

Graphical and Isoperimetric Perspectives on Sampling and Regularization

by

Edric Tam

Department of Statistical Science
Duke University

Defense Date: July 11, 2024

Approved:

David Dunson, Supervisor

Amy Herring

Peter Hoff

Jason Xu

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Statistical Science
in the Graduate School of Duke University
2024

ABSTRACT

Graphical and Isoperimetric Perspectives on Sampling and
Regularization

by

Edric Tam

Department of Statistical Science
Duke University

Defense Date: July 11, 2024

Approved:

David Dunson, Supervisor

Amy Herring

Peter Hoff

Jason Xu

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Statistical Science
in the Graduate School of Duke University
2024

Copyright © 2024 by Edric Tam
All rights reserved except the rights granted by the
<http://creativecommons.org/licenses/by-nc/3.0/us/> Creative Commons
Attribution-Noncommercial Licence

Abstract

Several recurring themes in modern Bayesian statistics and probabilistic machine learning are sampling, regularization and embeddings/latent variables. This thesis examines particular aspects of these themes from the perspective of concentration of measure, isoperimetry and spectral graph theory. These mathematical topics are intricately connected, and we often leverage related tools, such as Laplacians and concentration inequalities, in the theory and methods developed in this thesis.

From the sampling methodology end, we develop a novel spanning tree sampler called the fast-forwarded cover algorithm that improves upon the celebrated Aldous-Broder algorithm. The Aldous-Broder algorithm, and other related random-walk based approaches for spanning tree sampling, can be stuck when the underlying graph exhibits bottleneck structures. We show that our fast-forwarded cover algorithm can break free of bottlenecks that are arbitrarily small, while maintaining exact sampling guarantees. We demonstrate how our novel sampler can be applied to posterior sampling from Bayesian models with tree components.

For sampling theory, we investigate the statistical capacity of broad classes of deep generative models. We use concentration of measure techniques to show that there is a limitation to what kinds of distributions deep generative models can learn to sample from. In particular, for generative adversarial networks and variational autoencoders, under standard priors on the latent variables, only light tailed distributions can be learned. This fact is generalized to the manifold setting using tools

from concentration and geometry. We also provide extra quantification of these models' capacities based on functional inequalities. This debunks a popular belief about the learning capacity of deep generative models.

On the regularization side, we propose Fiedler regularization, a way to use a neural network's underlying graph structure to regularize itself. It can be used in the supervised learning setting, as well as more general settings such as for regularizing autoencoders. We give theoretical guarantees and empirical experimental results.

Last but not least, on the embeddings front, we propose a novel piece of methodology for graph comparison called embedded Laplacian distance. Typical methods for comparing graphs are computationally demanding, due to the discrete and combinatorial nature of graphs. We observe that for many applications where graph comparison is needed, an approximate comparison that captures the structures of the graphs suffice. We propose to compare graphs based on continuous representations that capture meaningful graph structure, namely spectral embeddings. Theoretical and empirical results are provided.

Dedication

To Mom, Dad, Venus and Edmund

Contents

Abstract	iv
List of Tables	xii
List of Figures	xiii
Acknowledgements	xiv
1 Introduction	1
2 Fiedler Regularization	4
2.1 Introduction	4
2.2 Spectral Graph Theory	7
2.2.1 Setup and Background	7
2.2.2 Graph Laplacian	8
2.2.3 Edge Expansion and Cheeger’s Inequality	9
2.3 Fiedler Regularization	11
2.3.1 Supervised Learning: Classification Setup	12
2.3.2 Neural Network Setup	12
2.3.3 Penalizing with Fiedler Value	13
2.3.4 Properties of the Fiedler Value	14

2.3.5	Closed-form Expression of Gradient	16
2.4	Variational, Approximate Approach for Computational Speedup	18
2.4.1	Rayleigh Quotient Characterization of Eigenvalues	19
2.5	Weighted- L_1 Formulation and Sparsity	21
2.6	Error Generalization Bounds	24
2.6.1	Rademacher Complexity and Error Bounds	25
2.6.2	Rademacher Complexity Bounds for Neural Networks	27
2.7	Supervised Classification: Experiments and Results	35
2.7.1	MNIST	36
2.7.2	CIFAR10	37
2.7.3	TCGA Cancer Classification	38
2.7.4	Analysis of Results	38
2.8	Discussion	40
3	Fast-Forwarded Exact Sampling of Spanning Trees	43
3.1	Introduction	43
3.2	Method	47
3.2.1	Background on Aldous–Broder Algorithm	47
3.2.2	Bottlenecks in Random Walk Cover Algorithm	49
3.2.3	Skipping Bottlenecks via Fast-forwarding	52
3.2.4	Generalizing the Applicability of Cover Algorithms	54
3.3	Simulations	56

3.4	Bayesian Dendrogram Inference for Crime and Communities Data . . .	58
3.5	Discussion	63
3.5.1	Comparisons of number of iterations	66
3.5.2	Spanning tree-augmented dendrogram Gibbs sampler	66
3.5.3	Posterior similarity matrices and map	67
3.5.4	Details for reversible-jump Markov chain Monte Carlo	68
3.5.5	Modifications for Forest Models	69
3.5.6	Normalized Laplacian	70
4	Graph Comparison via Laplacian Embeddings	71
4.1	Introduction	71
4.1.1	Related Work	74
4.1.2	Organization	76
4.2	Preliminaries	77
4.2.1	Notation and Setup	77
4.2.2	Laplacian Embeddings	78
4.2.3	Wasserstein Distances	80
4.3	The Embedded Laplacian Discrepancy (ELD)	81
4.3.1	ELD for Graphs with Simple Spectrums	81
4.3.2	Design Choices	83
4.3.3	Computation of the ELD	84
4.3.4	Perturbed ELD for Graphs with Repeated Eigenvalues	84

4.3.5	Computation of the Perturbed ELD	86
4.3.6	Selection of hyperparameter k	86
4.4	Theoretical Properties	88
4.4.1	Theoretical Properties of ELD	88
4.4.2	Theoretical Properties of Perturbed ELD	89
4.5	Experiments and Results	92
4.6	Discussion and Future Directions	96
4.7	Appendix	98
4.7.1	Proof of pseudometric property of ELD	98
4.7.2	Proof of invariance under graph isomorphism	99
4.7.3	Proof of invariance under sign configurations of eigenvectors	100
4.7.4	Proof of perturbation almost surely splitting eigenvalues	101
5	On the Statistical Capacity of Deep Generative Models	104
5.1	Introduction	104
5.2	Introduction	105
5.3	Deep Generative Models	106
5.3.1	Deep neural networks	106
5.4	Isoperimetry and Concentration of Deep Generative Models	108
5.4.1	Manifold Setting	111
5.5	Quantification of capacity	111
5.6	Discussion	112

6 Conclusion	114
Bibliography	116
Biography	127

List of Tables

2.1	Classification accuracies for MNIST	37
2.2	Classification accuracies for CIFAR10	38
2.3	Classification accuracies for TCGA	39
4.1	Training and Testing Accuracies on MUTAG	96

List of Figures

2.1	Graphical illustration of graph cut (S, S^C) on neural networks.	10
3.1	Commonly encountered situations with graph bottlenecks	50
3.2	Runtime comparison of random walk and fast-forwarded versions . . .	57
3.3	Gibbs sampler mixing results for Bayesian dendrogram model	62
3.4	Posterior similarity matrices plot	63
3.5	Interpretation of clustering results	63
3.6	Iteration comparison of random walk and fast-forwarded versions . .	66
3.7	Gibbs sampler analysis for depth parameter	69
3.8	RJMCMC analysis for depth parameter	70
4.1	Depiction of Laplacian Embeddings	83
4.2	Comparison between ELD, NPD and NetLSD on rings of cliques . . .	94
4.3	Comparison between ELD, NPD and NetLSD on SBM graphs	102
4.4	ELD running time analysis and connectomes heatmap	103

Acknowledgements

First, I would like to thank my PhD advisor David Dunson. David offers me complete freedom to pursue my own research and learning, and yet is always timely and direct in providing helpful feedback. I echo James Johndrow here in that “the importance of this cannot be understated, nor can its rarity.” David has been exceptionally supportive of my career and is always generous with his time. His guidance and patience has been instrumental to my development. I cannot thank him enough.

In many ways, I am a rebellious student. I like to take a long-term view when it comes to my career development (procrastination?). I am, to put it mildly, an idealist when it comes to science and research. As such, I am extremely grateful to be in a department as accommodating, collegial and supportive as Duke Statistical Science. I cannot imagine doing my PhD anywhere else.

I would also like to thank Leo Duan at the University of Florida. Leo is basically like a second advisor to me. We share similar interests in graphs, MCMC and beyond. His papers and ideas are wildly creative, and I have benefited immensely from his sharp insights and sheer mathematical prowess. Leo has also been very supportive of my career. I am grateful to have Leo as a mentor.

Throughout my undergraduate and graduate studies, I have been most fortunate to have a broad group of the most distinguished scientific mentors. This is the perfect place for me to thank all these people who have helped me grow along the way.

At Duke, I thank Amy Herring, Peter Hoff, Jason Xu and Anru Zhang for agreeing

to serve on my dissertation committee. I also thank Peter Hoff (again!), Simon Mak, Rong Ge and Nick Cook for serving on my preliminary exam committee. I thank Surya Tokdar for being my first year advisor (and for recruiting me to Duke in the first place). I also want to thank Cynthia Rudin for giving me the opportunity to TA for her graduate machine learning course, which has been core part of my experience here at Duke.

At Hopkins, I thank Alex Kolodkin and Randal Hand at the School of Medicine for introducing me to developmental neuroscience, molecular biology, and bench research in general. Alex and Randal are role models who have shown me the way serious scientists operate, and I am forever grateful for their mentorship. I thank Michael I. Miller, J. Tilak Ratnanather and Daniel Tward at BME for introducing me to computational anatomy and quantitative research. It is by working with Dr. Miller that I first got exposed to pattern theory and became interested in advanced mathematics/machine learning. I also thank Joshua Vogelstein, Marc Donohue and Samer Hattar for giving me various opportunities to develop as a scientist and for supporting my career. I thank Don Geman for introducing me to computational biology and statistical learning. His teachings, philosophy and views on science, outlined in his article “Science in the age of selfies”, has influenced me tremendously. Jay Baraban, King-Wai Yau, Rai Winslow, Les Tung, Seth Blackshaw, Eric Young and Laurent Younes provided helpful advice on research, graduate/medical school and career development along the way, for which I am grateful. In an alternate life, I would have pursued a career in medicine/neuroscience. It is an immense privilege to work with all of my Hopkins mentors. I am sure what I have learned from them will benefit me in any field of scientific inquiry.

I thank John Carlson at Yale MCDB for introducing me to *Drosophila* and sensory neuroscience research. John has taught me many things beyond neuroscience. He has always emphasized the importance of doing something good for the world, a

principle that has helped me navigate many inflection points in my career. I must also thank Harry Zhou, Joseph Chang, David Pollard and Forrest Crawford at Yale Statistics/Biostatistics for their inspired teaching that spurred me to pursue a career in statistics (Harry in particular for generously allowing me to audit his high dimensional statistics/minimax course remotely on zoom during COVID, from which I have benefited enormously). I thank Veronika Ročková at Booth for her generous and patient mentorship during my time at Chicago. I also thank David Cheung and Joanne Cavanaugh Simpson for teaching me how to read and write at a higher level, although I fall far short of their standards.

