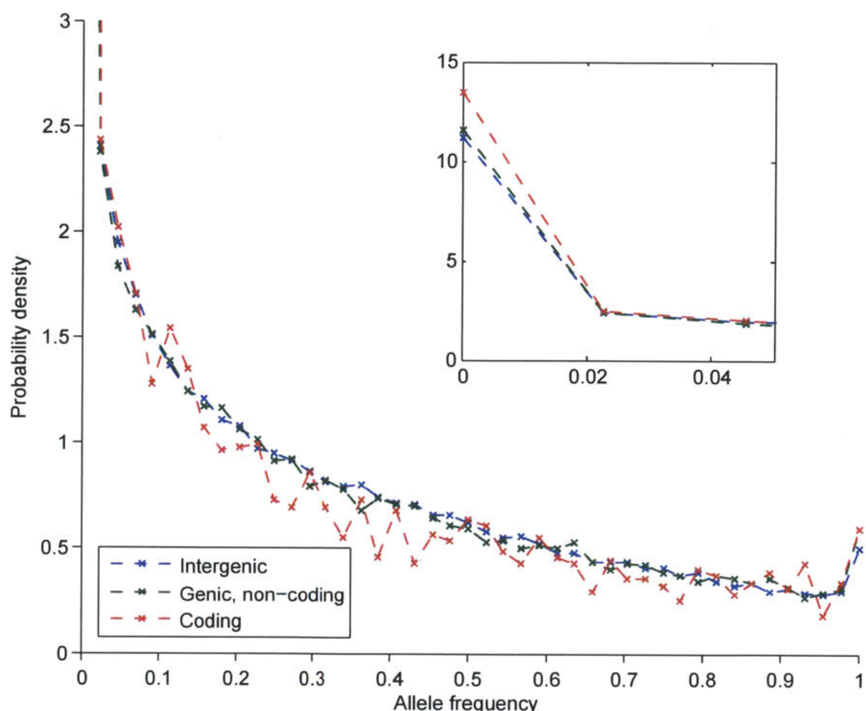



Figure 9-1: **Comparison of allele frequency spectra within and outside gene regions.** We divided the Panel 4 (San-ascertained) SNPs into three groups: those outside gene regions (101,944), those within gene regions but not in exons (58,110), and those within coding regions (3259). Allele frequency spectra restricted to each group are shown for the Yoruba population. Reduced heterozygosity within exon regions is evident, which suggests selection is occurring. (Inset) We observe the same effect in the genic, non-coding spectrum; it is less noticeable but can be seen at the edge of the spectrum.

that come up when using SNP arrays designed for applications in medical genetics.

We excluded 61,369 SNPs that are annotated as falling between the transcription start site and end site of a gene in the UCSC Genome Browser database [26]. Most of the excluded SNPs are not within actual exons, but we found that frequency spectra at these "gene region" loci were slightly shifted toward fixed classes relative to other SNPs, indicative of the action of natural selection (Figure 9-1). Since we assume neutrality in all of our analyses, we chose to remove these SNPs. Our final total of 101,944 SNPs is not as large as in some previous studies, but we feel that the value we gain from precision of ascertainment more than offsets any marginal gains in statistical power from raw numbers.

## 9.2 The $f$-statistics and population admixture

Here we include derivations of the genetic distance equations solved by MixMapper to determine the optimal placement of admixed populations. These results were first presented in [63, 59], and we reproduce them here for completeness, with slightly different emphasis and notation. We also describe in the final paragraph how the structure of the equations leads to a particular form of the system for a full admixture tree.

Our basic quantity of interest is the second $f$-statistic, $f_2$, as defined in [63], which is the squared allele frequency difference between two populations at a biallelic SNP. That is, at SNP locus $i$, we define

$$f_2^i(A, B) := (p_A - p_B)^2,$$

where $p_A$ is the frequency of the standard allele in population $A$ and $p_B$ is the frequency of the allele in population $B$. This is the same as Nei's minimum genetic distance $D_{AB}$ for the case of a biallelic locus [57]. As in [63], we define the unbiased estimator $\hat{f}_2^i(A, B)$, which is a function of finite population samples:

$$\hat{f}_2^i(A, B) := (\hat{p}_A - \hat{p}_B)^2 - \frac{\hat{p}_A(1 - \hat{p}_A)}{n_A - 1} - \frac{\hat{p}_B(1 - \hat{p}_B)}{n_B - 1},$$

where, for each of $A$ and $B$, $\hat{p}$ is the the empirical allele frequency and $n$ is the total number of sampled alleles.

We can also think of $f_2^i(A, B)$ itself as the outcome of a random process of genetic history. In this context, we define

$$F_2^i(A, B) := E((p_A - p_B)^2),$$

the expectation of $(p_A - p_B)^2$ as a function of population parameters. So, for example, if $B$ is descended from $A$ via one generation of Wright-Fisher genetic drift in a population of size $N$, then $F_2^i(A, B) = p_A(1 - p_A)/2N$.

While $\hat{f}_2^i(A,B)$ is unbiased, its variance may be large, so in practice, we use the statistic

$$\hat{f}_2(A,B) := \frac{1}{m}\sum_{i=1}^{m} \hat{f}_2^i(A,B),$$

i.e., the average of $\hat{f}_2^i(A,B)$ over a set of $m$ SNPs. Note that $F_2^i(A,B)$ is not the same for different loci, meaning $\hat{f}_2(A,B)$ will depend on the choice of SNPs. However, we do know that $\hat{f}_2(A,B)$ is an unbiased estimator of the true average $f_2(A,B)$ of $f_2^i(A,B)$ over the set of SNPs.

The utility of the $f_2$ statistic is due largely to the relative ease of deriving equations for its expectation between populations on an admixture tree. The following derivations are borrowed from [63]. We assume throughout that all SNPs are neutral, biallelic, and autosomal, and that divergence times are short enough that there are no further mutations at the selected SNPs. We consider the tree shown in Figure 9-2, consisting of unadmixed populations $A$ and $B$ with common ancestor $P$; an admixed population $C$, descended from a mixture of $A'$ and $B'$ to form $C'$; and the common ancestors $A''$ and $B''$ of $A$ and $A'$ and $B$ and $B'$, respectively. Our admixture model is of a one-time exchange of genetic material: two parent populations mix to form a single descendant population whose allele frequencies are a linear combination of those in the parents with coefficients $\alpha$ and $1-\alpha$. This is of course a very rough approximation to true mixture events, but we feel that it is flexible enough to serve as a reasonable first-order model, along the lines of the common assumption of a constant migration rate for a certain period of time. Most importantly, the point admixture assumption allows us to derive simple formulas for $f_2$ statistics.

As above, let the frequency of a SNP $i$ in population $X$ be $p_X$. Then, for example,

$$\begin{aligned}
E(f_2^i(A,B)) &= E((p_A - p_B)^2) \\
&= E((p_A - p_P + p_P - p_B)^2) \\
&= E((p_A - p_P)^2) + E((p_P - p_B)^2) + 2E((p_A - p_P)(p_P - p_B)) \\
&= E(f_2^i(A,P)) + E(f_2^i(B,P)),
\end{aligned}$$

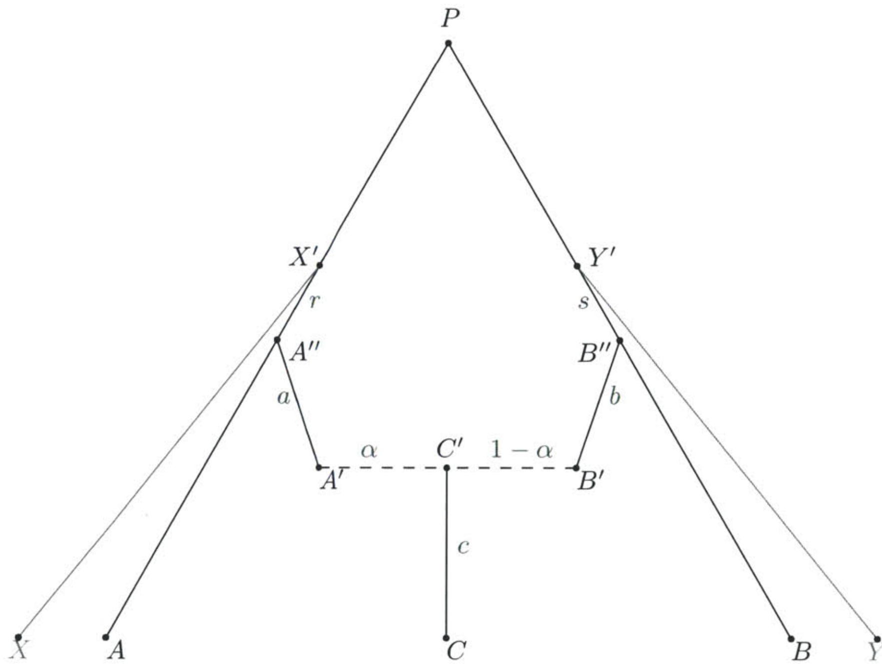



Figure 9-2: **Schematic of part of an admixture tree.** Population $C$ is derived from an admixture of populations $A'$ and $B'$ with proportion $\alpha$ coming from $A'$. The $f_2$ distances from $C$ to the present-day populations $A, B, X, Y$ give four relations from which we are able to infer four parameters: the mixture fraction $\alpha$, the locations of the split points $A''$ and $B''$ (i.e. $r$ and $s$), and the combined drift $\alpha^2 a + (1-\alpha)^2 b + c$.

since the genetic drifts $p_A - p_P$ and $p_P - p_B$ are uncorrelated and have expectation 0. We can decompose these terms further:

$$\begin{aligned}
E(f_2^i(A,P)) &= E((p_A - p_P)^2) \\
&= E((p_A - p_{A''} + p_{A''} - p_P)^2) \\
&= E((p_A - p_{A''})^2) + E((p_{A''} - p_P)^2) + 2E((p_A - p_{A''})(p_{A''} - p_P)) \\
&= E(f_2^i(A, A'')) + E(f_2^i(A'', P)).
\end{aligned}$$

Here, again, $E(p_A - p_{A''}) = E(p_{A''} - p_P) = 0$, but $p_A - p_{A''}$ and $p_{A''} - p_P$ are not independent; for example, if $p_{A''} - p_P = -p_P$, i.e. $p_{A''} = 0$, then necessarily $p_A - p_{A''} = 0$. However, $p_A - p_{A''}$ and $p_{A''} - p_P$ are independent conditional on a single value of $p_{A''}$, meaning the conditional expectation of $(p_A - p_{A''})(p_{A''} - p_P)$ is 0. By the double expectation theorem,

$$E((p_A - p_{A''})(p_{A''} - p_P)) = E(E((p_A - p_{A''})(p_{A''} - p_P)|p_{A''})) = E(E(0)) = 0.$$

From $E(f_2^i(A,P)) = E(f_2^i(A,A'')) + E(f_2^i(A'',P))$, we can take the average over a set of SNPs to yield, in the notation from above,

$$F_2(A,P) = F_2(A,A'') + F_2(A'',P).$$

We have thus shown that $f_2$ distances are additive along an unadmixed-drift tree. This property is fundamental for our theoretical results and is also essential for finding admixtures, since, as we will see, additivity does not hold for admixed populations.

Given a set of populations with allele frequencies at a set of SNPs, we can use the estimator $\hat{f}_2$ to compute $f_2$ distances between each pair. These distances should be additive if the populations are related as a true tree. Thus, it is natural to build a phylogeny using neighbor-joining [66], yielding a fully parameterized tree with all branch lengths inferred. However, in practice, the tree will not exactly be additive, and we may wish to try fitting some population $C$ as an admixture. To do so, we

would have to specify six parameters: the locations on the tree of $A''$ and $B''$; the branch lengths $f_2(A'', A')$, $f_2(B'', B')$, and $f_2(C', C)$; and the mixture fraction $\alpha$. In the notation of Figure 9-2, these are the variables $r$, $s$, $a$, $b$, $c$, and $\alpha$.