It is an incredible honor and privilege to work in a field as supportive as mathematics/statistics/computer science. Throughout my PhD, various people have shared with me unpublished drafts/code/book manuscripts, offered research opportunities, gave career advice, provided constructive feedback on papers, as well as answered my (silly) questions about all things math/research related. These acts of kindness have saved me much time and headache. I thank, in no particular order, Dan Spielman, Lap Chi Lau, Aaditya Ramdas, Yin Tat Lee, Ronen Eldan, Rick Durrett, Michel Talagrand, Terry Tao, Fang Han, Matt Stephens, Stephen Smale, Iain Carmichael, Ian Goodfellow and Julyan Arbel.

Part of me being an idealist academic is perhaps reflected in my obsession towards teaching/mentoring. I thank Alex Belloni for the opportunity to TA at the business school, and Sumi Ariely for the opportunity to mentor undergraduate students on their honors theses. I thank Tavis Abrahamson, Elijah Meyers, Robert Wolpert, Eric Laber and Cynthia Rudin for their mentorship during my TA stints. I thank my undergraduate mentees and friends Albert, Flora, Jenny, Gaurav, Tony, Chris, Danny, Rui, Richard, Arthi, Niisoja, Shagun and Kai. Our informal Thursday lunch group, where we share insights about all things research, graduate school, internships, books and even startups etc, has been a highlight of my time here at Duke. I have

learned as much, if not more, from my mentees as they have from me.

I would like to thank my friends in the Duke statistical science department. In particular, I thank Andy for being an amazing buddy to share ideas with over basketball, dinners and hikes; Michele for being a great ping pong buddy, for introducing me to all things Italian and for our conversations about math which we both obsess about; Shounak for being a great friend and for always sharing the funniest memes with me. I also want to thank all my friends outside of Duke Statistics. In particular, I thank Henry Chan and Larry Wang at Oxford for our many illuminating conversations about research, reinforcement learning, and beyond.

Even though my postdoc hasn't started yet, I have to thank Barbara Engelhardt for already being an incredibly supportive and accommodating mentor, and Didong Li for his generous advice.

I thank my friends, teammates, teachers and coaches at Duke Rowing, Forge Fencing, as well as the Duke German department for giving me many of my most cherished memories at Duke and Durham outside of my formal academic pursuits.

I am incredibly privileged to have had the opportunity to explore industry research during my PhD. I thank Yang Zhao at Google, Eric Hayward at Amazon Science, Lingrui Gan at Facebook, Jay Wacker at Apple AIML, as well as Han Sun and Andrei Curelea at Pinterest Labs. Although my eyes are set on academia, the industry perspectives have been invaluable.

Last, but certainly not least, I thank my family (my parents Doris and Vincent, and my siblings Venus and Edmund) for their support. For me, the psychological ups and downs of the PhD process is actually much tougher than the research. I would not be here without my family's support.

Introduction

Spectral methods for analyzing graphs and networks are ubiquitous in mathematics, statistics and computer science. The concentration of measure phenomenon from probability has also been instrumental in many developments in these areas. These two themes are joined by the notion of isoperimetry. We use tools from spectral graph theory, concentration of measure and isoperimetry to develop novel methods and theory in sampling, regularization and embeddings.

In chapter 3, which is joint work with Leo Duan, we improve the celebrated Aldous–Broder algorithm along several fronts. The Aldous–Broder algorithm generate a random spanning tree on a given graph by running a random walk until the walk covers all nodes. We formally show that such random walk based cover algorithms can be slow, since they are prone to getting stuck when bottlenecks are present in the graph. We devise a marginalization argument that allows for a random spanning tree sampling that breaks free from such graph bottlenecks while maintaining exact sampling guarantees. We call our resulting sampling the fast-forwarded cover algorithm. We also generalize the applicability of such cover algorithms for sampling spanning trees from more general graphs that go beyond the undirected

setting. We propose a very natural Bayesian hierarchical clustering model with a tree component, and we illustrate how the proposed fast-forwarded cover algorithm can be used for effective block posterior sampling under such a model, exhibiting substantially better performance than more traditional reversible jump approaches.

In chapter 5, we investigate the theoretical limits of certain foundational classes of deep generative models, such as variational autoencoders and generative adversarial networks. Using tools from concentration of measure and functional inequalities, we demonstrate that in practice, under typical choice of priors on the latent variables, any finite sized variational autoencoders and generative adversarial networks can only learn certain light tailed distributions, debunking a popular folklore belief that such models can learn arbitrary continuous distributions given enough data. This can be generalized to cases where the latent variables are distributed on a manifold.

In chapter 2, we investigate the idea of using leveraging a model’s inherent graph structures to regularize itself. Our starting point was that of a neural network. We use the Fiedler value of the neural network’s underlying graph to perform regularization during training. A simple variational characterization of the regularizer leads to a weighted L1 formulation, which leads to sparsity induction. We give theoretical guarantees on such an approach. Such approaches could be used beyond the classification setting, for example in autoencoders, where the bottleneck structures present in the neural network makes Fiedler regularization a natural candidate for regularization.

In chapter 4, we look into the topic of graph comparison. In applications it is often the case that we want to determine how similar two graphs are, for example for downstream graph classification. This problem is often computationally challenging, since graphs are discrete objects. We propose the idea to approximately compare graphs using continuous representations instead. In particular, we choose spectral embeddings of graphs as the continuous representation of choice for comparisons. We

name this method the embedded Laplacian distance. We provide theoretical results and empirical evidence that this method of graph comparison is sound and works excellently in practice.

Fiedler Regularization

2.1 Introduction

Neural networks (NNs) are important tools with many applications in various machine learning domains such as computer vision, natural language processing and reinforcement learning. NNs have been very effective in settings where large labeled datasets are available. Empirical and theoretical evidence has pointed to the ever-increasing capacity of recent NN models, both in depth and width, as an important contributor to their modeling flexibility and success. To ameliorate any potential overfitting that comes with these flexible models, a wide range of techniques for regularizing NNs have been developed. These techniques often regularize the network from a global/uniform perspective, e.g. weight decay (Krogh and Hertz, 1992), L_1/L_1 penalization of weights, dropping nodes/weights in a Bernoulli manner with uniform probability across units/layers (Hinton et al., 2012; Srivastava et al., 2014; Wan et al., 2013), or stopping training early. These commonly used approaches ignore the NN's underlying graphical structure, which can provide valuable connectivity information for regularization.

Existing feedforward NN architectures, e.g. multi-layer perceptrons, frequently employ fully connected layers that lead to many redundant paths between nodes of the network. These redundant connections can contribute to over-fitting through the phenomenon of co-adaptation, where weights become dependent on one another, leading to highly correlated behavior amongst different hidden units (Hinton et al., 2012). Empirical work has shown that dropping weights and nodes randomly during training can significantly improve test performance by reducing co-adaptation (Hinton et al., 2012; Srivastava et al., 2014; Wan et al., 2013).

One natural alternative to these approaches is to take the graph structure of the NN into consideration during regularization. In this work, we would like to regularize the NN through reducing co-adaptation and penalizing extraneous connections in a way that respects the NN’s graphical/connectivity structure. We introduce Fiedler regularization, borrowing from advances in spectral graph theory (Godsil and Royle, 2013; Chung, 1997; Spielman, 2019). The Fiedler value of a connected graph, denoted λ_2 , also known as the algebraic connectivity, is the second smallest eigenvalue of the graph’s Laplacian matrix. The magnitude of λ_2 characterizes how well connected a graph is: if the Fiedler value is small, then the graph is close to disconnected. By adding the Fiedler value as a penalty term to the loss function during training, we can penalize the connectedness of the NN and reduce co-adaptation while taking into account the graph’s connectivity structure.

We also exploit several useful characteristics of the Fiedler value. We show that the Fiedler value is a concave function on the sizes of the NN’s weights. This allows us to draw connections to the literature on folded-concave penalties in variable selection, which are widely adopted in statistics to reduce bias in the penalized regression setting. We additionally show that the Fiedler value’s gradient with respect to the network’s weights admits a closed form expression, which gives insight into the behavior of Fiedler regularization.

In practice, for larger networks, to speed up computation, we propose a variational approach, which replaces the original Fiedler value penalty term by a quadratic form of the graph Laplacian. When used together with the so called Laplacian test vectors, such a Laplacian quadratic form provides a sharp upper bound of the Fiedler value. This variational approximation allows for substantial speedups during training. We give an alternative but equivalent formulation of the variational penalty in terms of a structurally weighted L_1 penalization, where the weights depend on the (approximate) second eigenvector of the graph Laplacian. This L_1 formulation allows us to link Fiedler regularization to sparsity induction, similar to the parallel literature in statistics (Tibshirani, 1996; Zou, 2006). To provide theoretical guarantees of such an approach, we characterize how Fiedler regularization reduces the Rademacher complexity of neural networks. This leads to uniform risk upper bounds for our approach, which sheds light on generalization performance.

There has been prior work on using the Laplacian structure of the input data to regularize NNs (Kipf and Welling, 2016; Jiang and Lin, 2018; Zeng et al., 2019). There has also been recent work on understanding regularization on NNs in Bayesian settings (Vladimirova et al., 2018; Polson and Ročková, 2018). These approaches do not consider the graphical connectivity structure of the underlying NN. One of the main conceptual contribution of this work is to use the graphical/connectivity structure of a model to regularize itself. After the publication of the conference version of this article, several related directions of research have been pursued. (Carmichael, 2021) adopted a similar folded concave Laplacian spectral (FCLS) penalty in the block sparsity estimation setting in statistics. (Arbel et al., 2021) adopted a related Fiedler prior in the Bayesian block-diagonal graphical models setting. In related Laplacian regularization problems considered in (Tuck et al., 2019) and (Carmichael, 2021), majorization-minimization procedures are proposed to optimize the Laplacian regularization term. However, in the context of neural networks, majorization-

minimization is not a computationally feasible method for optimization, in part due to the deep and high-dimensional nature of the model as well as the highly non-convex nature of the loss function.

The main contributions of this work include: (1) to the authors’ best knowledge, this is the first application of spectral graph theory and Fiedler values in regularization of NNs via their own underlying graphical/connectivity structures. (2) We give practical and fast approaches for Fiedler regularization, along with strong theoretical guarantees and experimental performances. This is a journal extension of the conference paper (Tam and Dunson, 2020). The journal version extends the conference paper by providing new generalization error guarantees via a Rademacher complexity analysis as well as a new connection to the variable selection literature via the folded-concave penalty interpretation.

2.2 Spectral Graph Theory

2.2.1 Setup and Background

Let W denote the set of weights of a feedforward NN \mathbf{f} . We denote \mathbf{f} ’s underlying graph structure as G . We would like to use structural information from G to regularize \mathbf{f} during training. Feedforward NNs do not allow for self-loops or recurrent connections, hence it suffices that G be a finite, connected, simple, weighted and undirected graph in this setting. Such a graph G can be fully specified by a triplet $(V, E, |W|)$. Here the vertex set V of G corresponds to all the units (including units in the input and output layers) in the NN \mathbf{f} , while the edge set E of G corresponds to all the edges in \mathbf{f} . For our purposes of regularization, G is restricted to have non-negative weights $|W|$, which are taken to be the absolute value of the corresponding weights W in the NN \mathbf{f} . Throughout the paper we use n to denote the number of vertices in G . $|W|$ and E can be jointly represented by an $n \times n$ weighted adjacency matrix $|\mathbf{W}|$, where $|\mathbf{W}|_{ij}$ is the weight on the edge (i, j) if vertices i and j are

connected, and 0 otherwise. The degree matrix \mathbf{D} of the graph is a $n \times n$ diagonal matrix, where $\mathbf{D}_{ii} = \sum_{j=1}^n |\mathbf{W}|_{ij}$. The Laplacian matrix \mathbf{L} , which is a central object of study in spectral graph theory, is defined as the difference between the degree and the adjacency matrix, i.e. $\mathbf{L} = \mathbf{D} - |\mathbf{W}|$. In certain contexts where we would like to emphasize the dependency of \mathbf{L} on the particular graph G or the weights $|\mathbf{W}|$, we adopt the notation \mathbf{L}_G or $\mathbf{L}_{|\mathbf{W}|}$. We use $[M]$ to denote the set $\{1, 2, \dots, M\}$ for positive integer M . Throughout the paper, eigenvalues are real-valued since the matrices under consideration are symmetric. We adopt the convention where all eigenvectors are taken to be unit vectors. We order the eigenvalues in ascending order, so $\lambda_i \leq \lambda_j$ for $i < j$. When we want to emphasize λ_i as a function of the weights, we use $\lambda_i(|\mathbf{W}|)$. We use \mathbf{v}_i to denote the corresponding eigenvector for λ_i . We use $n(S)$ to denote the cardinality of a given set S .

2.2.2 Graph Laplacian

The Laplacian matrix encodes much information about the structure of a graph. Laplacian eigenvalues and eigenvectors are widely used in tasks such as spectral clustering Von Luxburg (2007); Ng et al. (2001) and manifold learning Belkin and Niyogi (2003), precisely because they capture valuable graph connectivity information. One particularly useful characterization of the graph Laplacian that we use heavily is the so called Laplacian quadratic form (Batson et al., 2012; Spielman, 2019; Chung, 1997), defined below.

Definition 2.2.1 (Laplacian quadratic form) *Given a graph $G = (V, E, |W|)$ with Laplacian matrix \mathbf{L}_G , define its Laplacian quadratic form $Q_G : \mathbb{R}^{|V|} \rightarrow \mathbb{R}^+$ as*

$$Q_G(\mathbf{z}) := \mathbf{z}^T \mathbf{L}_G \mathbf{z} = \sum_{(i,j) \in E} |\mathbf{W}|_{ij} (\mathbf{z}(i) - \mathbf{z}(j))^2, \quad (2.1)$$

where $\mathbf{z}(k)$ denotes the k^{th} entry of the vector $\mathbf{z} \in \mathbb{R}^{|V|}$.

The Laplacian quadratic form demonstrates how a graph’s boundary information can be recovered from its Laplacian matrix. One can think of \mathbf{z} as a mapping that assigns to each vertex a value. If we denote the characteristic vector of a subset of vertices $S \subset V$ as $\mathbf{1}_S$, i.e. $\mathbf{1}_S(i) = 1$ if $i \in S$ and $\mathbf{1}_S(i) = 0$ otherwise, and apply it to the Laplacian quadratic form, we obtain $\mathbf{1}_S^T \mathbf{L}_G \mathbf{1}_S = \sum_{(i,j) \in E, i \in S, j \notin S} |\mathbf{W}|_{ij}$. This expression characterizes the size of the graph cut $(S, V - S)$, which is the sum of the weights of edges crossing the boundary between S and $V - S$. The size of any graph cut can therefore be obtained by application of the corresponding characteristic vector on the Laplacian quadratic form.

2.2.3 Edge Expansion and Cheeger’s Inequality

The above discussion on the sizes of graph cuts is highly related to our study of regularizing a NN. Reducing the sizes of graph cuts in a NN would imply reducing the NN’s connectivity and potential co-adaptation. A related but more convenient construct that captures this notion of boundary sizes in a graph is the graph’s edge expansion (also known as the Cheeger constant or the isoperimetric constant), which can be informally thought of as the smallest “surface-area-to-volume ratio” achieved by a subset of vertices not exceeding half of the graph.

Definition 2.3.1 (Edge expansion of a graph) *The edge expansion ϕ_G of a graph $G = (V, E, |W|)$ is defined as*

$$\phi_G = \min_{S \subset V, n(S) \leq \frac{n(V)}{2}} \frac{\sum_{i \in S, j \notin S} |\mathbf{W}|_{ij}}{n(S)}, \quad (2.2)$$

where $n(S)$ denotes the number of vertices in S .

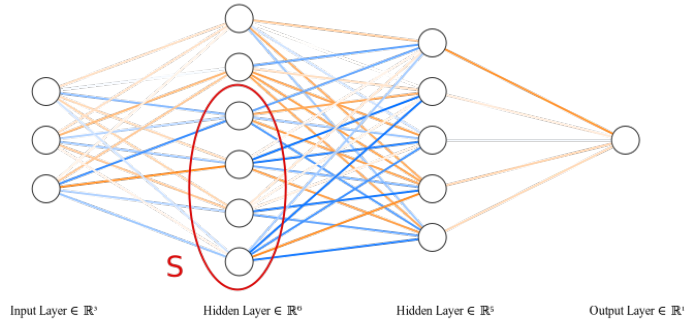


FIGURE 2.1: Graphical illustration of graph cut (S, S^C) on neural networks. A subset of vertices S are circled in red, and the size of the graph cut (S, S^C) is the sum of the sizes of the weights of all the edges that intersect the red boundary. Weights are colored by their signs (orange for positive, blue for negative) and their opacity determined by the absolute values.

Observe that the term in the numerator characterizes the size of the graph cut $(S, V - S)$, while the denominator normalizes the expression by the number of vertices in S . The edge expansion is then taken to be the smallest such ratio achieved by a set of vertices S that has cardinality at most half that of V . One can think of the edge expansion of a graph as characterizing the connectivity bottleneck of a graph. It is highly related to how sparse the graph is and whether there exist nice planar embeddings of the graph (Hall, 1970).

We would like to control the edge expansion of the NN’s underlying graph for regularization. It is well known that direct computation of the edge expansion is a hard problem Garey et al. (1974) due to the combinatorial structure. Instead, we control the edge expansion indirectly through the Fiedler value λ_2 , the second smallest eigenvalue of the graph’s Laplacian \mathbf{L}_G . λ_2 is related to the edge expansion of the graph through Cheeger’s inequality (Spielman, 2019; Chung, 1997; Godsil and Royle, 2013).

Proposition 2.3.2 (Cheeger’s inequality) *Given a graph $G = (V, E, |W|)$, the*

edge expansion ϕ_G is upper and lower bounded as follows:

$$\sqrt{2d_{\max}(G)\lambda_2} \geq \phi_G \geq \frac{\lambda_2}{2}, \quad (2.3)$$

where $d_{\max}(G)$ is the maximum (weighted) degree of vertices in G and λ_2 , the Fiedler value, is the second smallest eigenvalue of G 's Laplacian matrix.

Cheeger's inequality originally arose from the study of Riemannian manifolds and was later extended to graphs. There are many versions of Cheeger's inequality depending on the types of graphs and normalizations used. The proofs can be found in many references, including Spielman (2019); Chung (1997); Godsil and Royle (2013). These types of Cheeger's inequalities are generally tight asymptotically up to constant factors.

The Fiedler value is also known as the algebraic connectivity because it encodes considerable information about the connectedness of a graph. Cheeger's inequality allows sharp control over the edge expansion of G via the Fiedler value. By making the Fiedler value small, we can force the edge expansion of the graph to be small, thus reducing the connectedness of the graph and potentially alleviating co-adaptation in the NN setting. On the other hand, a large Fiedler value necessarily implies a large edge expansion. This implies that penalization of the Fiedler value during NN training is a promising regularization strategy to reduce connectivity and thus co-adaptation.

2.3 Fiedler Regularization

In this section we introduce Fiedler Regularization of neural networks. We characterize this in a supervised classification setting to illustrate the main ideas, even though the general framework is easily extendable to other settings such as regression and autoencoding. The relevant setup and background is introduced below.

2.3.1 Supervised Learning: Classification Setup

We are given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ assumed to be drawn independently from some joint distribution \mathcal{P} . Here i indexes the N data points, $\mathbf{x} \in \mathbb{R}^d$ denotes the d -dimensional feature vectors, and y denotes the outcomes, which for simplicity of presentation we assume to take values in a binary space $\{0, 1\}$ (i.e. binary classification). Many of our results can be easily generalized to settings such as multi-class classification and regression with straightforward modifications. We aim to find a mapping \mathbf{f} that predicts y through $\mathbf{f}(\mathbf{x})$ so that the expectation of some pre-specified loss $\mathbb{E}_{\mathcal{P}}\mathcal{L}(f(\mathbf{x}), y)$ is minimized. Typical choices of the loss function $\mathcal{L}(\cdot, \cdot)$ for classification include the cross-entropy loss, hinge loss etc, whereas for regression problems the mean squared error is common. The empirical approximation $\mathbb{E}_{\mathcal{P}_N}\mathcal{L}(f(\mathbf{x}), y) := \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f(\mathbf{x}_i), y_i)$ is used as a proxy for the expected loss during training, where \mathcal{P}_N denotes the empirical measure. It is assumed that new observations in the testing set are sampled independently from the same distribution \mathcal{P} as the training data. We denote the estimator of \mathbf{f} as $\hat{\mathbf{f}}$. When we want to emphasize the estimator's dependence on the weights, we use $\hat{\mathbf{f}}_{\mathbf{w}}$.

2.3.2 Neural Network Setup

For Fiedler Regularization, we consider for now only feedforward NNs (see discussion for potential extensions). For feedforward NNs, $\hat{\mathbf{f}}$ is characterized as a composition of non-linear functions $\{\mathbf{g}^{(l)}\}_{l=1}^{\Lambda}$, i.e. $\hat{\mathbf{f}} = \mathbf{g}^{(\Lambda)} \circ \mathbf{g}^{(\Lambda-1)} \circ \dots \circ \mathbf{g}^{(1)}(\mathbf{x})$, where Λ is the number of layers in the network. The outputs of the l^{th} layer have the form $\mathbf{h}^{(l)} := \mathbf{g}^{(l)}(\mathbf{h}^{(l-1)}) := \sigma^{(l)}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$, where $\sigma^{(l)}$, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the activation function, weight matrix and bias of the l^{th} layer of the NN, respectively. Note that we use \mathbf{W} to denote the weighted adjacency matrix of the entire neural network encoded as a graph, and $\mathbf{W}^{(l)}$ to denote the matrix encoding the linear transformation applied at the l^{th} layer. In essence, each hidden layer first performs

an affine transformation on the previous layer’s outputs, followed by an element-wise activation that is generally nonlinear. For additional details, see the excellent review by Fan et al. (2019).

2.3.3 Penalizing with Fiedler Value

Given the motivations from section 2, we would like to penalize the graph connectivity of the NN during training. In the Fiedler regularization approach, we add λ_2 as a penalty term to the objective.

Definition 3.2.1 (Fiedler Regularization) *During training of the neural network, we optimize the following objective:*

$$\min_{\mathbf{W}} \mathbb{E}_{\mathcal{P}_N} \mathcal{L}_{\mathbf{W}} + \delta \lambda_2(|\mathbf{W}|), \quad (2.4)$$

where $\lambda_2(|\mathbf{W}|)$ is the Fiedler value of the NN’s underlying graph, \mathbf{W} is the weight matrix of the NN, δ is a tuning parameter, and Y and X denote the training labels and training features, respectively.

We remark that the actual NN $\hat{\mathbf{f}}_{\mathbf{W}}$ has weights that can be negative, but the regularization term $\lambda_2(|\mathbf{W}|)$ only depends on the sizes of such weights, which can be thought of as edge capacities of the underlying graph G .