In order to fit $C$ onto an unadmixed tree (that is, solve for the six mixture parameters), we use the equations for the expectations $F_2(M, C)$ of the $f_2$ distances between $C$ and each other population $M$ in the tree. Referring to Figure 9-2, with the point admixture model, the allele frequency in $C'$ is $p_{C'} = \alpha\ p_{A'} + (1-\alpha)\ p_{B'}$. So, for a single locus, using additivity,

$$
\begin{aligned}
E(f_2^i(A, C)) &= E((p_A - p_C)^2) \\
&= E((p_A - p_{A''} + p_{A''} - p_{C'} + p_{C'} - p_C)^2) \\
&= E((p_A - p_{A''})^2) + E((p_{A''} - \alpha\ p_{A'} - (1-\alpha)\ p_{B'})^2) + E((p_{C'} - p_C)^2) \\
&= E(f_2^i(A, A'')) + \alpha^2 E(f_2^i(A'', A')) + (1-\alpha)^2 E(f_2^i(A'', B')) + E(f_2^i(C', C)).
\end{aligned}
$$

Averaging over SNPs, and replacing $E(f_2(A, C))$ by the estimator $\hat{f}_2(A, C)$, this becomes

$$
\begin{aligned}
\hat{f}_2(A, C) &= F_2(A, X') - r + \alpha^2 a + (1-\alpha)^2 (r + F_2(X', Y') + s + b) + c \\
\implies \hat{f}_2(A, C) - F_2(A, X') &= (\alpha^2 - 2\alpha) r + (1-\alpha)^2 s + \alpha^2 a + (1-\alpha)^2 b + c + \\
&\quad (1-\alpha)^2 F_2(X', Y').
\end{aligned}
$$

The quantities $F_2(X', Y')$ and $F_2(A, X')$ are constants that can be read off of the neighbor-joining tree. Similarly, we have

$$
\hat{f}_2(B, C) - F_2(B, Y') = \alpha^2 r + (\alpha^2 - 1) s + \alpha^2 a + (1-\alpha)^2 b + c + \alpha^2 F_2(X', Y').
$$

For the outgroups $X$ and $Y$, we have

$$\begin{aligned}\hat{f}_2(X,C) &= \alpha^2(c+a+r+F_2(X,X')) + (1-\alpha)^2(c+b+s+F_2(X',Y')+F_2(X,X')) + \\ &\quad 2\alpha(1-\alpha)\,(c+F_2(X,X')) \\ &= \alpha^2 r + (1-\alpha)^2 s + \alpha^2 a + (1-\alpha)^2 b + c + (1-\alpha)^2 F_2(X',Y') + F_2(X,X')\end{aligned}$$

and

$$\hat{f}_2(Y,C) = \alpha^2 r + (1-\alpha)^2 s + \alpha^2 a + (1-\alpha)^2 b + c + \alpha^2 F_2(X',Y') + F_2(Y,Y'). \quad (9.1)$$

Assuming additivity within the neighbor-joining tree, any population descended from $A''$ will give the same equation (the first type), as will any population descended from $B''$ (the second type), and any outgroup (the third type, up to a constant and a coefficient of $\alpha$). Thus, no matter how many populations there are in the unadmixed tree—and assuming there are at least two outgroups $X$ and $Y$ such that the points $X'$ and $Y'$ are distinct—the system of equations consisting of $E(f_2(P,C))$ for all $P$ will contain precisely enough information to solve for $\alpha$, $r$, $s$, and the linear combination $\alpha^2 a + (1-\alpha)^2 b + c$. We also note the useful fact that for a fixed value of $\alpha$, the system is linear in the remaining variables.

This allows us to make predictions for the most likely location of the admixture event. First, if we assume that the admixture takes place off two fixed branches, we can use the equations above to solve for the quantities discussed above, and then consider the norm of the residuals. For a fixed $\alpha$, the system is an overdetermined linear system, and we solve it by least squares. We then optimize for $\alpha$ and consider the residual norm.

The predicted branches are those where the residual norm for the above process is smallest.

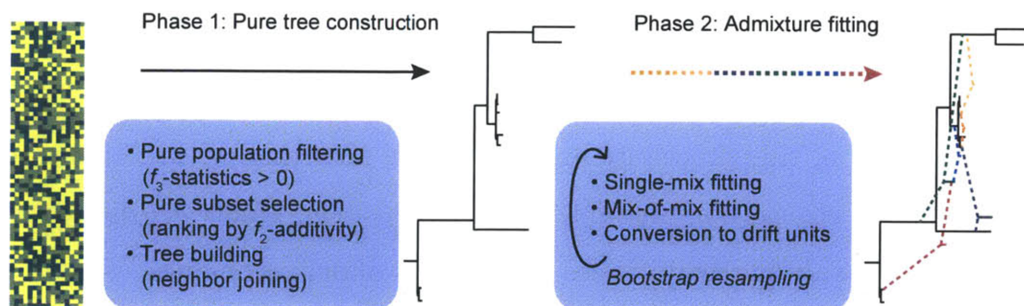



Figure 9-3: **MixMapper workflow.** MixMapper takes as input an array of SNP calls annotated with the population to which each individual belongs. The method then proceeds in two phases, first building a tree of (approximately) unadmixed populations and then attempting to fit the remaining populations as admixtures. In the first phase, MixMapper produces a ranking of possible unadmixed trees in order of deviation from $f_2$-additivity; based on this list, the user selects a tree to use as a scaffold. In the second phase, MixMapper tries to fit each remaining population as a simple two-way mixture between branches of the chosen unadmixed tree. Based on the results, the user can ask MixMapper to perform further fitting of populations as mixtures involving admixed populations. In each case MixMapper produces an ensemble of predictions via bootstrap resampling, enabling confidence estimation for inferred results.

## 9.3 The MixMapper Algorithm

We give an illustration of our workflow in Figure 9-3. The entire pipeline for constructing the history for a set of populations is as follows:

1. **Creating a pure tree with a subset of these populations.** The details of this procedure are given in Section 10.1.

2. **Fitting the other populations as admixtures coming from populations in the tree.** We do this using the linear algebraic approach described in the previous sections.

3. **Assessing significance.** The results of the fitting step depend on the choice of SNPs as well as the individuals making up the HGDP populations. We would like to produce estimates for the variation in our parameters, as well as confidence intervals. To that end, we use bootstrapping. We repeat the fitting step with 500 bootstrapped replicates. In each bootstrap sample, we

choose both SNPs and individuals in populations with replacement, such that the total number of SNPs and individuals is the same as in the original dataset.

The details of this procedure are below. We note that we are able to do this efficiently because our method runs so fast. The ability to quickly do this statistical analysis reemphasizes the benefits of our techniques.

## 9.4 Bootstrapping procedure

In order to measure the statistical significance of our parameter estimates, we computed bootstrap confidence intervals for the inferred branch lengths and mixture fractions [22, 23]. Under our model, we identified two primary sources of statistical error: the randomness of the drift process at each of a finite number of SNPs, and the random choice of individuals to represent each population. Our bootstrap procedure was designed to account for both of these simultaneously. First, we divided the genome into 50 evenly-sized blocks, with the premise that this scale should easily be larger than that of linkage disequilibrium among our SNPs. Then, for each of 500 replicates, we resampled the data set by (a) selecting 50 of these SNP blocks at random with replacement; and (b) for each population group, selecting a random set of individuals with replacement, preserving the number of individuals in the group.

For each replicate, we recalculated all pairwise $f_2$ distances and present-day heterozygosity values using the resampled SNPs and individuals (adjusting the bias-correction terms to account for the repetition of individuals) and then constructed the admixture tree of interest. Even though the mixture parameters we estimate (branch lengths and mixture fractions) depend in very complicated ways on many different random variables, we can directly apply the nonparametric bootstrap to obtain confidence intervals [23]. For simplicity, we used percentile bootstrap; thus, our 95% confidence intervals indicate 2.5 and 97.5 percentiles of the distribution of each parameter among the replicate trees.

Computationally, we parallelized MixMapper's mixture-fitting over the bootstrap replicates using MATLAB's Parallel Computing Toolbox.

## 9.5 Heterozygocity and drift length

One disadvantage to building trees with $f_2$ statistics is that the values are not in easily interpretable units. For a single locus, the $f_2$ statistic measures the squared allele frequency change between two populations. However, in practice, one needs to compute an average $f_2$ value over many loci. Since the amount of drift per generation is proportional to $p(1-p)$, the expected frequency change in a given time interval will be different for loci with different initial frequencies. This means that the estimator $\hat{f}_2$ depends on the distribution of frequencies of the SNPs used to calculate it. For example, within an $f_2$-based phylogeny, the lengths of non-adjacent edges are not directly comparable.

In order to make use of the properties of $f_2$ statistics for admixture tree building and still be able to present our final trees in more directly meaningful units, we will show now how $f_2$ distances can be converted into absolute drift lengths. Again, we consider a biallelic, neutral SNP in two populations, with no further mutations, under a Wright-Fisher model of genetic drift.

Suppose populations $A$ and $B$ are descended independently from a population $P$, and we have an allele with frequency $p$ in $P$, $p_A = p + a$ in $A$, and $p_B = p + b$ in $B$. The (true) heterozygosities at this locus are $h_P^i = 2p(1-p)$, $h_A^i = 2p_A(1-p_A)$, and $h_B^i = 2p_B(1-p_B)$. As above, we write $\hat{h}_A^i$ for the unbiased single-locus estimator

$$\hat{h}_A^i := \frac{2n_A\hat{p}_A(1-\hat{p}_A)}{n_A - 1},$$

$\hat{h}_A$ for the multi-locus average of $\hat{h}_A^i$, and $H_A^i$ for the expectation of $h_A^i$ under the Wright-Fisher model (and similarly for $B$ and $P$).

Say $A$ has experienced $t_A$ generations of drift with effective population size $N_A$ since the split from $P$, and $B$ has experienced $t_B$ generations of drift with effective population size $N_B$. Then it is well known that $H_A^i = h_P^i(1 - D_A)$, where $D_A =$

$1-(1-1/(2N_A))^{t_A}$, and $H^i_B = h^i_P(1-D_B)$. We also have

$$\begin{aligned} H^i_A &= E(2(p+a)(1-p-a)) \\ &= E(h^i_P - 2ap + 2a - 2ap - 2a^2) \\ &= h^i_P - 2E(a^2) \\ &= h^i_P - 2F^i_2(A,P), \end{aligned}$$

so $2F^i_2(A,P) = h^i_P D_A$. Likewise, $2F^i_2(B,P) = h^i_P D_B$ and $2F^i_2(A,B) = h^i_P(D_A + D_B)$. Finally,

$$H^i_A + H^i_B + 2F^i_2(A,B) = h^i_P(1-D_A) + h^i_P(1-D_B) + h^i_P(D_A+D_B) = 2h^i_P.$$

This equation is essentially equivalent to one in [57], although Nei interprets his version as a way to calculate the expected present-day heterozygosity rather than estimate the ancestral heterozygosity. To our knowledge, the equation has not been applied in the past for this second purpose.

In terms of allele frequencies, the form of $h^i_P$ turns out to be very simple:

$$h^i_P = p_A + p_B - 2p_A p_B = p_A(1-p_B) + p_B(1-p_A),$$

which is the probability that two alleles, one sampled from $A$ and one from $B$, are different by state. We can see, therefore, that this probability remains constant in expectation after any amount of drift in $A$ and $B$. This fact is easily proved directly:

$$E(p_A + p_B - 2p_A p_B) = 2p - 2p^2 = h^i_P,$$

where we use the independence of drift in $A$ and $B$.

Let $\hat{h}^i_P := (\hat{h}^i_A + \hat{h}^i_B + 2\hat{f}^i_2(A,B))/2$, and let $h_P$ denote the true average heterozygosity in $P$ over an entire set of SNPs. Since $\hat{h}^i_P$ is an unbiased estimator of $(h^i_A + h^i_B + 2f^i_2(A,B))/2$, its expectation under the Wright-Fisher model is $h^i_P$. So, the average $\hat{h}_P$ of $\hat{h}^i_P$ over a set of SNPs is an unbiased (and potentially low-variance)

estimator of $h_P$. If we have already constructed a phylogenetic tree using pairwise $f_2$ statistics, we can use the inferred branch length $\hat{f}_2(A, P)$ from a present-day population $A$ to an ancestor $P$ in order to estimate $\hat{h}_P$ more directly as $\hat{h}_P = \hat{h}_A + 2\hat{f}_2(A, P)$. This allows us, for example, to estimate heterozygosities at intermediate points along branches or in the ancestors of present-day admixed populations.

The statistic $\hat{h}_P$ is interesting in its own right, as it gives an unbiased estimate of the heterozygosity in the common ancestor of any pair of populations (for a certain subset of the genome). For our purposes, though, it is most useful because we can form the quotient

$$\hat{d}_A := \frac{2\hat{f}_2(A, P)}{\hat{h}_P},$$

where the $f_2$ statistic is inferred from a tree. This statistic $\hat{d}_A$ is not exactly unbiased, but by the law of large numbers, if we use many SNPs, its expectation is very nearly

$$E(\hat{d}_A) \approx \frac{E(2\hat{f}_2(A, P))}{E(\hat{h}_P)} = \frac{h_P D_A}{h_P} = D_A,$$

where we use the fact that $D_A$ is the same for all loci. Thus $\hat{d}$ is a simple, direct, nearly unbiased moment estimator for the drift length between a population and one of its ancestors. This allows us to convert branch lengths from $f_2$ distances into absolute drift lengths, one branch at a time, by inferring ancestral heterozygosities and then dividing.

An alternative definition of $\hat{d}_A$ would be $1 - \hat{h}_A/\hat{h}_P$, which also has expectation (roughly) $D_A$. In most cases, we prefer to use the definition in the previous paragraph, which allows us to leverage the greater robustness of the $f_2$ statistics, especially when taken from a multi-population tree.

We note that this estimate of drift lengths is similar in spirit to the widely-used statistic $F_{ST}$. For example, under proper conditions, the expectation of $F_{ST}$ among populations that have diverged under pure drift is also $1 - (1 - 1/(2N_e))^t$ [57]. When $F_{ST}$ is calculated for two populations at a biallelic locus using the formula $(\Pi_D - \Pi_S)/\Pi_D$, where $\Pi_D$ is the probability two alleles from different populations

are different by state and $\Pi_S$ is the (average) probability two alleles from the same population are different by state (as in [63] or the measure $G'_{ST}$ in [57]), then this $F_{ST}$ is exactly half of our $\hat{d}$. As a general rule, drift lengths $\hat{d}$ are approximately twice as large as values of $F_{ST}$ reported elsewhere.

# Chapter 10

# Results

## 10.1 Constructing the pure tree

Our first step in applying MixMapper was to determine a set of populations to use in the unadmixed tree. As has been noted previously [59], most of the 53 HGDP groups exhibit signs of admixture (from the 3-population test), despite the design of the HGDP favoring isolated populations. Thus, computing $f_3$ statistics for all triples of populations and removing populations with negative values (indicative of recent admixture) left only 20 that are potentially unadmixed. Furthermore, most subsets including even half of those 20 populations exhibited significant divergence from $f_2$-additivity, which should hold in the case of pure drift [59].

Upon ranking subsets by additivity, we noticed that there is a substantial penalty (indicating admixture) for any combination of populations including Europeans along with representatives of at least three other continents. Consequently, the putatively unadmixed tree we selected excluded Europeans, consisting instead of the following 10 populations: Dai, Japanese, Karitiana, Lahu, Mandenka, Naxi, Papuan, Suruí, Yi, and Yoruba. Five are from East Asia, two are from Africa, two are from the Americas, and one is from Oceania. These form one of the most additive 10-population subsets representing at least four of the five major continental groups (Africa, Americas, Asia, Europe, Oceania) in the HGDP data set.

The largest absolute error between an $f_2$ distance on this tree and the correspond-

ing value from the data is $0.00112 \pm 0.00032$ s.d. (estimated using our bootstrap procedure). While this is a statistically significant deviation from unadmixed drift, it is quite small when compared to other combinations of populations, and given the simplifying assumptions of our model, exact additivity is not to be expected. We also checked that none of the 10 populations can be fit in a reasonable way as an admixture on a tree built with the other nine.

Finally, after choosing the unadmixed tree, we re-optimized its branch lengths to minimize the sum of squared errors of all pairwise $f_2$ distances. This resulted in only minor changes from the neighbor-joining tree.

## 10.2 Case study: The genetic history of European populations

One particularly notable application of our methods is in determining a likely genetic history of Europeans. Among the HGDP populations, the European populations are Adygei, Basque, French, Italian, Orcadian, Russian, Sardinian, and Tuscan.

A preliminary analysis using $f_3$ statistics suggests that Basque and Sardinian may be best modeled as pure (i.e. non-admixed) populations (data not shown). However, using MixMapper, we are able to see a robust signal of admixture in all European HGDP populations. This admixture involves ancestors of the South Americans as well as an ancient population, which we interpret as ancestral Western Eurasians. We interpret this admixture as a sign of gene flow from ancient Siberians (who are ancestors of the South Americans). This was originally noticed by Patterson *et al.* [59], and our findings give another line of evidence for their discovery.

The results are given in Table 10.1 and illustrated in Figure 10-2. Note that we report the results for 500 bootstrap repetitions, where we sample both the potential SNPs, as well as the individuals from the populations. They are qualitatively similar for all the European populations. We see evidence of ancient admixture between ancestral Eurasians and Siberians, with roughly similar mixture proportions. Notice

Table 10.1: **Mixture parameters for Europeans.**

| AdmixedPop | # rep | $\alpha$ | Branch1Loc (AncNEur) | Branch2Loc (AncWEur) | MixedDrift |
|---|---|---|---|---|---|
| Adygei | 500 | 0.254-0.461 | 0.033-0.078 / 0.195 | 0.140-0.174 / 0.231 | 0.077-0.092 |
| Basque | 464 | 0.160-0.385 | 0.053-0.143 / 0.196 | 0.149-0.180 / 0.231 | 0.105-0.121 |
| French | 491 | 0.184-0.386 | 0.054-0.130 / 0.195 | 0.149-0.177 / 0.231 | 0.089-0.104 |
| Italian | 497 | 0.210-0.415 | 0.043-0.108 / 0.195 | 0.137-0.173 / 0.231 | 0.092-0.109 |
| Orcadian | 442 | 0.156-0.350 | 0.068-0.164 / 0.195 | 0.161-0.185 / 0.231 | 0.096-0.113 |
| Russian | 500 | 0.278-0.486 | 0.045-0.091 / 0.195 | 0.146-0.181 / 0.231 | 0.079-0.095 |
| Sardinian | 480 | 0.150-0.350 | 0.045-0.121 / 0.195 | 0.146-0.176 / 0.231 | 0.107-0.123 |
| Tuscan | 489 | 0.179-0.431 | 0.039-0.118 / 0.195 | 0.137-0.177 / 0.231 | 0.088-0.110 |

Mixture parameters from MixMapper for modern-day European populations (cf. [59]). All eight are nearly unanimously optimized as a mixture between populations related to the "Ancient Northern Eurasian" and "Ancestral Western Eurasian" branches in the pure tree (see Figure 10-2A). Branch1Loc and Branch2Loc are the points at which the mixing populations split from these branches; $\alpha$ is the proportion of ancestry from the Northern Eurasian side; MixedDrift is the sum of drift lengths $\alpha^2 a + (1-\alpha)^2 b + c$; and # rep is the number of bootstrap replicates (out of 500) placing the mixture between these two branches. All ranges shown are 95% bootstrap confidence intervals. See Figure 10-1A for an illustration of the parameters.

also the slightly higher "Mixed Drift" for the Basque and Sardinian populations, consistent with their being small, geographically-isolated populations. This could be the reason that the populations look pure when considering $f_3$ tests. Indeed, as was shown by Reich *et al.* (mathematical appendix to [63]) large post-mixture drift can wash out the $f_3$ signal.

## 10.3 Discussion

We have presented MixMapper, a flexible and robust computational tool for inferring admixture trees from large-scale SNP frequency data. The method can be applied to any number of populations and can fit simple and second-order admixtures at any points within an initial unmixed tree. Unlike previous procedures, only the lists of unadmixed and admixed populations need to be supplied by hand: all of the topological relationships within the resulting admixture tree are inferred automatically. We also take advantage of a new SNP data set based on an unbiased ascertainment

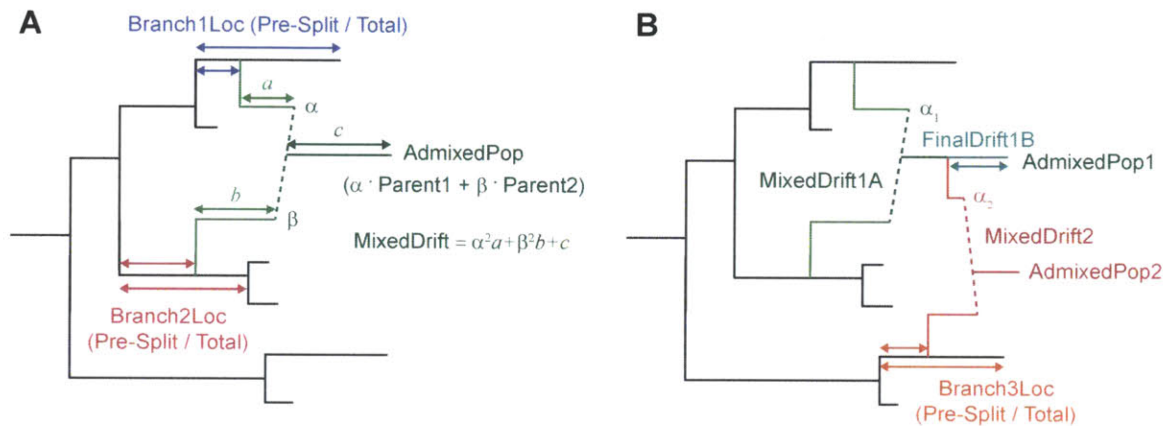
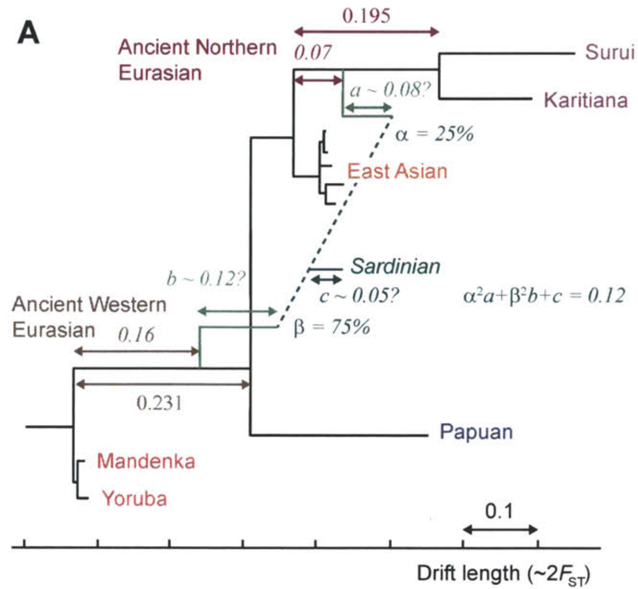



Figure 10-1: **Schematic of mixture parameters reported in tables.** (A) A simple two-way admixture. MixMapper infers four parameters when fitting a given population as an admixture. It finds the optimal pair of branches between which to place the admixture and reports the following: Branch1Loc and Branch2Loc are the points at which the mixing populations split from these branches; $\alpha$ is the proportion of ancestry from Branch1 and $\beta = 1 - \alpha$ is the proportion from Branch2; and MixedDrift is the linear combination of drift lengths $\alpha^2 a + \beta^2 b + c$. (B) A mixture of mixtures: here AdmixedPop2 is modeled as an admixture between AdmixedPop1 and Branch3. There are now four additional parameters; three are analogous to the above, namely, Branch3Loc, $\alpha_2$, and MixedDrift2. The remaining degree of freedom is the position of the split along the AdmixedPop1 branch, which divides MixedDrift into MixedDrift1A and FinalDrift1B.