Note that one can generalize the penalty by applying a differentiable function Q to λ_2 . For our discussion below, we focus on the case where no Q is applied in order to focus the analysis on the Fiedler value. However, incorporating Q is straightforward, and desirable properties such as having a closed-form gradient can generally be retained. We also note that, without loss of generality, one can consider the biases of the units in the NN as additional weights with constant inputs, so it is straightforward to include consideration of both biases and weights in Fiedler

regularization.

For our purposes, we consider the main parameters of interest to be the weights of the NN. The choice of the activation function(s), architecture, and hyperparameters such as the learning rate or the tuning factor are all considered to be pre-specified in our study. There is a separate and rich literature devoted to methods for selecting activation functions/architectures/hyperparameters that we do not consider here.

It is worth noting that the Fiedler Regularization defined in this section is meant to illustrate the core ideas and motivations in a theoretical and idealized way. In practice, it is computationally intensive to penalize the Fiedler value directly, and for implementation, we refer to the approximate variational approach described in the subsequent section.

2.3.4 Properties of the Fiedler Value

It is instructive to examine some properties of the Fiedler value in order to understand why it is an appropriate tool for regularization.

First, one concern is whether using the Fiedler value as a penalty in the objective would complicate the optimization process. The Fiedler value can be viewed as a root of the Laplacian matrix’s characteristic polynomial, which in higher dimensions has no closed-form solution and can depend on the network’s weights in a convoluted manner.

To address this concern, the following proposition shows that the Fiedler penalty is a concave function of the sizes of the NN’s weights. This shows that when we add the Fiedler penalty to deep learning objectives, which are typically highly non-convex, we are not adding substantially to the optimization problem’s difficulty.

Proposition 3.3.1 (Concavity of Fiedler Value) *The function $\lambda_2(|W|)$ is a concave function of the sizes of the NN’s weights $|W|$.*

Proof: Since the Fiedler value λ_2 is just the second smallest eigenvalue of the Laplacian, and we know that the first eigenvector of the Laplacian must be constant, we can consider λ_2 's Rayleigh-Ritz variational characterization as follows:

$$\begin{aligned} \lambda_2(|\mathbf{W}|) &= \inf_{\|\mathbf{u}\|=1, \mathbf{u}^T \mathbf{1}=0} \mathbf{u}^T \mathbf{L} \mathbf{u} \\ &= \inf_{\|\mathbf{u}\|=1, \mathbf{u}^T \mathbf{1}=0} \sum_{(i,j) \in E} |\mathbf{W}|_{ij} (\mathbf{u}(i) - \mathbf{u}(j))^2 \end{aligned}$$

Note that this is a pointwise infimum of a linear function of $|\mathbf{W}|_{ij}$. Since linear functions are concave (and convex), and the pointwise infimum preserves concavity, we have that λ_2 is a concave function of the sizes of the weights.

This is related to Laplacian eigenvalue optimization problems and we refer to Boyd (2006) and Sun et al. (2006) for a more general treatment.

An immediate corollary of Proposition 3.3.1 is that the Fiedler value is a folded-concave penalty with respect to the weights W of the neural network. Typical sparsity-inducing methods with L_1 penalties, such as the least absolute shrinkage and selection operator (LASSO) in statistics (Tibshirani, 1996), can suffer from large biases. Methods with folded-concave penalties, such as the Smoothly Clipped Absolute Deviation (SCAD) (Fan and Li, 2001) penalty and the minimax concave penalty (MCP) (Zhang, 2010), have been shown to reduce the biases of L_1 -based sparsity-inducing methods. This shape of folded-concave penalty functions leads to the property that smaller weights are going to be penalized relatively more than the larger weights. In the case of Fiedler regularization, the weights of neural network that have “low connectivity” are penalized more. We can further understand this phenomenon via analyzing the gradient of the Fiedler value with respect to the neural network's weights.

2.3.5 Closed-form Expression of Gradient

In all except the most simple of cases, optimizing the loss function $\min_{\mathbf{W}} \mathcal{L}(Y, \mathbf{f}_{\mathbf{W}}(X))$ is a non-convex problem. There are a variety of scalable, stochastic algorithms for practical optimization on such objectives. Virtually all of the widely used methods, such as stochastic gradient descent (SGD) (Ruder, 2016), Adam (Kingma and Ba, 2014), Adagrad (Duchi et al., 2011), RMSProp (Graves, 2013) etc, require computation of the gradient of the objective with respect to the parameters.

We provide a closed-form analytical expression of the gradients of a general Laplacian eigenvalue with respect to the entries of the Laplacian matrix. From that, as a straightforward corollary, a closed-form analytical expression of the Fiedler value's gradient is obtained.

Proposition 3.4.1 (Gradient of Laplacian Eigenvalue) *Assuming that the eigenvalues of the Laplacian \mathbf{L} are not repeated, the gradient of the k^{th} smallest eigenvalue λ_k with respect to \mathbf{L} 's $(ij)^{\text{th}}$ entry \mathbf{L}_{ij} can be analytically expressed as*

$$\frac{d\lambda_k}{d\mathbf{L}_{ij}} = \mathbf{v}_k(i) \times \mathbf{v}_k(j),$$

where $\mathbf{v}_k(i)$ denotes the i^{th} entry of the k^{th} eigenvector of the Laplacian. We adopt the convention in which all eigenvectors under consideration are unit vectors.

Proof: To compute $\frac{d\lambda_k}{d\mathbf{L}_{ij}}$, note that since the Laplacian matrix is symmetric, all eigenvalues are real. By assumption, the eigenvalues are not repeated. Under this situation, there is an existing closed-form formulae (see (Petersen et al., 2008)): $\mathbf{v}_k^T(\partial\mathbf{L})\mathbf{v}_k = d\lambda_k$. Specializing to individual entries, we get $\frac{d\lambda_k}{d\mathbf{L}_{ij}} = \mathbf{v}_k(i) \times \mathbf{v}_k(j)$.

From the above, we can easily obtain an analytical expression for the gradient of

the Fiedler value with respect to the weights of the neural network.

Corollary 3.4.2 (Gradient of Fiedler value with respect to weights) *Under the same assumptions of Proposition 3.4.1, as an immediate special case, the gradient of the Fiedler value λ_2 can be expressed as:*

$$\frac{d\lambda_2}{d\mathbf{L}_{ij}} = \mathbf{v}_2(i) \times \mathbf{v}_2(j)$$

Since $\mathbf{L}_{ij} = -|\mathbf{W}|_{ij}$ for $i \neq j$, this yields the following:

$$\frac{d\lambda_2}{d|\mathbf{W}|_{ij}} = -\mathbf{v}_2(i) \times \mathbf{v}_2(j)$$

Under the additional assumption that the weights under consideration are non-zero, one can remove the absolute value mapping in the gradient by a simple application of the chain rule.

Our assumption that the eigenvalues are not repeated generally holds in the context of NNs. The weights in a NN are usually initialized as independent draws from certain continuous distributions, such as the uniform or the Gaussian. Repeated Laplacian eigenvalues often occur when there are strong symmetries in the graph. Such symmetries are typically broken in the context of NNs since the probability of different weights taking the same non-zero value at initialization or during training is negligible.

The reason we develop the above analytical form for the gradient of the Fiedler value is not for direct optimization of real life neural networks: recomputing the second Laplacian eigenvector at every iteration of optimization (e.g. in SGD) would be prohibitively expensive for all but the smallest neural networks. Rather, we use the analytical form of the gradient to shed light on what Fiedler Regularization achieves

theoretically. For actual computation, we propose a tight variational approximation in the next section which works well in practice.

There is a large literature on spectral clustering that analyzes the interpretation of the entries of the second Laplacian eigenvector. In particular, if vertices i and j are poorly connected to each other (i.e. they belong to different “clusters”), then $\mathbf{v}_2(i) \times \mathbf{v}_2(j)$ would be a very small/negative real number. This indicates that the derivative $\frac{d\lambda_2}{d|\mathbf{W}|_{ij}} = -\mathbf{v}_2(i) \times \mathbf{v}_2(j)$ is larger for vertices i and j that are less well connected, which in turn implies that edges/weights between poorly connected vertices gets penalized most under Fiedler regularization.

2.4 Variational, Approximate Approach for Computational Speedup

We have given theoretical motivation and outlined the use of the Fiedler value as a tool for NN regularization. However, typical matrix computations for eigenvalues and eigenvectors are of order $O(n^3)$, where $n = |V|$. This can be computationally prohibitive for moderate to large networks. Even though there exist theoretically much more efficient algorithms to approximate eigenvalues/eigenvectors of graph-related matrices, in practice computing the Fiedler value in every iteration of training can prove costly. To circumvent this issue, we propose an approximate, variational approach to speed up the computation to $O(|E|)$, where $|E|$ is the number of edges in the graph. This proposed approach can be readily implemented in popular deep learning packages such as Tensorflow and PyTorch.

We make the following observations. First, for the purpose of regularizing a NN, we do not need the exact Fiedler value of the Laplacian matrix. A good approximation suffices. Second, there is no need to update our approximate λ_2 at every iteration during training. We can set a schedule to update our λ_2 approximation periodically, say once every 100 iterations. Both of these observations allow for sub-

stantial speedups in practice. An outline of the pseudo-code for this approximate, variational approach is provided in Algorithm 1.

To obtain an approximate Fiedler value, we use a special type of Laplacian quadratic form involving the so called test vectors. We additionally provide a perturbation bound that gives justification for periodically updating the Fiedler value.

2.4.1 Rayleigh Quotient Characterization of Eigenvalues

We can closely upper-bound the Fiedler value via the notion of test vectors (Spielman, 2019), which depends crucially upon the Rayleigh quotient characterization of eigenvalues.

Proposition 4.1.1 (Test Vector Bound) *For any unit vector \mathbf{u} that is perpendicular to the constant vector $\mathbf{1}$, we have:*

$$\lambda_2 \leq \mathbf{u}^T \mathbf{L} \mathbf{u}$$

Any such unit vector is called a test vector. Equality is achieved when $\mathbf{u} = \mathbf{v}_2$.

Proof: The Laplacian matrix of a non-negatively weighted graph is symmetric and positive semidefinite. Thus all eigenvalues are real and non-negative. In particular, the smallest eigenvalue of the Laplacian is 0, with the constant vector being the first eigenvector. For a connected graph, the second smallest eigenvalue, which is the Fiedler value, can thus be variationally characterized via the Courant-Fischer theorem by $\lambda_2 = \min_{\|\mathbf{u}\|=1, \mathbf{u}^T \mathbf{1}=0} \mathbf{u}^T \mathbf{L} \mathbf{u}$. This gives us the desired upper bound.

The above description implies that by appropriately choosing test vectors, we can effectively upper bound the Fiedler value. This in turn implies that during training, instead of penalizing by the exact Fiedler value, we can penalize by the quadratic

form upper bound instead. In other words, we would perform the following optimization,

$$\min_{\mathbf{w}} \mathcal{L}(Y, \hat{\mathbf{f}}_{\mathbf{w}}(X)) + \delta \mathbf{u}^T \mathbf{L} \mathbf{u}$$

For a given \mathbf{u} , this speeds up computation of the penalty term considerably to $O(|E|)$ since $\mathbf{u}^T \mathbf{L} \mathbf{u} = \sum_{(i,j) \in E} |\mathbf{W}|_{ij} (\mathbf{u}(i) - \mathbf{u}(j))^2$. The core question now becomes how to choose appropriate test vectors \mathbf{u} that are close to \mathbf{v}_2 .

We propose to initialize training with the exact \mathbf{v}_2 and recompute/update \mathbf{v}_2 only periodically during training. For an iteration in between two exact \mathbf{v}_2 updates, the \mathbf{v}_2 from the previous update carries over and serves as the test vector for the current iteration. By proposition 4.1.1, this always upper-bounds the true λ_2 .