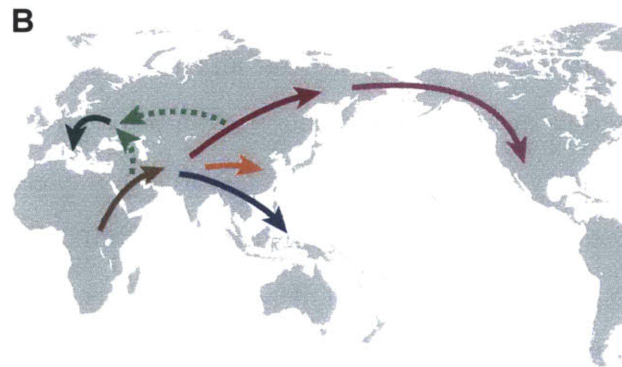




Figure 10-2: **Detail of inferred European admixture.** (A) Detail of the inferred ancestral admixture for Sardinians. One mixing population splits from the pure tree along the common ancestor branch of Americans ("Ancient Northern Eurasian") and the other along the common ancestor branch of all non-Africans ("Ancient Western Eurasian"). Median parameter values are shown; 95% bootstrap confidence intervals can be found in Table 10.1. The branch lengths $a$, $b$, and $c$ are confounded, so we show a plausible combination. (B) As in [59], we interpret the inferred admixture as having occurred between ancient populations living in Siberia and in Europe, respectively. Colored arrows correspond to labeled branches in (A).

scheme, which allows us to perform computations using additive $f_2$ distance units and then convert to readily interpretable absolute drift lengths. Solving the full system of $f_2$ equations with many free parameters can perhaps create a danger of overfitting, but as discussed in more detail in Chapter 9, we use several criteria, most notably bootstrap confidence intervals, to help avoid this problem and generally corroborate the reliability of our results.

We chose to implement MixMapper with trees of intermediate size. Our preferred unadmixed tree contained 10 populations, which is small enough to be reasonably free of complications but large enough to give good coverage of most areas of the world. The program can also be applied at a rougher level with a more global starting tree, requiring virtually no manual intervention, or in more precise, fine-scale applications, if an unadmixed tree consisting of a small set of carefully chosen populations is used.

Using MixMapper, we constructed an admixture tree containing 30 HGDP populations: 10 unmixed, 14 as two-way admixtures, and 6 as mixtures of mixtures. Perhaps the most notable result within the tree is that all European populations we tested are optimally represented as mixtures between a population related to the common ancestor of Americans and a population related to the common ancestor of all non-African populations in our unadmixed tree, confirming an admixture signal first reported by [59]. Our interpretation is that most if not all modern Europeans are descended from at least one large-scale ancient admixture event involving, in some combination: at least one population of Mesolithic European hunter-gatherers; Neolithic farmers, originally from the Near East; and/or further migrations from northern or Central Asia. Either the first or second of these could be related to the "ancient western Eurasian" branch in Figure 10-2, and either the first or third could be related to the "ancient northern Eurasian" branch. While the admixture signal is quite strong, we are unable to pinpoint the sources more closely using these populations; in particular, none of these putative ancestral groups have any direct descendants in our data set, and hence this scenario involves substantial drift (branches $a$ and $b$ in Figure 10-2A) between the split points of the parent populations from the unadmixed tree and the mixture event itself. Present-day Europeans differ in the amount of drift they have

experienced since the admixture and in the proportions of the ancestry components they have inherited, but their overall profiles are similar.

In all, we believe that methods such as MixMapper, and the dataset on which it is based, will prove useful in population genetics studies. We should note, however, that in certain applications, full genome sequences are beginning to replace more limited genotype data sets such as ours. Still we believe that our methods and SNP-based inference more generally will remain valuable in the future. Despite the increasing feasibility of sequencing, it is still much easier and less expensive to genotype samples using a SNP array, and with over 100,000 loci, the data used in this study provide substantial statistical power. Additionally, sequencing technology is currently more error-prone, which can lead to biases in allele frequency-based statistics [62]: for example, rare alleles can be difficult to distinguish from incorrect base calls, meaning that error correction will tend to flatten empirical frequency spectra. Thus, MixMapper should continue to contribute to an important niche of population history inference methods based on SNP allele frequency data.

# Part III

# RNA secondary structure design

# Chapter 11

# Introduction

The design of RNA sequences with specific folding properties is a critical problem in synthetic biology. Solving this problem is an important first step in controlling bio-molecular systems, which can have profound biomedical implications; indeed, it has already proven useful in modifying HIV-1 replication mechanisms [55], reprogramming cellular behavior [16], and designing logic circuits [35].

Here, we aim to design RNA sequences that fold into specific secondary structures. (This problem is also known as inverse folding.) Even in this case, efficient computational formulations remain difficult, with no exact solutions known. Instead, the solutions available today rely on local search strategies and heuristics. Indeed, the computational difficulty of the RNA design problem was proven by M. Schnall-Levin *et al.* [67].

One of the first and most widely known programs for the RNA inverse folding problem is `RNAinverse` [33]. The search starts with a seed sequence specified by the user. At each step thereafter, `RNAinverse` compares the minimum free energy (MFE) structure of the current sequence (i.e. the structure computed from a structure prediction algorithm) with the target structure to determine the mutations to perform; it attempts to traverse the mutational landscape in the direction that improves the current MFE structure's similarity to the target.

Better RNA design tools have been subsequently developed. To our knowledge, the best programs currently available are `INFO-RNA` [13], `RNA-SSD` [9, 3] and `NUPACK`

[79]. Other programs such as rnaDesign [17] or RNAexinv [10] also have demonstrated improvement over RNAinverse. Conceptually however, all current approaches rely on the same principle, which can be delineated in two steps: (i) selection of a seed, and (ii) a (stochastic) local search that aims to mutate the seed to fit the target structure.

The traditional single-sequence iterative-improvement approach is simple and computationally fast: at each point only the next possible point mutations need to be computed and evaluated for fitness so that the best one can be chosen. However, the sequences generated by this approach suffer from several shortcomings. Firstly, due to the presence of energy barriers in the mutational landscape, some good sequences (in terms of structure fit and energetic properties) might be difficult to reach from a given seed. Even worse, sometimes arbitrary initial choices made by such methods can irrevocably bias a search to produce ineffectual designs. For example, since it is easy to grow existing stem structures by single-point sequence mutations, the search can initially take off in the direction of "improving" the structural fit by growing stems, only to falter when other structural elements and rearrangements require multiple point mutations. Finally, constraining the search to directions that improve the structural fitness function in the initial phases of the search runs counter to biological reality because it rewards mutations that bring the structure "closer" to the desired shape but do not directly improve function (e.g., the binding affinity for some ligand).

In this paper, we present RNA-ensign, a novel and complementary approach to the RNA design problem that uses global sampling of an energetic ensemble model of the RNA mutation landscape. More precisely, starting from a random seed sequence, our scheme computes the Boltzmann distribution of *all* $k$-mutants of the seed and samples from these ensemble sequences [75]. RNA-ensign starts by looking at all samples with one mutation (i.e. $k = 1$) and increments this number $k$ until it finds a mutant whose MFE structure matches the design target's secondary structure. Unlike the classical RNA design schemes, this approach largely decouples the forces controlling the search in the mutational landscape from the stopping criterion.

We analyze design choices and show that, compared to local searches, our global sampling approach has advantages. While the importance of the choice of seed is

widely acknowledged, to our knowledge, very few exhaustive studies allow for the precise quantification of its importance given here. We also present an analysis of the strengths and weaknesses of the novel global sampling approach introduced here. While it generates more thermodynamically stable sequences at a high success rate, it is computationally more expensive than local search approaches. Nonetheless, our current implementation can be run on structures with sizes up to 200bp, and thus reaches the current limit of accuracy for base pairing predictions with a nearest neighbor energy models [43, 21].

This study aims to provide a complete comparison of our ensemble-based energy optimization approach with the classical path-directed searches. We compare `RNA-ensign` with `RNAinverse`, `NUPACK`, and, when possible, `RNA-SSD` and `INFO-RNA`. Nevertheless, `RNAinverse` must be seen as the most fair and instructive comparison as it is the only path-directed algorithm that decouples the initialization (i.e. the seed) from the optimization strategy and that uses the same stopping criterion as `RNA-ensign`.

We show that our global search approach has several attractive features: it is successful significantly more often, and produces sequences that attain the desired structure with higher probability and lower entropy, than those output by classical local search methods such as `RNAinverse`. Importantly, these results are achieved regardless of the choice of seed or target structure and require few mutations. Our results are in agreement with seminal studies on RNA sequence-structure maps [68, 64], which showed that neutral networks of low-structured RNA secondary structures are fragmented and thus can be hard to reach with local search approaches. Since our ensemble-based strategy does not rely on the existence of paths in the evolutionary landscape, it can circumvent these difficulties and offer a reasonable alternative for designing RNA sequences for the most challenging target structures.

# Bibliographic note

This part of the thesis is closely based on our paper [45].

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 12

# Materials and Methods

## 12.1 Overview of algorithm

### 12.1.1 The low-energy ensemble of a structure.

Let $S^*$ be a fixed target structure of length $n$. The *low-energy* ensemble of $S^*$ consists of sequences $w$ that can fold to $S^*$ with each such sequence being assigned a certain probability. The probability of a sequence $w$ is proportional to $e^{-E/RT}$, where $E$ is the energy of $w$ when folding to $S^*$. Here, $R$ is the gas constant, and $T$ is the absolute temperature. The constant of proportionality is the sum of the above quantities over all sequences that can fold to $S^*$.

Using our `RNAmutants` algorithm [75, 76], we can sample, in *polynomial time and space*, sequences from the low-energy ensemble of a given $S^*$ (a brute force approach would result in an exponential time algorithm). This is done by setting $S^*$ as a structural constraint when invoking the program.

In this paper, we will in fact be concerned with the low-energy ensemble of $S^*$ *around a certain seed sequence* $a_0$ (which we will also call the mutant ensemble). This involves sampling $k$-mutants of $a_0$ (i.e., sequences differing from $a_0$ in exactly $k$ places) with probabilities proportional to the quantities above (we get the constant of proportionality by summing only over $k$-mutants).

The samples from the low-energy ensemble around a given seed will be our can-

didate sequences in the design algorithm.

### 12.1.2 Sampling from a structure's sequence ensemble.

To motivate our ensemble-based design approach, we first examine how our sequence search technique (ensemble sampling) differs from sequences sampled uniformly at random from those sequences that can fold to our structure. To this end, we randomly select two RNA secondary structures (of 47 and 61 nucleotides) from the `RNA STRAND` database [8], and sample one hundred $k$-mutants of a random seed (i.e., differing from the seed by $k$ point mutations) for each structure, both (a) uniformly from all $k$-mutants that can fold to the target structure, and (b) with weight corresponding to the probability of the sequence in the ensemble of $k$-mutants folding to the given structure. We then compute the probability that each sequence folds into the target *structure* in the *sequence's* Boltzmann ensemble.

Figure 12-1 shows the probability of the structure in the ensemble of each sampled sequence, organized by the distance from the seed (i.e., the number of point mutations). We clearly see that sequences generated from the low-energy ensemble occur with much higher probabilities than those generated uniformly at random. Further, by allowing for a higher distance from the seed, we increase the probabilities of the energy-favorable samples in a dramatic fashion. While this is certainly not surprising, it helps give motivation for our approach: it is reasonable to expect that in a significant portion of samples, the desired structure will be the most probable one, and thus, we will find a sequence designing it by looking at enough samples.

We note that whether a structure has a high probability in a Boltzmann ensemble of a sequence is a different criterion from it being the MFE structure for that sequence, since a sequence can have multiple sub-optimal structures with similar folding energies and thus probabilities. Ideally, we would like both to be the case. Therefore, in this study we also investigate the impact of our techniques on the base pairing entropy of the designed sequences.

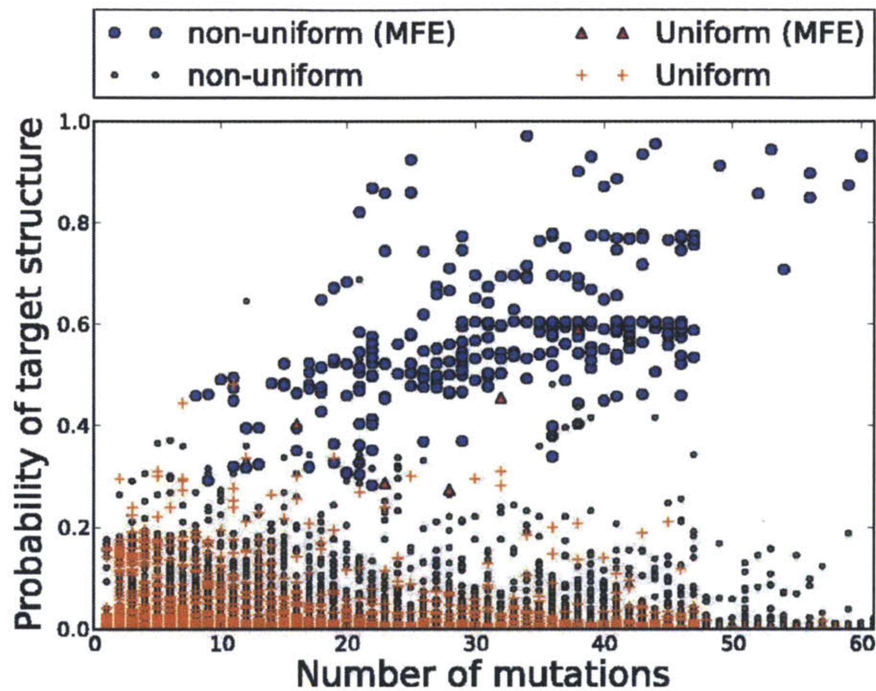



Figure 12-1: A scatter plot of the target structure probabilities on samples versus number of mutations from the seed. The "non-uniform" sequences (circles) are generated from the low-energy ensemble, while the "uniform" sequences (triangles and crosses) are generated uniformly at random from all $k$-mutants consistent with the structures. The sequences satisfying the MFE criterion are indicated with a large circle (non-uniform) and a triangle (uniform). In both cases, we sampled 100 $k$-mutants for each $k$.

### 12.1.3 Design algorithm

We now describe a design algorithm for a target structure $S^*$ consisting of $n$ nucleotides starting from a seed sequence $w$. It is a stochastic search that takes advantage of the structure constraint option in `RNAmutants`.

The stochastic algorithm proceeds by sampling one thousand $k$-point mutants of $w$ (for $k = 1, 2, \ldots, n$) from the low-energy ensemble of $S^*$. Then, for each $k$ in turn, we examine the samples one by one, and see if each achieves $S^*$ as its MFE structure. If for a given $k$ there are samples that achieve $S^*$ as the MFE structure, we return the one for which $S^*$ has highest probability. If we have not found a sequence with the desired properties, we report failure. In this way, we try to find a sequence achieving the MFE criterion, and which is also close to $w$.

We note that in our algorithms, the requirement that $S^*$ be achieved as an MFE structure is fairly arbitrary, but also quite natural since the MFE structure is the highest-probability structure. In particular, this criterion has the strong advantage of unifying the stopping criteria for the two primary methods evaluated in this paper (`RNA-ensign` and `RNAinverse`). It also enables us to generate solutions with few mutations of the seed that are good candidates for mutagenesis and synthetic biology experiments. It is worth noting that the `-Fp` option of `RNAinverse`, which optimizes the Boltzmann probability of the target structure, tends to produce better sequences (at least in terms of probability of the target structure), but this is achieved by optimizing sequences that *already* satisfy the MFE criterion and that are farther from the seed.

Our approach selects $k$-point mutants of $w$ optimizing the energy of the target structure. We hope that in this way it also optimizes its probability in the Boltzmann ensemble, until it emerges as the structure with highest probability. Of course, if the energies of other structures are also reduced substantially, this may not be the case (the probability of the target may not increase). However, it is reasonable to believe that in many cases it will, and as our results show, our method succeeds reasonably often.

## 12.2 Software selection

We aim to compare the advantages of local versus global search techniques for RNA secondary structure design. In addition, we also wish to evaluate the influence of the seed and target structure selection on the performance of each methodology. Thus, the programs used in this benchmark must (i) allow us to use any arbitrary seed sequence, and (ii) use the same stopping criteria (i.e. we stop once we have found a sequence that achieves the target structure as its MFE structure).

Under these constraints, only `RNAinverse` satisfies all our criteria. For the sake of completeness, we also provide the results achieved by `NUPACK` (the latter does not use the same stopping criteria), `RNA-SSD` and `INFO-RNA` (these two programs do not use the same stopping criterion and fully integrate the choice of the seed in their methodology). Nonetheless, to avoid any confusion, we will intentionally discuss the performance of these programs separately.

We remark that currently `RNAmutants`, which we use in `RNA-ensign`, does not handle dangling end energies. The `RNAinverse` and `NUPACK` programs allow us to disable the dangling end contribution and thus to match our energy model. On the other hand, `RNA-SSD` and `INFO-RNA` do not allow this, and we use their default energy function to compute the MFE energy structures and their probabilities. A somewhat unfortunate consequence is that given a sequence the MFE structure assessed by the energy functions used by `RNA-ensign`, `RNAinverse`, and `NUPACK` on the one hand and `RNA-SSD` and `INFO-RNA` on the other may be different. However, we do not expect this to significantly bias our analysis and conclusions.

## 12.3 Dataset of random target structures and seed sequences

We created a random test set of artificial target secondary structures and seed sequences of size 30nt, 40nt, 50nt, and 60nt. In order to perform a rational random generation of realistic secondary structures, we used the weighted context-free gram-

mars introduced by A. Denise *et al.* [18]. This formalism associates weights to terminal symbols in a context-free grammar, and the weight of a word is obtained multiplicatively. This induces a Boltzmann-like distribution on each subset of words of fixed size generated by the grammar. Efficient random generation algorithms, in quadratic time and memory, based on the so-called recursive method [78], can then be used to draw words from the weighted distribution [18]. It is worth noting that any two structures having the same distribution are being assigned equal probabilities in the weighted distribution, so that the uniform distribution is a special case (unit weights) of the weighted one. The addition of weights *shifts* the expectations of the numbers of occurrences, allowing one to gain control in a flexible manner (each structure remains possible) over the average profile of sampled words.

We modeled secondary structures using a grammar, independently found by M. Nebel [56] and Y. Ponty [60], that uses distinct terminal symbols to mark each occurrence of structural features (bulges, helices, internal loops, ...) and their content, allowing one to adjust their average lengths. We focused on a subset of features that is most essential to the overall topology of secondary structures: number of paired bases, number of helices, number of multiloops and number of bases appearing in multiloops. We analyzed this set of features on a set of native secondary structures from D. Mathews *et al.* [53] through systematic annotation. We used our optimizer `GrgFreqs` [18] to compute a set of weights such that the expected values for the features among sampled structures matches that of native structures. Finally, we used `GenRGenS` [61] to draw structures from the weighted ensemble.

We chose sets of seed sequences that evaluate the effects of the guanine/cytosine (`GC`) and purine (`AG`) contents. To this end, for each structure, and for each pair $(x, y)$, where both $x$ and $y$ come from $\{10\%, 20\%, \ldots, 90\%\}$, we generated seeds with `C+G` content of $x$ and `A+G` content of $y$. For each structure and each such $(x, y)$ (of which there are 81 choices) we generated 20 seeds, for a total of 1620 seeds per structure. We then used the sample sequences as seeds for our design algorithm, as well as for `RNAinverse`.

## 12.4 Dataset of known secondary structures

We built a complementary dataset of known secondary structures. We extracted all secondary structures without pseudo-knots with size up to 100 bases from the RNA STRAND database [8]. This resulted in a set of 396 targets with many similar structures. We clustered these structures into 50 classes using a single linkage method with the full tree edit distance implemented in RNAdistance [33]. This combination of clustering method, distance and cluster separation produced the best results we have been able to obtain. The final dataset contains 50 sequences of sizes ranging from 22 to 100 nucleotides and is available at http://csb.cs.mcgill.ca/RNAensign.

## 12.5 Structure and sequence analysis

### 12.5.1 Characterizing sequences.

First, we characterized the sequences (seeds and designed sequences) by their C+G content, as well as their purine (A+G) content. Since the thermodynamically advantageous effect of base-pair stacking in RNA helices is more pronounced with C≡G base pairs, sequences with higher stem C+G content tend to be more stable, and we reasoned that naturally arising structures with higher contiguity would also tend to have higher C+G content. Purine content, on the other hand, is a proxy for how many base-pairing opportunities the sequence provides: since a purine cannot base-pair with itself, very low and high A+G content means that relatively few base-pair combinations are possible and, compared with medium-A+G content sequences, relatively few structures can be formed.

### 12.5.2 Characterizing structures.

We tested the performance of our algorithm based on inherent thermodynamic stability offered by the target's structural motifs (i.e., stability of the structure without explicit reference to a sequence attaining the structure). Numerous motifs affect stability, and we selected one natural feature to study, namely the fraction of stacking

base pairs. Base pair stacking stabilizes the structure, and so our measure is a natural proxy of inherent stability.

### 12.5.3 Evaluation of performance

We use several metrics to estimate quality of a solution and the performance of the algorithms. We estimate the fitness of a sequence $w$ for a target secondary structure $T$ using (i) the Boltzmann probability of the target structure for the sequence defined and (ii) the normalized Shannon entropy of the base pairing probabilities [25]. The former assesses the likelihood of the target on the sequence, while low entropy values ensure that there are few competing structures in the energy landscape.

We also report the success rate and the number of mutations between the seed and the solution. The latter criterion is often important in synthetic biology studies [52, 48, 16], where one often wants to change a molecule's folding properties while perturbing the biological system as little as possible.

### 12.5.4 The challenges of fair comparison

One thing that we should note is that it is extremely difficult to get a fair comparison between various design methodologies, since the programs often have different goals. Even when comparing algorithms with similar objectives, e.g. finding a sequence satisfying the MFE criterion for a target structure and which is close to the seed, the methods have different tradeoffs of various desired properties, such as running time.

In our study, we put great effort into making our comparisons as complete and fair as possible, as we will see below. For example, we notice that `RNA-ensign` produces sequences with greater stability than those obtained by other methods, but the other methods are significantly faster. As a result, we compare the results of `RNA-ensign` to the best results of repeated runs of the other methods, where the number of repetitions is chosen so that the total time is comparable to the running time of `RNA-ensign`. Additionally, there are numerous instances where we modify `RNA-ensign` to make it better conform to the objectives of other programs. By doing a careful study, we

hope to present as fair a comparison as possible, given the substantial limitations.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 13

# Results and Discussion

We compare RNA-ensign with existing approaches and show that our method offers better success rates and more stable structures, regardless of the choice of the seed or target structure. In our experiments, only NUPACK outperforms our method on the specific criterion of the target structure stability. However, we show that by relaxing the stopping criterion used in RNA-ensign we can, in turn, achieve more stable structures than NUPACK.

## 13.1 Influence of the seed

Here we provide the first quantitative analysis of the influence of the nucleotide composition of the seed on the search algorithm's performance, as well as their impact on designed sequences. The $x$ and $y$ axis of the heat maps represent the A+G content and C+G content of the seeds. As mentioned earlier, we will discuss NUPACK separately as, unlike RNA-ensign and RNAinverse, it does not stop its optimization once the MFE criterion is achieved.

### 13.1.1 Impact on success rate

We start our analysis by looking at the success ratio of each program (i.e., the number of seeds producing sequences that fold into the target structure). We show our results

in the first row of Figure 13-1. Here, we observe a striking difference between the two methods. `RNA-ensign` clearly outperforms `RNAinverse` in all cases. While the success rates of `RNAinverse` vary between 0.4 and 0.8, the latter in rare cases (low `C+G` content and extreme values of the `A+G` content), `RNA-ensign` *uniformly* achieves a success rate of 0.9. The most significant difference occurs for seeds with high `C+G` content and medium `A+G` content. In this region of the sequence composition landscape, `RNAinverse` performs poorly (below 0.5) while `RNA-ensign` achieves a success rate of 0.9. It turns out that this region also corresponds to the seeds requiring more mutations to produce a sequence achieving the target structure (see Figure 13-1(g)). This insight could suggest that, particularly from these seeds but most likely for the others as well, `RNA-ensign` explores a different region of the mutational landscape, one that is more prone to contain sequences that fold into the desired structure. This exploration of a diverse mutational landscape is one motivation for using our method.

Compared to `RNAinverse`, `NUPACK` performs relatively well and does not seem significantly affected by the nucleotide composition of the seed. However, its performance (`NUPACK` exhibits a success rate oscillating between 0.7 and 0.8) remains lower than that obtained by `RNA-ensign`.