During training, weights of the NN are updated at each iteration. This is equivalent to adding a symmetric matrix \mathbf{H} to the Laplacian matrix \mathbf{L} , which is also symmetric, at every iteration. We can bound the effect of such a perturbation on the eigenvalues via Weyl's inequality (Horn and Johnson, 2012; Horn et al., 1998).

Proposition 4.1.2 (Weyl's Inequality) *Given a symmetric matrix \mathbf{L} and a symmetric perturbation matrix \mathbf{H} , both with dimension $n \times n$, for any $1 \leq i \leq n$, we have:*

$$|\lambda_i(\mathbf{L} + \mathbf{H}) - \lambda_i(\mathbf{L})| \leq \|\mathbf{H}\|_{op}$$

where $\|\cdot\|_{op}$ denotes the operator norm.

Weyl's inequality is a classic result in matrix analysis, and its proof can be found in the excellent reference (Horn and Johnson, 2012) as a straightforward result of linearity and the Courant-Fischer theorem. As an immediate special case, $|\lambda_2(\mathbf{L} + \mathbf{H}) - \lambda_2(\mathbf{L})| \leq \|\mathbf{H}\|_{op}$. Proposition 4.1.2 tells us that as long as the perturbation \mathbf{H} is small, the change in Fiedler value caused by the perturbation is also small. In

Input: Training data $\{\mathbf{x}_i, y_i\}_{i=1}^N$
Hyperparameters: Learning rate η , batch size m , penalty parameter δ , updating period T
Algorithm:
Initialize parameters \mathbf{W} of the NN
Compute the Laplacian $\mathbf{L}_{|\mathbf{W}|}$ of the NN
Compute the Fiedler vector \mathbf{v}_2 of the Laplacian $\mathbf{L}_{|\mathbf{W}|}$
Set $\mathbf{u} \leftarrow \mathbf{v}_2$
Initialize counter $c = 0$
while Stopping criterion not met **do**
 Sample minibatch $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^m$ from training set
 Set gradient $\boldsymbol{\gamma} = \mathbf{0}$
 for $i = 1$ to m **do**
 Compute gradient $\boldsymbol{\gamma} \leftarrow \boldsymbol{\gamma} + \nabla_{\mathbf{W}} \mathcal{L}(\hat{\mathbf{f}}_{\mathbf{W}}(\mathbf{x}^{(i)}), y^{(i)}) + \delta \nabla_{\mathbf{W}} \mathbf{u}^T \mathbf{L}_{|\mathbf{W}|} \mathbf{u}$
 end for
 Apply gradient update $\mathbf{W} \leftarrow \mathbf{W} - \eta \boldsymbol{\gamma}$
 Update Laplacian matrix $\mathbf{L}_{|\mathbf{W}|}$
 Update counter $c \leftarrow c + 1$
 if $c \bmod T = 0$ **then**
 Recompute \mathbf{v}_2 from $\mathbf{L}_{|\mathbf{W}|}$
 Set $\mathbf{u} \leftarrow \mathbf{v}_2$
 end if
end while
Algorithm 1: Variational Fiedler Regularization with SGD

fact, this shows that the map $\mathbf{L} \rightarrow \lambda_2(\mathbf{L})$ is Lipschitz continuous on the space of symmetric matrices. In the context of training NNs, \mathbf{H} represents the updates to the weights of the network during training. This justifies updating λ_2 only periodically for the purposes of regularization. It suggests that with a smaller learning rate, \mathbf{H} would be smaller, and therefore the change to λ_2 would also be smaller, and updates of the test vectors can be more spaced apart. On the other hand, for larger learning rates we recommend updating test vectors more frequently.

2.5 Weighted- L_1 Formulation and Sparsity

We now expand on an equivalent formulation of the variational Fiedler penalty as a weighted L_1 penalty. We note that the Laplacian quadratic form in the variational

Fiedler penalty can be written as $\mathbf{u}^T \mathbf{L} \mathbf{u} = \sum_{(i,j) \in E} |\mathbf{W}|_{ij} (\mathbf{u}(i) - \mathbf{u}(j))^2$. This yields the variational objective:

$$\min_{\mathbf{W}} \mathcal{L}(Y, \hat{\mathbf{f}}_{\mathbf{W}}(X)) + \delta \sum_{(i,j) \in E} |\mathbf{W}|_{ij} (\mathbf{u}(i) - \mathbf{u}(j))^2$$

We note that both $|\mathbf{W}|_{ij}$ and $(\mathbf{u}(i) - \mathbf{u}(j))^2$ are non-negative. This is equivalent to performing L_1 penalization on \mathbf{W}_{ij} with weights $(\mathbf{u}(i) - \mathbf{u}(j))^2$.

There is an immense literature on modified L_1 penalties in shallow models (Zou, 2006; Candes et al., 2008). It is well known that optimizing an objective under (weighted) L_1 constraints often yields sparse solutions. This thus connects our Fiedler regularization approach with sparsity induction on the weights of the NN. From an optimization perspective, recall that $\lambda_2(|\mathbf{W}|)$ is a concave function of $|\mathbf{W}|$ and thus a folded concave function of \mathbf{W} . One way to optimize this folded concave function is to substitute it with a majorizing surrogate function, in the spirit of majorization-minimization algorithms. It is easy to see from our test vector bound that the weighted L_1 formulation is indeed a majorizer of $\lambda_2(|\mathbf{W}|)$, lending support to our variational approximation from a separate perspective. Similar ideas are explored in (Carmichael, 2021).

In Fiedler regularization, the weights $|\mathbf{W}|_{ij}$ are scaled by a factor of $(\mathbf{u}(i) - \mathbf{u}(j))^2$, where \mathbf{u} is a test vector that approximates \mathbf{v}_2 . From the spectral clustering literature (Hagen and Kahng, 1992; Donath and Hoffman, 1972), we understand that \mathbf{v}_2 is very useful for approximating the minimum conductance cut of a graph. The usual heuristic is that one sorts entries of \mathbf{v}_2 in ascending order, sets a threshold t , and groups all vertices i having $\mathbf{v}_2 > t$ into one cluster and the rest into another cluster. If the threshold t is chosen optimally, this clustering will be a good approximation to the minimum conductance cut of a graph. As such, the farther apart $\mathbf{v}_2(i)$ and $\mathbf{v}_2(j)$ are (1) the edge between nodes i and j (if it exists) is likely “less important”

for the connectivity structure of the graph, and (2) the more likely that nodes i and j belong to different clusters.

This is also connected to spectral drawing of graphs (Spielman, 2019; Hall, 1970), where it can be shown that “nice” planar embeddings/drawings of the graph do not exist if λ_2 is large. In this sense, Fiedler regularization is forcing the NN to be “more planar” while respecting its connectivity structure.

Hence, with Fiedler regularization, the penalization on $|\mathbf{W}|_{ij}$ is the strongest when this edge is “less important” for the graph’s connectivity structure. This would in theory lead to greater sparsity in edges that have low weights and connect distant vertices belonging to different clusters. Thus, Fiedler regularization sparsifies the NN in a way that respects its connectivity structure.

To this end, an alternative guarantee is the ordering property of Laplacian eigenvalues with respect to a sequence of graphs (Godsil and Royle, 2013).

Proposition 5.1 (Laplacian Eigenvalue Ordering) *Given the graph G with non-negative weights, if we remove an edge (a, b) to obtain $G \setminus (a, b)$, we have that $\lambda_2(G \setminus (a, b)) \leq \lambda_2(G)$.*

Proof: The proof is a simple generalization of Godsil and Royle (2013). Pick \mathbf{q}_2 and \mathbf{r}_2 to be second eigenvectors of G and $G \setminus (a, b)$ respectively. By the Laplacian quadratic form characterization of eigenvalues, we have

$$\begin{aligned} \lambda_2(G) &= \mathbf{q}_2^T \mathbf{L}_G \mathbf{q}_2 = \sum_{(c,d) \in E} |\mathbf{W}|_{cd} (\mathbf{q}_2(c) - \mathbf{q}_2(d))^2 \\ &\geq \left[\sum_{(c,d) \in E} |\mathbf{W}|_{cd} (\mathbf{q}_2(c) - \mathbf{q}_2(d))^2 \right] - |\mathbf{W}|_{ab} (\mathbf{q}_2(a) - \mathbf{q}_2(b))^2 \\ &= \mathbf{q}_2^T \mathbf{L}_{G \setminus (a,b)} \mathbf{q}_2 \geq \mathbf{r}_2^T \mathbf{L}_{G \setminus (a,b)} \mathbf{r}_2 = \lambda_2(G \setminus (a, b)) \end{aligned}$$

where the first inequality follows from the non-negativity of the weights under consideration and the second inequality follows from the Rayleigh-Ritz variational characterization of eigenvalues.

Hence, by sparsifying edges and reducing edge weights, we are in effect reducing the Fiedler value of the NN. The above ordering property is a special case of the more general Laplacian eigenvalue interlacing property, where $\lambda_2(G \setminus (a, b))$ also admits a corresponding lower bound. For a general treatment, see Godsil and Royle (2013).

We remark that since Fiedler regularization encourages sparsity, during the training process the NN might become disconnected, rendering λ_2 to become 0. This issue could be easily avoided in practice by dropping one of the disconnected components (e.g. the smaller one) from the Laplacian matrix during training, i.e. remove the Laplacian matrix's rows and columns that correspond to the vertices in the dropped component.

2.6 Error Generalization Bounds

The guarantees that we have provided above focus on the approximation and variational aspects of Fiedler Regularization. In this section, we provide some theoretical guarantees on the generalization error of such an approach. In particular, we use the notion of Rademacher complexity (Bartlett and Mendelson, 2002) from statistical learning theory to provide finite-sample, uniform generalization error upper bounds for Fiedler regularization. This yields insights into how Fiedler regularization reduces the expressiveness of the learned NN, thus reducing overfitting and improving generalization performance. We focus on the case of supervised classification. We analyze the Laplacian quadratic form/weighted L_1 penalization formulation, which includes the exact Fiedler value penalization as a special case when the test vector

used is exactly \mathbf{v}_2 . To the best of our knowledge, this is the first Rademacher complexity analysis for a weighted- L_1 penalized neural network. The results here might be of independent interest.

2.6.1 Rademacher Complexity and Error Bounds

One key aspect of characterizing the generalization performance of a class of functions/classifiers is via understanding their expressiveness. The notion of empirical Rademacher complexity provides a useful approach for quantifying the expressiveness of a class of functions.

Definition 6.1 (Rademacher Complexity) *Given a sample of N points $\mathbf{z}_1, \dots, \mathbf{z}_N$ drawn i.i.d. from some probability distribution \mathcal{P} on a set $Z \subseteq \mathbb{R}^d$, as well as a function class \mathcal{H} consisting of functions that map from Z to \mathbb{R} , we define the empirical Rademacher complexity of \mathcal{H} , denoted $\hat{R}_N(\mathcal{H})$, as:*

$$\hat{R}_N(\mathcal{H}) := \mathbb{E}_{\mathcal{R}} \left[\sup_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \Omega_i h(\mathbf{z}_i) \middle| \{\mathbf{z}_i\}_{i=1}^N \right]$$

where Ω_i are independent and identically distributed random variables drawn from the Rademacher distribution \mathcal{R} (i.e. the uniform distribution on $\{-1, 1\}$), and the expectation is taken over the Rademacher random variables Ω_i .