### 13.1.2 Impact on target probability.

We observe here that the choice of seed affects the quality and behavior of the design methods. First, we investigate if this choice has an influence on the thermodynamical stability of the target structure for the designed sequences (for our purposes, its "quality"). Our results, shown in the second row of Figure 13-1, demonstrate that the sequences designed with `RNA-ensign` are more stable (ensemble folding probabilities ranging from $\approx 0.4$ to $\approx 0.7$) than those obtained with `RNAinverse` (ensemble probabilities between $\approx 0.3$ and $\approx 0.5$). `NUPACK` appears to produce more stable structures (probabilities varying between $\approx 0.7$ and $\approx 0.8$) and seems less dependent on the seed. However as we will see, these results come with drawbacks.

The `A+G` content of the seeds has a strong influence on the quality of the designed sequences produced by `RNA-ensign` (see Figure 13-1(d)): medium `A+G` content val-

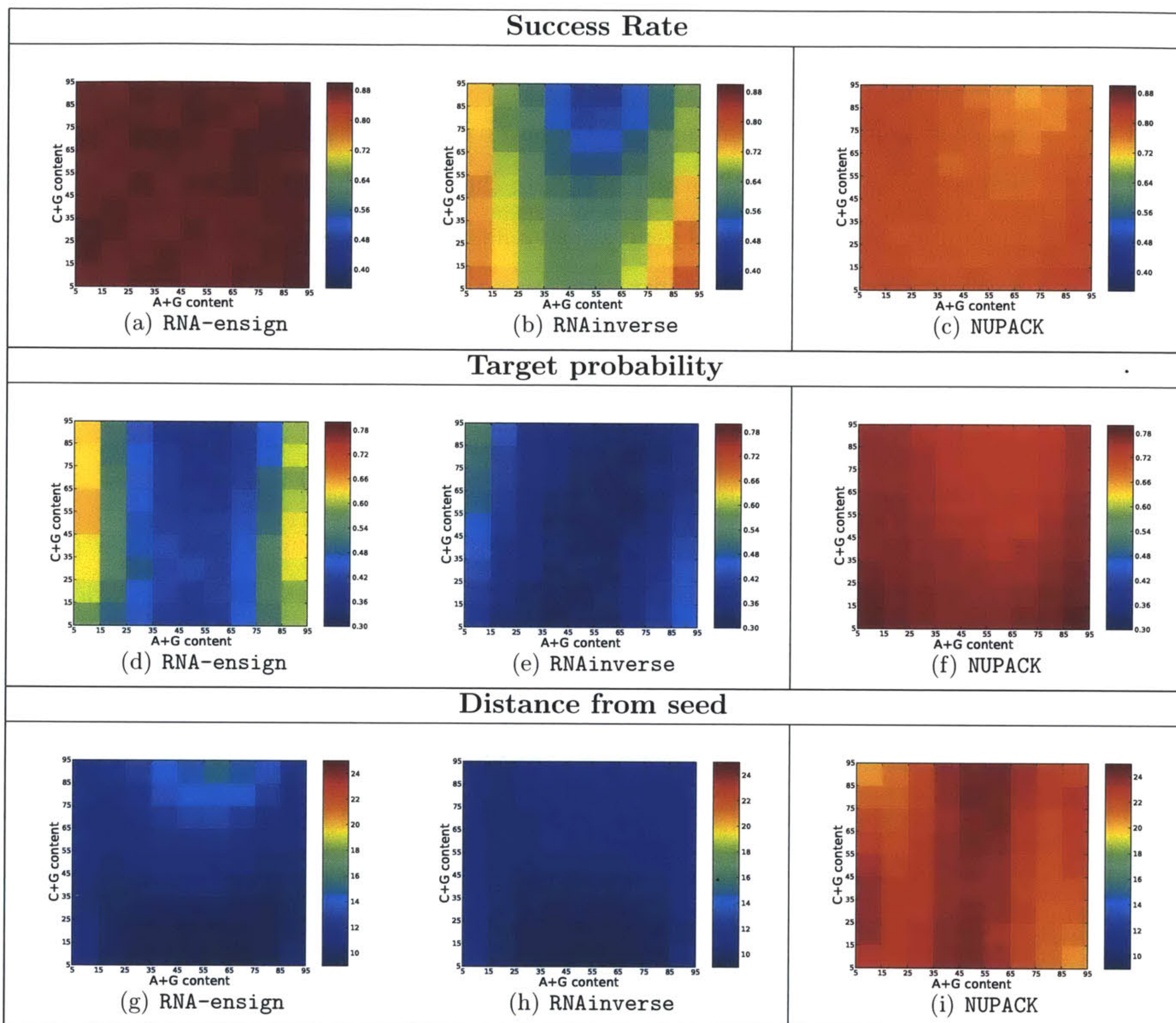



Figure 13-1: Evaluation of the influence of the nucleotide composition of the seeds on `RNA-ensign` (first column), `RNAinverse` (second column), and `NUPACK` (third column). The $x$ and $y$ axis represent respectively the `A+G` content and `C+G` content of the sequences. The first row shows the success rates of each method; the second row shows the probability of the target structure for the designed sequences; the third row reports the Hamming distance (i.e., number of mutations) between the seed and the designed sequence.

ues produce sequences with lower ensemble probabilities, while extreme ranges of the A+G content give highly thermodynamically stable sequences. This is likely a consequence of combinatorics: extreme ends of the A+G content spectrum mean combinatorially fewer opportunities for base pairing, and therefore fewer possible structures for each sequence. Because there are fewer possible structures, a "good" structure will comprise a much higher percentage of the folding ensemble. This gradient is less pronounced with sequences generated by RNAinverse, which do not reach the same level of thermodynamic stability even for extreme A+G content values. Moreover, the distribution for RNAinverse follows a slightly different pattern, where the least stable sequences lie along the diagonal of equal A+G and C+G content.

The impact of the nucleotide composition of the seed on the base pair entropy is similar to what has been observed with the target probability. Overall, NUPACK shows better performance (i.e. lower entropy), and extreme A+G contents tend to significantly reduce the entropy values of RNA-ensign and RNAinverse solutions (see supplementary material in [45]).

### 13.1.3 Impact on distance between seed and solution.

Our next experiments, shown in the third row of Figure 13-1, illustrate how the choice of seed influences the number of mutations performed to reach a solution (i.e., the designed sequence) under each search method. Overall, both methods perform similarly with an average number of mutations (over all sequence sizes) of approximately 10. The exception is the region of high C+G content and medium A+G content, which, on average, requires almost 15 mutations with RNA-ensign and 12–13 with RNAinverse. This may be because higher C+G content means that triple hydrogen C≡G bonds lead to lower folding energies and "democratize" the folding ensemble by more effectively competing against folding energies of loop structures; in a more diverse ensemble, RNA-ensign is less likely to sample a favorable structure and must move on to a higher mutation distance. RNAinverse, which is much less demanding when it comes to the energetic properties of the designed sequence (see Figure 13-1(e)), settles for a less stable structure at a lower mutation distance; thus the high-C+G content effect,

though visible, is much less dramatic.

It is worth noting that NUPACK disadvantageously requires almost twice as many mutations as RNA-ensign and RNAinverse. This is most likely a consequence of the different stopping criterion and a necessity to achieve the highly stable sequences observed in Figure 13-1(f). As we will see below, because NUPACK produces a sequence vastly different from the seed, the nucleotide composition of the solutions will also be affected.

### 13.1.4 Nucleotide composition of designed sequences.

Finally, we complete this analysis by looking at the nucleotide composition of the designed sequences. We refer to [45, Figure 2(j-l)]. The sequences generated by RNA-ensign and RNAinverse appear to have similar A+G contents. Both methods have a slight bias toward well-balanced Purine compositions. However, their influence on the C+G content differs. While RNAinverse has a tendency to produce sequences with low C+G content (to approximately 35%), RNA-ensign tends to increase this value (approximately 60%). Nevertheless, in both cases, the influence of the method on the nucleotide composition seems minor.

In contrast, NUPACK has a stronger influence on the final nucleotide composition. Indeed, the method has a clear tendency to generate sequences with a C+G content between 45% and 65%. It follows that the choice of the seed cannot be reliably used to control the nucleotide composition of the designed sequences and that NUPACK provides less diverse solutions.

## 13.2 Influence of the target structure

We now discuss the effect of the target structure on the performance of the various methods. In particular, we focus on the stability of the designed sequence on the structures, as well as the success rates. Since this benchmark does not depend on the seed but only on the target structure, we also include RNA-SSD and INFO-RNA in this test. However, their results should be discussed with caution, since the results of

RNAinverse and RNA-ensign are averaged over all seeds while RNA-SSD and INFO-RNA automatically select favorable seed sequences.

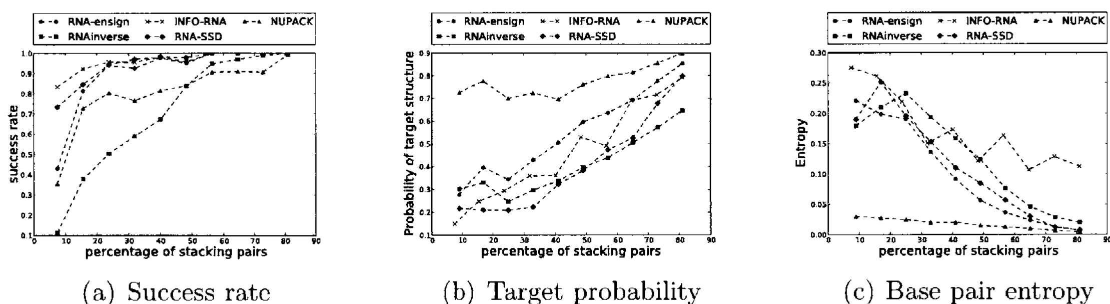
We characterize the target structures by the percentage of stacking pairs they contain. This is a natural measure in our context since the energy calculation of the nearest-neighbor energy model we use [53] is based on the energetic contribution of the stacking of base pairs. We can also characterize secondary structures by other local motifs such as hairpins, bulges, internal loops and multiloops. However, in this study these parameters did not exhibit clear correlations (data not shown).

### 13.2.1 Impact on success rate.

The most significant discrepancy between the performances of all methods with respect to the target structures relates to the success rates. We show in Figure 13-2(a) how the percentage of stacks in the target structure correlates with the ratio of successful designs. Remarkably, RNA-ensign clearly outperforms RNAinverse for target structures with a low percentage of stacking pairs. This observation is important because these structures can be quite irregular (i.e., including bulges, internal loops and/or multiloops) and are precisely those that are most difficult to design. Even for targets with only 20% stacking base pairs, RNA-ensign is able to reach a success rate of 0.9. In contrast, RNAinverse requires targets with at least 50% of base pairs stacking to reach the same success rate.

This phenomenon reemphasizes the benefits of an ensemble approach to capture compensatory and epistasis effects in the mutational landscape. Indeed, the design of RNA secondary structures with few stacking pairs can only be achieved by combining several mutations with sometimes contradictory effects [15]. We know from previous studies that the neutral network of these structures is highly fragmented [68, 64] and difficult to reach with a search guided by phenotype (i.e. MFE structure). Furthermore, the performance of RNA-SSD and INFO-RNA suggests that local search heuristics are subject to optimization and thus could benefit from the results reported in this study.

This experiment (i.e. Figure 13-2) also shows the tremendous progress achieved by




(a) Success rate (b) Target probability (c) Base pair entropy

Figure 13-2: Evaluation of the influence of (random) target structures. The $x$-axis represents the percentage of stacks in the target structure. In figure (a), we show how this parameter impacts the success rates of the programs. In the figure (b), we depict the probability of the target structure for the designed sequence. In figure (c), we show the influence on the base pairing entropy.

the path-directed approaches since `RNAinverse`. Indeed, `RNA-SSD` and `INFO-RNA` both perform very well on unstructured RNA targets and their success rates even exceed that of `RNA-ensign`. Noticeably, `NUPACK` does not offer the same level of performance, although it still does reasonably well (approximately 80% while `RNA-ensign`, `INFO-RNA` and `RNA-SSD` easily reach 95%).