$\hat{R}_N(\mathcal{H})$ is a random quantity since it conditions on the sample $\mathbf{z}_1, \dots, \mathbf{z}_N$. The expectation of $\hat{R}_N(\mathcal{H})$ under the product measure \mathcal{P}^N is known as the Rademacher complexity of \mathcal{H} , denoted $R_{N,\mathcal{P}}(\mathcal{H})$. In other words,

$$R_{N,\mathcal{P}}(\mathcal{H}) := E_{\mathcal{P}^N}[\hat{R}_N(\mathcal{H})]$$

On an intuitive level, the Rademacher complexity of the function class \mathcal{H} charac-

terizes how “expressive” \mathcal{H} is, in terms of the ability of \mathcal{H} to match a random sign pattern (i.e. the Rademacher random variables). The larger the Rademacher complexity, the more expressive the function class, and this generally indicates poorer generalization performance due to overfitting. Note that the expectation in the definition of the Rademacher complexity depends on \mathcal{P} , which in many problems is complicated, high-dimensional and/or unknown. The notion of empirical Rademacher complexity bypasses this issue by conditioning on the sample.

The Rademacher complexity is a useful notion in part because it naturally occurs in many bounds of interest in empirical process theory and statistical learning theory. For the case of supervised classification where $Z = (X, Y)$, one can choose $\mathcal{H} := \{(x, y) \mapsto \mathcal{L}(f(x), y) : f \in \mathcal{F}\}$, where \mathcal{L} is a fixed loss function, and \mathcal{F} is the class of classifiers under consideration. In the case of classification, the function \mathcal{L} is commonly chosen to be a surrogate loss, such as the hinge loss or the logistic loss. These commonly used loss functions are Lipschitz continuous with Lipschitz constant 1. The following theorem is then useful.

Theorem 6.2 (Generalization Bounds via Rademacher Complexity) *Let $\mathcal{H} := \{(x, y) \mapsto \mathcal{L}(f(x), y) : f \in \mathcal{F}\}$, where \mathcal{L} is a 1-Lipschitz continuous loss function. Then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ under the product measure \mathcal{P}^N , the following inequality holds:*

$$\sup_{h \in \mathcal{H}} \left[\frac{1}{N} \sum_{i=1}^N h(\mathbf{z}_i) - \mathbb{E}_{\mathcal{P}}(h(\mathbf{z})) \right] \leq 2\hat{R}_N(\mathcal{H}) + O\left(\sqrt{\frac{\log(1/\delta)}{N}}\right),$$

where each \mathbf{z}_i is distributed according to \mathcal{P} independently.

The proof of the above theorem could be found in many sources and textbooks, including Bartlett and Mendelson (2002); Shalev-Shwartz and Ben-David (2014);

Bousquet et al. (2003). The above probabilistic bound is uniform and finite-sample. When applied to the learning problem, the quantity $\mathbb{E}_{\mathcal{P}}(h(\mathbf{z}))$ corresponds to the “true” error whereas the quantity $\frac{1}{N} \sum_{i=1}^N h(\mathbf{z}_i)$ corresponds to the error obtained on the training set. Thus theorem 6.2 provides a way to control the generalization error, in a uniform, finite-sample and distribution-agnostic way, which is particularly suited to analyzing modern machine learning models. The asymptotic notation $O(\cdot)$ is used here to hide constant factors that are very small. Explicit forms of the inequality with the constant factors shown are described in the sources cited above.

2.6.2 Rademacher Complexity Bounds for Neural Networks

Armed with the above theorem, our focus turns to upper-bounding the empirical Rademacher complexity $\hat{R}_N(\mathcal{H})$ for feedforward neural networks penalized by Fiedler regularization. This in turn shows us how Fiedler regularization affects generalization error bounds of neural networks. Our strategy is to first upper bound the Rademacher complexity $\hat{R}_N(\mathcal{F})$ of the class of feedforward neural networks classifiers under consideration. We then use the results on $\hat{R}_N(\mathcal{F})$ to derive upper bounds on $\hat{R}_N(\mathcal{H})$ by applying the loss function.

To gain intuition to our approach, recall that we operate in the following setting: fix the architecture of a neural network, setting σ to be the activation function and Λ to be the number of layers. Then the feedforward neural network under our consideration would have the following form:

$$f(\mathbf{x}) := \mathbf{W}^{(\Lambda)} \sigma(\mathbf{W}^{(\Lambda-1)} (\sigma(\dots \sigma(\mathbf{W}^{(1)} \mathbf{x}) \dots))) \quad (2.5)$$

where the activation function σ is applied component-wise. Note that biases are absorbed into the inputs \mathbf{x} without loss of generality. Also note that $\mathbf{W}^{(\Lambda)}$ denotes that matrix encoding the linear transformation that maps nodes from the $l - 1^{\text{th}}$ layer to nodes in the l^{th} layer.

To prove Rademacher complexity bounds for neural networks, we first list several useful lemmas. Given the compositional nature of equation 2.5, the first lemma below helps control the Rademacher complexity of function classes under Lipschitz compositions.

Lemma 6.3 (Talagrand’s Contraction Lemma) (Ledoux and Talagrand, 2013) *Let ϕ_1, \dots, ϕ_N be γ -Lipschitz functions mapping from $\mathbb{R} \rightarrow \mathbb{R}$ for some $\gamma > 0$.*

Then

$$\begin{aligned} \hat{R}_N((\phi_1, \dots, \phi_N) \circ \mathcal{H}) &:= \mathbb{E}_{\mathcal{R}} \left[\sup_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \Omega_i \phi_i(h(\mathbf{z}_i)) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \\ &\leq \gamma \mathbb{E}_{\mathcal{R}} \left[\sup_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^N \Omega_i h(\mathbf{z}_i) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] = \gamma \hat{R}_N(\mathcal{H}) \end{aligned}$$

where the \circ notation denotes function composition.

In other words, we can use the (empirical) Rademacher complexity of \mathcal{H} to control its (empirical) Rademacher complexity after an element-wise composition with γ -Lipschitz functions. Talagrand’s contraction lemma provides a useful way to analytically control the expressivity of functions/classifiers that exhibit compositional representations.

Feedforward NNs naturally admit compositional descriptions, with the original input data undergoing interleaving affine transformations and activations. Virtually all activation functions that are used in practice are Lipschitz continuous, with popular choices such as tanh, ReLU, etc being 1-Lipschitz. Popular operators, most notably the convolutional operator commonly used in computation settings, are also Lipschitz. Talagrand’s contraction lemma thus allow us to study broad classes of feedforward neural networks.

Linear transformations also form a crucial part of neural networks. The following

lemma gives us control over Rademacher complexity of linear transformations under weights with bounded L_1 norm.

Lemma 6.4 (Rademacher Complexity of a Linear Class) *Given an i.i.d. sample $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ and a function class $\mathcal{F}_{linear}^B := \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle \mid \mathbf{w} \in \mathbb{R}^d, \|\mathbf{w}\|_1 \leq B\}$, suppose that $\|\mathbf{x}_i\|_\infty \leq C$ for all \mathbf{x}_i in the dataset, we have*

$$\hat{R}_N(\mathcal{F}_{linear}^B) \leq BC \sqrt{\frac{2 \log(2d)}{N}}$$

This is a standard result whose proof is based on a lemma of Massart (2000). A complete proof can be found in many references, for example section 26.2 of Shalev-Shwartz and Ben-David (2014).

The assumption of bounded L_1 norm of the weights \mathbf{w} is appropriate for our context of studying neural network sparsity, and the assumption of uniformly bounded L_∞ norm on the data points in the dataset is usually going to be satisfied in practice.

A small modification of the above lemma would lend itself to the scenario of a weighted L_1 scenario that is applicable to the Fiedler regularization setting:

Corollary 6.5 (Rademacher Complexity of a Linear Class under Weighted L_1 Constraints) *Given an i.i.d. sample $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ and a function class $\mathcal{F}_1 := \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle \mid \mathbf{w} \in \mathbb{R}^d, \sum_{i=1}^d \mathbf{c}_i |\mathbf{w}_i| \leq B\}$, where $\mathbf{c} \geq \mathbf{0}$ is a fixed d -dimensional weighting vector, suppose that $\|\mathbf{x}_i\|_\infty \leq C$ for all \mathbf{x}_i in the dataset, we have*

$$\hat{R}_N(\mathcal{F}_1) \leq \frac{B}{\min(\mathbf{c})} C \sqrt{\frac{2 \log(2d)}{N}}$$

where we use $\min(\mathbf{c})$ to denote the minimum entry in the vector \mathbf{c} .

Proof: Note that $\min(\mathbf{c}) \sum_{i=1}^d |\mathbf{w}_i| \leq \sum_{i=1}^d \mathbf{c}_i |\mathbf{w}_i| \leq B$. This implies $\|\mathbf{w}\|_1 \leq \frac{B}{\min(\mathbf{c})}$.

Note that this implies $\mathcal{F}_1 \subseteq \mathcal{F}_{linear}^{\frac{B}{\min(\mathbf{c})}}$. Invoking Lemma 6.4, we obtain the desired

upper bound $\hat{R}_N(\mathcal{F}_1) \leq \hat{R}_N(\mathcal{F}_{linear}^{\frac{B}{\min(\mathbf{c})}}) \leq \frac{B}{\min(\mathbf{c})} C \sqrt{\frac{2 \log(2d)}{N}}$.

Recall that by the Lagrangian dual formulation in optimization, an L_1 penalty added to the objective corresponds to imposing a L_1 constraint on the domain on the objective. Analogously, in the case of Fiedler regularization, since a weighted L_1 penalty is added to the objective, it is equivalent to imposing a weighted L_1 constraint on the weights.

We now have the tools to control the Rademacher complexity a neural network layer by layer. Let $d_0 = d$ be the dimension of the input data, and d_l be the width of layer l in the neural network. Let $\mathbf{c}^0, \mathbf{c}^1, \dots, \mathbf{c}^{l-1}$ be non-negative weighting vectors of dimensions d_0, d_1, \dots, d_{l-1} respectively. Let B_0, B_1, \dots, B_{l-1} be non-negative constants. Let the base class be $\mathcal{F}_1 := \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle \mid \mathbf{w} \in \mathbb{R}^d, \sum_{i=1}^d \mathbf{c}_i^0 |\mathbf{w}_i| \leq B_0\}$. Recursively define \mathcal{F}_l for l running through 2 to Λ as

$$\mathcal{F}_l := \left\{ \mathbf{x} \mapsto \sum_{i=1}^{d_{l-1}} \mathbf{w}_i \sigma(f_i(\mathbf{x})) \mid f_i \in \mathcal{F}_{l-1}, \sum_{i=1}^{d_{l-1}} \mathbf{c}_i^{l-1} |\mathbf{w}_i| \leq B_{l-1} \right\}$$

Currently, we leave $\mathbf{c}^0, \mathbf{c}^1, \dots, \mathbf{c}^{l-1}$ as generic non-negative weight vectors to derive a general result, and later we choose them to correspond to the case of Fiedler regularization specifically.