### 13.2.2 Impact on target probability and base pair entropy.

In Figure 13-2(b), we show how the stability of the target structures on designed sequences correlates with the percentage of stacks. For all methods except `NUPACK`, we observe a linear correlation with a similar slopes (above 20% of stacks). This indicates that the quality of the designed sequences is dependent of the number of stacks in the target structure, and that all methods scale similarly. However, we also observe that `RNA-ensign` outperforms `RNAinverse`, `RNA-SSD` and `INFO-RNA` by a constant factor (i.e., higher affine constant). It follows that the gain obtained by `RNA-ensign` versus these programs is independent of the target structure.

It is worth noting that `INFO-RNA` and `RNA-SSD` have only slightly better performance than `RNAinverse` in this regard (in contrast `RNA-ensign` clearly outperforms the latter). However as we have seen earlier, the main benefits of `INFO-RNA` and `RNA-SSD` reside in their success rates. Interestingly, `NUPACK` exhibits a different be-

havior than other methods. Despite a lower success rate, the sequences produced are significantly more stable than those obtained with other software, and the quality of the structures does not seem to affect its performance.

Similarly, Figure 13-2(c) shows that RNA-ensign returns sequences with better (i.e. lower) base pair entropy values than RNAinverse, RNA-SSD and INFO-RNA. It also shows that NUPACK clearly outperforms all other software for this test.

## 13.3 Alternate stopping criterion

As we have remarked, NUPACK often produces sequences with higher target structure probabilities, at the expense of lower success rates and finding designed sequences that are farther away from the seed. These differences are primarily due to the use of a different stopping criterion. We decided to investigate this case and changed our stopping criterion. Rather than stop the search once we have found a sample that achieves the MFE criterion, we considered 1000 samples for all possible numbers of mutation (i.e. 1 up to the length of the seed), and selected the sample achieving the MFE criterion that satisfied some other desirable property. More specifically, we implemented two variants. The first one (called RNA-ensign-P) selects the mutant with the highest Boltzmann probability of attaining the target structure, and the second one (called RNA-ensign-S) selects the mutant with the lowest entropy. Similarly, we note that using the "-Fp" option, RNAinverse can also return the highest probability sequence found during a local search. It is worth noting that, in both of these algorithms, we are not concerned with finding a designed sequence that is close to the seed.

We tested all these variants, as well as the standard RNA-ensign, RNAinverse and NUPACK algorithms on the RNA STRAND dataset. For each target structure, we used 10 random seeds.

Figure 13-3(a) shows the Boltzmann probability of the solution vs. the number of stacks in the secondary structure target. It reveals that RNAinverse-Fp followed by RNA-ensign-P and RNA-ensign-S outperform other methods. RNAinverse-Fp

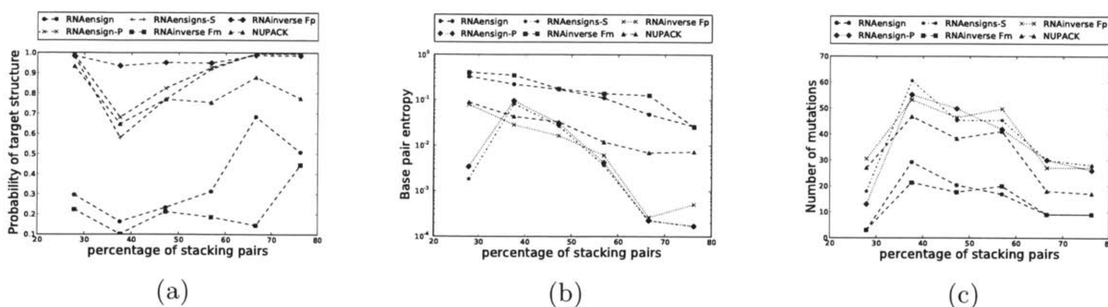



Figure 13-3: Comparison of the probability-optimized RNA-ensign with NUPACK and the original version of RNA-ensign. This benchmark has been realized on 100 secondary structure targets of length 60 with 10 random seeds for each target. The $x$-axis represents the number of stacks in the target structure. In figure (a), the $y$-axis represents the probability of the target structure on the sequence. In figure (b), the $y$-axis indicates the entropy of the solutions using a log-scale. In figure (c), the $y$-axis reports the number of mutations between the seed and the solution.

outperforms RNA-ensign for target structures with 35 to 55% of stacks. In fact, these targets are characterized by long bulges and internal loops. All methods but RNAinverse-Fp are affected to various degrees by this phenomenon. The impact of stacking pairs on the entropy is shown in Figure 13-3(b). Here, RNA-ensign-S and RNA-ensign-P globally outperform all other methods. Only RNAinverse-Fp manages to match the performance of RNA-ensign-P and RNA-ensign-S above 50% of stacks. RNA-ensign-P and RNA-ensign-S remain better for the most difficult cases. Noticeably, NUPACK behaves differently from the probability and entropy optimized variants of RNA-ensign and RNAinverse. Higher percentages of stacking pairs (above 60%) seem to significantly reduce the entropy of the solutions returned by RNA-ensign-P, RNA-ensign-S and RNAinverse-Fp, while NUPACK scales like the MFE variants of RNA-ensign and RNAinverse. Unsurprisingly, the numbers of mutations required by the optimized variants of RNA-ensign and RNAinverse increase significantly and exceed the values required by NUPACK (See Figure 13-3(c)).

## 13.4 Running time and multiple runs

For molecules of 40nt, our design method took about a minute per structure/seed input to complete on a 3.33 GHz CPU; for 60nt molecules, runtime grew to ca. 20 minutes and used 300 MB of memory. We investigated the runtime and compared the performance of `RNA-ensign` to other local search approaches of the first generation (`RNAinverse`) and second generation (`NUPACK`). In particular, we ran `RNAinverse` 10000 times and `NUPACK` 100 times on the RNA-STRAND dataset using random seeds (`C+G` content and `A+G` content of 50%). These settings enabled us to have comparable runtimes. For each experiment, we computed the Boltzmann probability of the target structure and the base pair entropy of the best solution found over all runs, and reported the total running time. Our results are shown in Table 13.1. We split our dataset in 3 categories based on the length of the structure (small: 40 nucleotides or fewer; medium: between 41 and 80 nucleotides; large: 81 nucleotides or more).

On small targets, our data show that with a similar amount of time the global search approach outperforms the local search method `RNAinverse`, while the results are reversed on medium size targets. Nonetheless, for the longest structures it appears that `RNA-ensign` tends to produce better solutions than `RNAinverse`. To understand this, we note that small targets are single stem structures that can be easily stabilized by improving stacking energies—a strategy matching the principles of our objective function. When the structures grow and become more sophisticated (i.e. with multi-loops), local search methods apply efficient heuristics to accommodate the presence of complex motifs. This strategy could eventually increase the folding energy of the mutants and therefore is not captured by `RNA-ensign`. However, on longer targets (80 nucleotides or more), heuristics become less efficient in handling the combinatorial explosion of the number of candidate sequences. As a consequence, these heuristics have more chances to drive the mutants to sub-optimal regions of the sequence landscape. On the other hand, a global search approach becomes more competitive because searches distant from the seed are not influenced by potentially misleading in-

| Length | Probability | | | | Entropy | | | | Time (sec.) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | A | B | C | D | A | B | C | D |
| 0-40 | 0.69 | 0.65 | 0.60 | 0.97 | 0.056 | 0.051 | 0.065 | 0.003 | 62 | 28 | 61 | 27 |
| 41-80 | 0.35 | 0.21 | 0.53 | 0.89 | 0.148 | 0.157 | 0.100 | 0.008 | 1883 | 742 | 711 | 8973 |
| 81+ | 0.40 | 0.30 | 0.29 | 0.93 | 0.062 | 0.147 | 0.125 | 0.006 | 9332 | 2434 | 1269 | 2920 |

Table 13.1: Comparison of `RNA-ensign` (column A) with multiple runs of `RNAinverse` (C) and `NUPACK` (D). We ran `RNAinverse` 10000 times and `NUPACK` 100 times on the RNA-STRAND dataset using random seeds (`C+G` content and `A+G` content of 50%) and reported the Boltzmann probability of the target structure and the base pair entropy of the best solution found over all runs. The total running time is indicated in seconds in the last columns. We also included the performance achieved by `RNA-ensign` with a number of mutations bounded by 50% of the number of nucleotides (B).

termediate choices. In all cases, it is worth noting that `NUPACK`, with improved search heuristics and stopping criteria, offers excellent performance with multiple runs. This suggests that second generation methods of global search approaches could drastically improve as well.

We completed this study by running a version of `RNA-ensign` with a number of mutations bounded by 50% of the number of nucleotides. With a minor loss of performance that does not alter the overall trends discussed above, this variant drastically improves the running time of `RNA-ensign`. To this, we must add that once the partition function has been calculated with `RNAmutants`, the cost for sampling new structures is cheap (i.e., $O(n^2)$ in the worst case with the current implementation). Thus, the size of the search that has been heuristically fixed at 1000 samples in this work, can be easily increased to improve `RNA-ensign` performance with minimal changes to the running time.

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 14

# Discussion

In this work, we have demonstrated that *ensemble-based* approaches provide a good alternative to *stochastic local search* methods for the RNA secondary structure design problem. Our results suggest that our techniques have the potential to improve several aspects of classical path-directed methods. In particular, we have shown that our strategy is efficient on target structures with few stacking base pairs and the influence of the choice of the seed on the success rate is minimal. Our methodology also appears to produce more stable sequences and has a limited impact on the final nucleotide composition.

In a sense, our approach is a dual to McCaskill's classical algorithm for RNA folding [54]. That algorithm can efficiently sample possible secondary structures for a given sequence with the correct Boltzmann probabilities (roughly, those where lower energy structures are more likely). In this way, it allows us to see what structures the sequence is likely to fold into. In our approach, we reverse this logic, and try to find sequences for which a given (fixed) structure has a favorable energy. We do this using a dynamic programming approach similar to the one used by McCaskill. The most general `RNAmutants` program is in a very real sense a substantial generalization of McCaskill's algorithm [75], and our particular application presented here is one consequence of this generalization.

It is worth noting that `NUPACK` appears to produce more stable sequences than other implementations of the local search approach. But these benefits come with

noticeable disadvantages: the designed sequences are uniformly far from the seed in terms of Hamming distance and the final nucleotide composition has a strong bias. Consequently, in their framework the seeds cannot be used to control characteristics of the designed sequences such as the `C+G` content or to reengineer molecular systems with tight constraints on sequence deviation.

We also show that the stability of the target structure for sequences designed with `RNA-ensign` can still be improved. We relaxed the stopping criterion and demonstrated that, at the price of increased sequence deviations, our strategy can produce more stable structures than `NUPACK` and match the performance of the probability optimized variant of `RNAinverse`. Nonetheless, since the computational complexity of our method is bounded by the number of mutations it performs, this variant may be restricted to the design of small RNA elements such as those used in [58, 20, 52]. More importantly, beyond a strict numerical comparison, this result shows that `RNA-ensign` offers new perspectives for improved RNA secondary structure design algorithms.

Our results can also be compared to those obtained by R. Dirks *et al.* [19], who reported that a local search approach to design using only an energy-based optimization approach performs poorly. In contrast, our data suggest that an ensemble-based approach implementing similar objective functions should reverse this finding.

Due to its current time and memory requirements, thus far, our method is limited to the design of small RNAs (150 nucleotides or less). This limitation does not strike us as a major drawback since the sizes of most of the structural RNAs we aim to design fall below this limit. In the future, we envision hybrid approaches that will take advantage of both strategies, the classical local search methodology for its speed and versatility, and our ensemble-based approach for its capacity to generate high quality sequences even on hard instances of the problem.

Finally, our ensemble-based method could also benefit from recent `RNAmutants` developments [77] that enable us to explore specific regions of the mutational landscape. These techniques could be applied to account for external constraints on the sequence composition (e.g. AT-rich thermophiles), improving the potential of our designed sequences to be active within realistic cellular contexts.

An implementation of the method and its variants described in this paper is publicly available at: `http://csb.cs.mcgill.ca/RNAmutants`.

THIS PAGE INTENTIONALLY LEFT BLANK

# Bibliography


[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008.

[2] Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281, New York, NY, USA, 2001. ACM.

[3] Rosalía Aguirre-Hernández, Holger H Hoos, and Anne Condon. Computational RNA secondary structure design: empirical complexity and improved methods. *BMC Bioinformatics*, 8:34, 2007.

[4] K.J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467. SIAM, 2012.

[5] Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, ICALP '09, pages 328–338, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. ACM.

[7] David J Aldous. A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 3:450 465, 1990.

[8] Mirela Andronescu, Vera Bereg, Holger H Hoos, and Anne Condon. RNA STRAND: the RNA secondary structure and statistical analysis database. *BMC Bioinformatics*, 9:340, 2008.

[9] Mirela Andronescu, Anthony P Fejes, Frank Hutter, Holger H Hoos, and Anne Condon. A new algorithm for RNA secondary structure design. *J Mol Biol*, 336(3):607–24, Feb 2004.

[10] Assaf Avihoo, Alexander Churkin, and Danny Barash. Rnaexinv: An extended inverse RNA folding from shape and physical attributes to sequences. *BMC Bioinformatics*, 12(1):319, Aug 2011.

[11] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *STOC '09: Proceedings of the 41st Annual ACM symposium on Theory of Computing*, pages 255–262, New York, NY, USA, 2009. ACM.

[12] András A. Benczúr and David R. Karger. Approximating $s$-$t$ minimum cuts in $O(n^2)$ time. In *Proceedings of the Twenty-Eighth Annual ACM symposium on Theory of Computing*, STOC '96, pages 47–55, New York, NY, USA, 1996. ACM.

[13] Anke Busch and Rolf Backofen. INFO-RNA–a fast approach to inverse RNA folding. *Bioinformatics*, 22(15):1823–31, Aug 2006.

[14] P. Christiano, J.A. Kelner, A. Madry, D.A. Spielman, and S.H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 273–282. ACM, 2011.

[15] Matthew C Cowperthwaite and Lauren Ancel Meyers. How mutational networks shape evolution: Lessons from RNA models. *Annual Review Ecology Evolution and Systematics*, 2007.

[16] Stephanie J Culler, Kevin G Hoff, and Christina D Smolke. Reprogramming cellular behavior with RNA controllers responsive to endogenous proteins. *Science*, 330(6008):1251–5, Nov 2010.

[17] Denny C. Dai, Herbert H. Tsang, and Kay C. Wiese. rnaDesign: Local search for RNA secondary structure design. In *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, 2009.

[18] Alain Denise, Yann Ponty, and Michel Termier. Controlled non uniform random generation of decomposable structures. *Journal of Theoretical Computer Science (TCS)*, 411(40-42):3527–3552, 2010. 68R05;68R15.

[19] Robert M Dirks, Milo Lin, Erik Winfree, and Niles A Pierce. Paradigms for computational nucleic acid design. *Nucleic Acids Res*, 32(4):1392–403, 2004.

[20] Neil Dixon, John N Duncan, Torsten Geerlings, Mark S Dunstan, John E G McCarthy, David Leys, and Jason Micklefield. Reengineering orthogonally selective riboswitches. *Proc Natl Acad Sci U S A*, 107(7):2830–5, Feb 2010.

[21] Kishore J Doshi, Jamie J Cannone, Christian W Cobaugh, and Robin R Gutell. Evaluation of the suitability of free-energy minimization using nearest-neighbor energy parameters for rna secondary structure prediction. *BMC Bioinformatics*, 5:105, Aug 2004.

[22] B. Efron. Bootstrap methods: another look at the jackknife. *Annals of Statistics*, 7(1):1–26, 1979.

[23] B. Efron and R. Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science*, 1:54–75, 1986.

[24] Josep Fàbrega. Random walks on graphs. Available at `www.lirmm.fr/~sau/JCALM/Josep.pdf`.

[25] Eva Freyhult, Paul P Gardner, and Vincent Moulton. A comparison of RNA folding measures. *BMC Bioinformatics*, 6:241, 2005.

[26] P.A. Fujita, B. Rhead, A.S. Zweig, A.S. Hinrichs, D. Karolchik, M.S. Cline, M. Goldman, G.P. Barber, H. Clawson, A. Coelho, et al. The UCSC Genome Browser database: update 2011. *Nucleic Acids Research*, 39(suppl 1):D876–D882, 2011.

[27] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC '11: Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 71–80, 2011.

[28] J.H. Gillespie. *Population genetics: a concise guide*. Johns Hopkins University Press, 2004.

[29] A. Goel, M. Kapralov, and S. Khanna. Graph sparsification via refinement sampling. arXiv: 1004.4915 [cs.DS].

[30] Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012.

[31] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, May 1984.

[32] Nicholas Harvey. Lecture 11 notes for C&O 750: Randomized algorithms (Waterloo, Winter 2011). Available at `http://www.math.uwaterloo.ca/~harvey/W11`.

[33] I. L. Hofacker, W. Fontana, P. F. Stadler, S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie*, 125:167–188, 1994.

[34] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[35] Farren J Isaacs, Daniel J Dwyer, and James J Collins. RNA synthetic biology. *Nat Biotechnol*, 24(5):545–54, May 2006.

[36] A. Keinan, J.C. Mullikin, N. Patterson, and D. Reich. Measurement of the human allele frequency spectrum demonstrates greater genetic drift in East Asians than in Europeans. *Nature Genetics*, 39(10):1251–1255, 2007.

[37] Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. In Thomas Schwentick and Christoph Dürr, editors, *STACS*, volume 9 of *LIPIcs*, pages 440–451. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[38] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. *CoRR*, abs/1301.6628, 2013.

[39] Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for SDD matrices. arXiv: 1209.5821v1 [cs.DS].

[40] Ioannis Koutis, Alex Levin, and Richard Peng. Improved spectral sparsification and numerical algorithms for SDD matrices. In *Proceedings of the 29th Internaational Symposium on Theoretical Aspects of Computer Science*, STACS '12, pages 266–277.

[41] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, FOCS '10, 2010.

[42] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ solver for SDD linear systems. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, FOCS '11, 2011.

[43] Sita J Lange, Daniel Maticzka, Mathias Möhl, Joshua N Gagnon, Chris M Brown, and Rolf Backofen. Global or local? predicting secondary structure and accessibility in mRNAs. *Nucleic Acids Res*, Feb 2012.

[44] Gregory F. Lawler and Lester N. Coyle. *Lectures on contemporary probability*, volume 2 of *Student Mathematical Library*. American Mathematical Society, Providence, RI, 1999.

[45] A. Levin, M. Lis, Y. Ponty, C.W. O'Donnell, S. Devadas, B. Berger, and J. Waldispühl. A global sampling approach to designing and reengineering RNA secondary structures. *Nucleic Acids Research*, 2012.

[46] Alex Levin. Some variants on spectral sparsification. Unpublished manuscript.

[47] J.Z. Li, D.M. Absher, H. Tang, A.M. Southwick, A.M. Casto, S. Ramachandran, H.M. Cann, G.S. Barsh, M. Feldman, L.L. Cavalli-Sforza, et al. Worldwide human relationships inferred from genome-wide patterns of variation. *Science*, 319(5866):1100–1104, 2008.

[48] Joe C Liang, Ryan J Bloom, and Christina D Smolke. Engineering biological systems with synthetic RNA molecules. *Mol Cell*, 43(6):915–26, Sep 2011.

[49] Mark Lipson, Po-Ru Loh, Alex Levin, David Reich, Nick Patterson, and Bonnie Berger. Efficient moment-based inference of admixture parameters and sources of gene flow.

[50] Y. Lu, T. Genschoreck, S. Mallick, A. Ollmann, N. Patterson, Y. Zhan, T. Webster, and D. Reich. A SNP array for human population genetics studies. 2011. Presented at the 12th International Congress of Human Genetics/61st Annual Meeting of The American Society of Human Genetics, October 12, 2011, Montreal, Canada.

[51] Yontao Lu, Nick Patterson, Yiping Zhan, Swapan Mallick, and David Reich. *Technical design document for a SNP array that is optimized for population genetics*, 2011.

[52] Julius B Lucks, Lei Qi, Vivek K Mutalik, Denise Wang, and Adam P Arkin. Versatile RNA-sensing transcriptional regulators for engineering genetic networks. *Proc Natl Acad Sci U S A*, 108(21):8617–22, May 2011.

[53] D H Mathews, J Sabina, M Zuker, and D H Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J Mol Biol*, 288(5):911–40, May 1999.

[54] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105—1119, 1990.

[55] Alessandro Michienzi, Shirley Li, John A Zaia, and John J Rossi. A nucleolar TAR decoy inhibitor of HIV-1 replication. *Proc Natl Acad Sci U S A*, 99(22):14047–52, Oct 2002.

[56] M.E. Nebel. Identifying good predictions of RNA secondary structure. In *Pacific Symposium on Biocomputing*, volume 9, pages 423–434, 2004.

[57] M. Nei. *Molecular Evolutionary Genetics*. Columbia University Press, 1987.

[58] Yoko Nomura and Yohei Yokobayashi. Reengineering a natural riboswitch by dual genetic selection. *J Am Chem Soc*, 129(45):13814–5, Nov 2007.

[59] N.J. Patterson, P. Moorjani, Y. Luo, S. Mallick, N. Rohland, Y. Zhan, T. Genschoreck, T. Webster, and D. Reich. Ancient admixture in human history. *Genetics*, 2012.

[60] Y. Ponty. Etudes combinatoire et génération aléatoire des structures secondaires d'ARN. Master's thesis, Université Paris Sud, 2003. Mémoire de DEA.

[61] Yann Ponty, Michel Termier, and Alain Denise. Genrgens: software for generating random genomic sequences and structures. *Bioinformatics*, 22(12):1534–1535, Jun 2006.

[62] J.E. Pool, I. Hellmann, J.D. Jensen, and R. Nielsen. Population genetic inference from genomic sequence variation. *Genome research*, 20(3):291–300, 2010.

[63] D. Reich, K. Thangaraj, N. Patterson, A.L. Price, and L. Singh. Reconstructing Indian population history. *Nature*, 461(7263):489–494, 2009.

[64] C Reidys, P F Stadler, and P Schuster. Generic properties of combinatory maps: neutral networks of RNA secondary structures. *Bull Math Biol*, 59(2):339–397, Mar 1997.

[65] N.A. Rosenberg, J.K. Pritchard, J.L. Weber, H.M. Cann, K.K. Kidd, L.A. Zhivotovsky, and M.W. Feldman. Genetic structure of human populations. *Science*, 298(5602):2381–2385, 2002.

[66] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[67] Michael Schnall-Levin, Leonid Chindelevitch, and Bonnie Berger. Inverting the Viterbi algorithm: an abstract framework for structure design. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 904–911. ACM, 2008.

[68] P Schuster, W Fontana, P F Stadler, and I L Hofacker. From sequences to shapes and back: a case study in RNA secondary structures. *Proc Biol Sci*, 255(1344):279–284, Mar 1994.

[69] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the 40th Annual ACM symposium on Theory of Computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. ACM.

[70] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 81–90, New York, NY, USA, 2004. ACM.

[71] Nikhil Srivastava. Spectral sparsification and restricted invertibility.

[72] Luca Trevisan. Approximation algorithms for unique games. In *FOCS '05: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 5–34, 2005.

[73] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript.

[74] Roman Vershynin. A note on sums of independent random matrices after Ahlswede-Winter.

[75] Jerome Waldispühl, Srinivas Devadas, Bonnie Berger, and Peter Clote. Efficient algorithms for probing the RNA mutation landscape. *PLoS Comput Biol*, 4(8):e1000124, 2008.

[76] Jerome Waldispühl, Srinivas Devadas, Bonnie Berger, and Peter Clote. RNA-mutants: a web server to explore the mutational landscape of RNA secondary structures. *Nucleic Acids Res*, 37(Web Server issue):W281–6, Jul 2009.

[77] Jérôme Waldispühl and Yann Ponty. An unbiased adaptive sampling algorithm for the exploration of RNA mutational landscapes under evolutionary pressure. In *RECOMB*, pages 501–515, 2011.

[78] H. S. Wilf. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Advances in Mathematics*, 24:281–291, 1977.

[79] Joseph N Zadeh, Brian R Wolfe, and Niles A Pierce. Nucleic acid sequence design via efficient ensemble defect optimization. *J Comput Chem*, 32(3):439–52, Feb 2011.