We can now state our main result:

Theorem 6.6 (Rademacher Complexity Bound for Neural Network under Weighted L_1 Penalty) *Let \mathcal{F}_Λ denote the class of Λ -layer neural networks with a fixed feedforward architecture and γ -Lipschitz activations applied element-*

wise, where the weight matrix for the l^{th} layer is denoted $\mathbf{W}^{(l)}$. Assume the input features $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ satisfy $\|\mathbf{x}_i\|_\infty \leq C$ for all \mathbf{x}_i . Also assume that for every layer l , the \mathbf{c}^{l-1} -weighted L_1 norm of every row of $\mathbf{W}^{(l)}$ is upper bounded by B_{l-1} . Then we have

$$\hat{R}_N(\mathcal{F}_\Lambda) \leq (2\gamma)^{\Lambda-1} \left(\prod_{l=1}^{\Lambda} \frac{B_{l-1}}{\min(\mathbf{c}^{l-1})} \right) C \sqrt{\frac{2 \log(2d)}{N}}$$

Proof: It is easy to see that the rows of the weight matrices $\mathbf{W}^{(l)}$ corresponds precisely to the weights \mathbf{w} of the linear transformation applied in \mathcal{F}_l . This means that the assumption where each row of $\mathbf{W}^{(l)}$ has \mathbf{c}^{l-1} -weighted L_1 norm at most B_{l-1} corresponds exactly to the constraints in \mathcal{F}_l . Hence we can proceed with the analysis via the function classes \mathcal{F}_l .

We first note that

$$\hat{R}_N(\mathcal{F}_\Lambda) \leq 2\gamma \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})} \hat{R}_N(\mathcal{F}_{\Lambda-1}) \quad (2.6)$$

To see this, use the definition

$$\begin{aligned} \hat{R}_N(\mathcal{F}_\Lambda) &= \mathbb{E}_{\mathcal{R}} \left[\sup_{f \in \mathcal{F}_\Lambda} \frac{1}{N} \sum_{i=1}^N \Omega_i f(\mathbf{z}_i) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \\ &= \mathbb{E}_{\mathcal{R}} \left[\sup_{f_j \in \mathcal{F}_{\Lambda-1}, \sum_{j=1}^{d_{\Lambda-1}} \mathbf{c}_j^{\Lambda-1} \mathbf{w}_j \leq B_{\Lambda-1}} \frac{1}{N} \sum_{i=1}^N \Omega_i \sum_{j=1}^{d_{\Lambda-1}} \mathbf{w}_j \sigma(f_j(\mathbf{z}_i)) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \\ &= \mathbb{E}_{\mathcal{R}} \left[\sup_{f_j \in \mathcal{F}_{\Lambda-1}, \sum_{j=1}^{d_{\Lambda-1}} \mathbf{c}_j^{\Lambda-1} \mathbf{w}_j \leq B_{\Lambda-1}} \sum_{j=1}^{d_{\Lambda-1}} \mathbf{w}_j \frac{1}{N} \sum_{i=1}^N \Omega_i \sigma(f_j(\mathbf{z}_i)) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \end{aligned}$$

Apply Cauchy-Schwartz, get:

$$\leq \mathbb{E}_{\mathcal{R}} \left[\sup_{f_j \in \mathcal{F}_{\Lambda-1}, \sum_{j=1}^{d_{\Lambda-1}} \mathbf{c}_j^{\Lambda-1} \mathbf{w}_j \leq B_{\Lambda-1}} \|\mathbf{w}\|_1 \max_j \left| \frac{1}{N} \sum_{i=1}^N \Omega_i \sigma(f_j(\mathbf{z}_i)) \right| \middle| \{\mathbf{z}_i\}_{i=1}^N \right]$$

Using the constraint $\sum_{j=1}^{d_{\Lambda-1}} \mathbf{c}_j^{\Lambda-1} \mathbf{w}_j \leq B_{\Lambda-1}$, which implies $\|\mathbf{w}\|_1 \leq \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})}$, we have:

$$\begin{aligned} &\leq \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})} \mathbb{E}_{\mathcal{R}} \left[\sup_{f \in \mathcal{F}_{\Lambda-1}} \left| \frac{1}{N} \sum_{i=1}^N \Omega_i \sigma(f(\mathbf{z}_i)) \right| \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \\ &\leq 2 \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})} \mathbb{E}_{\mathcal{R}} \left[\sup_{f \in \mathcal{F}_{\Lambda-1}} \frac{1}{N} \sum_{i=1}^N \Omega_i \sigma(f(\mathbf{z}_i)) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \end{aligned}$$

Applying lemma 6.3 (Talagrand's contraction lemma), get:

$$\begin{aligned} &\leq 2\gamma \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})} \mathbb{E}_{\mathcal{R}} \left[\sup_{f \in \mathcal{F}_{\Lambda-1}} \frac{1}{N} \sum_{i=1}^N \Omega_i f(\mathbf{z}_i) \middle| \{\mathbf{z}_i\}_{i=1}^N \right] \\ &= 2\gamma \frac{B_{\Lambda-1}}{\min(\mathbf{c}^{\Lambda-1})} \hat{R}_N(\mathcal{F}_{\Lambda-1}) \end{aligned}$$

This shows the claim 2.6.

Applying the inequality 2.6 recursively $\Lambda - 1$ times, we obtain

$$\hat{R}_N(\mathcal{F}_{\Lambda}) \leq 2^{\Lambda-1} \gamma^{\Lambda-1} \left(\prod_{l=2}^{\Lambda} \frac{B_{l-1}}{\min(\mathbf{c}^{l-1})} \right) \hat{R}_N(\mathcal{F}_1)$$

Applying corollary 6.5, get:

$$\begin{aligned} \hat{R}_N(\mathcal{F}_{\Lambda}) &\leq 2^{\Lambda-1} \gamma^{\Lambda-1} \left(\prod_{l=2}^{\Lambda} \frac{B_{l-1}}{\min(\mathbf{c}^{l-1})} \right) \frac{B_0}{\min(\mathbf{c}^0)} C \sqrt{\frac{2 \log(2d)}{N}} \\ &= (2\gamma)^{\Lambda-1} \prod_{l=1}^{\Lambda} \frac{B_{l-1}}{\min(\mathbf{c}^{l-1})} C \sqrt{\frac{2 \log(2d)}{N}} \end{aligned}$$

This shows the desired result.

The above result allows us to control the Rademacher complexity of a neural network under a weighted L_1 penalty in a layer-wise fashion. In the context of Fiedler

regularization, the penalty applied is $\sum_{(a,b) \in E} (\mathbf{u}(a) - \mathbf{u}(b))^2 |\mathbf{W}_{ab}|$, where we recall that the \mathbf{W} without superscript denotes the weighted adjacency of the underlying graph of the entire neural network, and \mathbf{u} being a fixed test vector (which can be chosen as the second Laplacian eigenvector \mathbf{v}_2 for exact results). We can rewrite this in matrix form by defining a $n \times n$ matrix \mathbf{U} as $\mathbf{U}_{ab} = (\mathbf{u}(a) - \mathbf{u}(b))^2$. We thus obtain an equivalent expression of the penalty term as $\sum_{(a,b) \in E} \mathbf{U}_{ab} |\mathbf{W}_{ab}|$.

This penalty is separable, and we can split it by layers. Denote the nodes in layer l by V_l and the edges that connects nodes from V_{l-1} to V_l as E_l . We obtain the equivalent expression for the penalty term $\sum_{l=1}^{\Lambda} \sum_{(a,b) \in E_l} \mathbf{U}_{ab} |\mathbf{W}_{ab}^{(l)}|$. This now has a weighted L_1 form that is similar to the layer-wise weighted L_1 constraints used in the function classes \mathcal{F}_l . Thus, the weighting vectors $\mathbf{c}^0, \mathbf{c}^1, \dots, \mathbf{c}^{\Lambda-1}$ used in theorem 6.6 above can now be chosen accordingly.

Recall that

$$\mathcal{F}_l := \left\{ \mathbf{x} \mapsto \sum_{i=1}^{d_{l-1}} \mathbf{w}_i \sigma(f_i(\mathbf{x})) \mid f_i \in \mathcal{F}_{l-1}, \sum_{i=1}^{d_{l-1}} \mathbf{c}_i^{l-1} |\mathbf{w}_i| \leq B_{l-1} \right\}$$

The weights \mathbf{w} in \mathcal{F}_l map the nodes in V_{l-1} to a single node in layer l , and corresponds to a row of the matrix $\mathbf{W}^{(l)}$. Thus one way to select \mathbf{c}^{l-1} is to pick its b^{th} entry, where b indexes the nodes in V_{l-1} , as the maximum entry inside the b^{th} column of \mathbf{U} . Due to the feedforward nature of the network (without any skip connections), this corresponds to finding the maximum of \mathbf{U} over the edges that connects node b in V_{l-1} to all the nodes in V_l , i.e. $\mathbf{c}^{l-1}(b) = \max_{\{a \mid (a,b) \in E_l\}} \mathbf{U}_{a,b}$. Selecting the max entry along the b^{th} column ensures that the resulting function class \mathcal{F}_Λ is large enough to include the Fiedler regularized neural network.

We then have the following corollary:

Corollary 6.7 (Rademacher Complexity Bound for Fiedler Regularized Neural Network) *Assume the same conditions of Theorem 6.6. In addition, fix a test vector \mathbf{u} for Fiedler regularization, which specifies a matrix \mathbf{U} . For each l from 1 to Λ , select weighting vectors \mathbf{c}^{l-1} according to $\mathbf{c}^{l-1}(b) = \max_{\{a|(a,b) \in E_l\}} \mathbf{U}_{a,b}$, where b indexes V_{l-1} . The empirical Rademacher complexity of the resulting function class is upper bounded as follows:*

$$\hat{R}_N(\mathcal{F}_\Lambda) \leq (2\gamma)^{\Lambda-1} \left(\prod_{l=1}^{\Lambda} \frac{B_l}{\min_{\{b \in V_{l-1}\}} (\max_{\{a|(a,b) \in E_l\}} \mathbf{U}_{a,b})} \right) C \sqrt{\frac{2 \log(2d)}{N}}$$

Proof: Simply set the vectors \mathbf{c}^{l-1} according to $\mathbf{c}^{l-1}(b) = \max_{\{a|(a,b) \in E_l\}} \mathbf{U}_{a,b}$ and apply Theorem 6.6.

There are more recent results (Golowich et al., 2018) that use a more refined Rademacher analysis to remove the exponential dependency on depth from the $(2\gamma)^{\Lambda-1}$ term. However, building on such results to obtain more refined bounds is a future direction, and does not affect the interpretation/derivations for Fiedler regularization, which controls the norm of the weight matrices.

The above bound in corollary 6.5 yields insight into how Fiedler regularization works. A larger value for $\mathbf{U}_{ab} = (\mathbf{u}(a) - \mathbf{u}(b))^2$ implies that the edge (a, b) is “less important” for connectivity. Thus the max operation $\max_{\{a|(a,b) \in E_l\}} \mathbf{U}_{a,b}$ can be interpreted as looking for the least important edge that node b is associated with. The subsequent $\min_{\{b \in V_{l-1}\}}$ operation finds the node in layer $l - 1$ whose least important edge has the best connectivity. It can be thought of as a way of characterizing the connectivity bottleneck inside a layer in a minimax/worst-case manner. The higher

this value, the stronger the penalization, and hence the smaller the Rademacher complexity.

We can combine theorem 6.2 and corollary 6.7 to obtain the generalization error bound on a Fiedler regularized neural network.

Corollary 6.8 (Generalization Error Bound for Fiedler Regularized Neural Network) *Let $\mathcal{H} := \{(x, y) \mapsto \mathcal{L}(f(x), y) : f \in \mathcal{F}_\Lambda\}$, where \mathcal{L} is a 1-Lipschitz continuous loss function. Then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ under the product measure \mathcal{P}^N , the following inequality holds:*

$$\begin{aligned} & \sup_{h \in \mathcal{H}} \left[\frac{1}{N} \sum_{i=1}^N h(\mathbf{z}_i) - \mathbb{E}_{\mathcal{P}}(h(\mathbf{z})) \right] \\ & \leq 2(2\gamma)^{\Lambda-1} \left(\prod_{l=1}^{\Lambda} \frac{B_l}{\min_{\{b \in V_{l-1}\}} (\max_{\{a, (a,b) \in E_l\}} \mathbf{U}_{a,b})} \right) C \sqrt{\frac{2 \log(2d)}{N}} + O\left(\sqrt{\frac{\log(1/\delta)}{N}}\right), \end{aligned}$$

where each \mathbf{z}_i is distributed according to \mathcal{P} independently.

Proof: Apply Talagrand’s contraction lemma to obtain $\hat{R}_N(\mathcal{H}) \leq 1 \times \hat{R}_N(\mathcal{F}_\Lambda)$, and then apply theorem 6.2.

2.7 Supervised Classification: Experiments and Results

Deep/multilayer feedforward NNs are useful in many classification problems, ranging from popular image recognition tasks to scientific and biomedical problems such as classifying diseases. We examine the performance of Fiedler regularization on the standard benchmark image classification datasets MNIST and CIFAR10. We also tested Fiedler regularization on the TCGA RNA-Seq PANCAN tumor classification dataset (Weinstein et al., 2013) from the UCI Machine Learning Repository. We compare the performance of several standard regularization approaches for NNs on

these datasets, including Dropout, L_1 regularization and weight decay. Extension of such experiments to other classification tasks is straightforward.

The purpose of the experiments below is not to use the deepest NNs, the latest architectures or the most optimized hyperparameters. Nor is the purpose to show the supremacy of NNs versus other classification methods like random forests or logistic regression. Rather, we attempt to compare the efficacy of Fiedler regularization against other NN regularization techniques as a proof of concept. Extensions to more complicated and general network architectures are explored in the discussion section.

For all our experiments, we consider 5-layer feedforward NNs with ReLU activations and fully connected layers. We used PyTorch 1.4 and Python 3.6 for all experiments. We adopt stochastic gradient descent with a momentum of 0.9 for optimization and a learning rate of 0.001. To select the dropping probability for dropout, as well as the regularization hyperparameter for L_1 , Fiedler regularization and weight decay, we performed a very rough grid search on a small validation dataset. The dropout probability is selected to be 0.5 for all layers, and the regularization hyperparameters for L_1 , Fiedler regularization and weight decay are 0.001, 0.01 and 0.01 respectively. All models in the experiments were trained under the cross-entropy loss. Each experiment was run 5 times, with the median and the standard deviation of the performances reported. All experiments were run on a Unix machine with an Intel Core i7 processor. The code used for the experiments can be found at the Github repository (<https://github.com/edictam/FiedlerRegularization>).

2.7.1 MNIST

Dataset and setup. MNIST is a standard handwriting recognition dataset that consists of 60,000 28×28 training images of individual hand-written digits and 10,000 testing images. We picked the hidden layers of our NN to be 500 units wide. We

Table 2.1: Classification accuracies for MNIST under various regularization schemes (units in percentages)

REGULARIZATION	TRAINING	TESTING
L ₁	90.32 ± 0.17	90.25 ± 0.35
WEIGHT DECAY	95.12 ± 0.06	94.98 ± 0.07
DROPOUT	94.52 ± 0.07	94.5 ± 0.18
FIEDLER	96.54 ± 0.08	96.1 ± 0.12

used a batch size of 100 and the networks were trained with 10 epochs.

Results. The results for MNIST are displayed in Table 1. For the MNIST dataset, we obtained very good accuracies for all the methods, with Fiedler regularization standing out, followed by weight decay and dropout. The high accuracies obtained for the MNIST dataset with feedforward NNs are consistent with results from previous studies (Srivastava et al., 2014). Fiedler regularization showed gains over its competitors.

2.7.2 CIFAR10

Dataset and setup. CIFAR10 is a benchmark object recognition dataset that consists of $32 \times 32 \times 3$ down-sampled RGB color images of 10 different object classes. There are 50,000 training images and 10,000 test images in the dataset. We picked the hidden layers of our NN to be 500 units wide. We used a batch size of 100 and the networks were trained with 10 epochs.

Results. The results for CIFAR10 are displayed in Table 2. While this is a more difficult image classification task than MNIST, the ordering of performances among the regularization methods is similar. Fiedler regularization performed the best, followed by dropout, weight decay and L₁ regularization.

Table 2.2: Classification accuracies for CIFAR10 under various regularization schemes (units in percentages)

REGULARIZATION	TRAINING	TESTING
L ₁	28.52± 0.9	28.88± 1.11
WEIGHT DECAY	52.93 ± 0.17	50.55± 0.39
DROPOUT	46.61± 0.35	44.63± 0.39
FIEDLER	57.99 ± 0.13	52.26± 0.27

2.7.3 TCGA Cancer Classification

Dataset and setup. The TCGA Pan-Cancer tumor classification dataset consists of RNA-sequencing as well as cancer classification results for 800 subjects. The input features are 20531-dimensional vectors of gene expression levels, whereas the outputs are tumor classification labels (there are 5 different tumor types under consideration). We used 600 subjects for training and 200 for testing. Due to the highly over-parametrized nature of this classification task, We picked the width of the hidden layers to be narrower, at 50 units. We used a batch size of 10 and the networks were trained with 5 epochs.

Results The results for the TCGA tumor classification experiment are displayed in Table 3. Fiedler regularization and L₁ had similarly high performances, followed by weight decay. It is interesting that Dropout achieved a relatively low accuracy, slightly better than chance. Since this dataset is relatively small (the testing set has only 200 data points and training set 600), the standard deviation of the accuracies are higher. L₁ and Fiedler regularization, which explicitly induce sparsity, performed the best.

2.7.4 Analysis of Results

We note that both MNIST and CIFAR10 have more training samples than the dimension of their features. The results from MNIST and CIFAR10 largely agree with

Table 2.3: Classification accuracies for TCGA under various regularization schemes (units in percentages)

REGULARIZATION	TRAINING	TESTING
L ₁	91.5 ± 13.73	94.53 ± 12.46
WEIGHT DECAY	57 ± 22.42	60.2 ± 20.94
DROPOUT	25.33 ± 5.9	23.88 ± 7.36
FIEDLER	93.33 ± 20.31	90.55 ± 20.73

each other and confirm the efficacy of Fiedler regularization. Under this setting, other regularization methods like dropout and weight decay also exhibited decent performance.

On the other hand, in the TCGA dataset, where the dimension of the input features (20531) is much higher than the number of training samples (600), L₁ and Fiedler regularization, which explicitly induce sparsity, performed substantially better than dropout and weight decay. An inspection of the TCGA dataset suggests that many of the gene expression levels in the input features are 0 (suggesting non-expression of genes), which likely implies that many of the weights in the network, particularly at the input layer, are not essential. It is thus not surprising that regularization methods that explicitly induce sparsity perform better in these “large p, small n” scenarios, often found in biomedical applications.

We remark that Fiedler regularization enjoys practical running speeds that are fast, generally comparable to (but slightly slower than) that of most commonly used regularization schemes such as L₁ and dropout. The running time of Fiedler regularization could likely be improved with certain implementation-level optimizations to speed up the software. Performances would also likely improve if more refined grid searches or more sophisticated tuning parameter selection methods like Bayesian optimization are adopted.

2.8 Discussion

The above experiments demonstrated several points of interest. The poorer performance of L_1 regularization in MNIST and CIFAR10 stands in sharp contrast to the much higher performance of Fiedler regularization, a weighted L_1 penalty. It is generally acknowledged that L_1 regularization does not enjoy good empirical performance in deep learning models. The precise reason why this happens is not exactly known. Previous studies have adopted a group-lasso formulation for regularization of deep NNs and have obtained good performance (Scardapane et al., 2017). These results suggest that modifications of L_1 through weighting or other similar schemes can often drastically improve empirical performance.

We have tracked the algebraic connectivity of the NNs during the training process in our experiments. In general, without any regularization, the NNs tend to become more connected during training, i.e. their Fiedler value increases. In the Fiedler regularization case, the connectivity is penalized and therefore decreases during training in a very gradual manner. Interestingly, in the L_1 case, the algebraic connectivity of the NN can decrease very quickly during training, often leading to disconnection of the network very early in the training process. This is likely related to L_1 regularization’s uniform penalization of all weights. It is therefore difficult to choose the regularization hyperparameter for L_1 : if it is too small, sparsity induction might occur too slowly and we would under-regularize; if it is too big, we risk over-penalizing certain weights and lowering the model’s accuracy. One advantage of a weighted scheme such as Fiedler penalization thus lies in its ability to adaptively penalize different weights during training.

While we only considered relatively simple feedforward, fully connected neural architectures, potential extensions to more sophisticated structures are straightforward. Many convolutional NNs contain fully connected layers after the initial convolutional

layers. One could easily extend Fiedler regularization to this case. In the context of ResNets, where there are skip connections, the spectral graph properties we have utilized still hold, and hence Fiedler regularization could be directly applied. In the non-classification setting, autoencoders exhibit very natural graph structures in the form of a bottleneck, and it is an open direction to study graph connectivity based penalizations, such as Fiedler regularization, in that setting. The spectral graph theory setup adopted in this paper generally holds for any undirected graph. An open direction is to establish appropriate spectral graph theory for regularization of directed graphs, which would be useful in training recurrent NNs.

While Fiedler regularization leads to sparsely connected NNs in theory, in practice it often takes a higher penalty value or longer training time to achieve sparsity with Fiedler regularization. This might be due in part to the optimization method chosen. It is known that generally SGD does not efficiently induce sparsity in L_1 penalized models, and certain truncated gradient methods (Langford et al., 2009) might prove more effective in this setting.

We remark that while Fiedler regularization emphasizes regularizing based on graphical/connectivity structure, global penalization approaches such as Dropout, L2 etc could still prove useful. One could combine the two regularization methodologies to achieve simultaneous regularization.

There are many important statistical and machine learning models that can be cast as special cases of shallow neural networks, e.g. factor analysis, linear regression etc. One natural direction of further investigation is to apply Fiedler regularization to these models.

There are intimate connections between the graph cut/edge expansion perspective considered above, and the geometric notion of isoperimetry. The Laplacian matrix considered above could be viewed as a discretized analog of the Laplace-Beltrami operator. An open direction is to investigate continuous relaxations of the Fiedler

regularization approach.

On the generalization error bound analysis, there is an independent line of work (Neyshabur et al., 2017) that utilizes the spectral norm of the neural network’s weight matrices to provide margin bounds on the generalization error. Their focus is on utilizing the spectral norms of the weight matrices for the layers, whereas our focus is on leveraging the global structure via the spectrum of the Laplacian of the NN’s underlying graph.

Lastly, we have adopted a version of spectral graph theory that considers the un-normalized (combinatorial) Laplacian $\mathbf{L} = \mathbf{D} - |\mathbf{W}|$ as well as the edge expansion of the graph. A similar theory for regularization could be developed for the normalized Laplacian $\mathbf{L}' = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}|\mathbf{W}|\mathbf{D}^{-\frac{1}{2}}$ and the conductance of the graph, after appropriately accounting for the total scale of the NN. While the combinatorial Laplacian that we considered is related to the notion of RatioCuts, the normalized Laplacian is associated with the notion of NCuts. Both notions could in theory be used for reducing connectivity/co-adaptation of the NN.

Fast-Forwarded Exact Sampling of Spanning Trees

3.1 Introduction

Tree graphs are commonly encountered in statistical modeling. An undirected tree is an acyclic and connected graph $T = (V_T, E_T)$ with nodes $V_T = (1, \dots, m)$ and edges $E_T = \{(j, l)\}$ with $|E_T| = m - 1$. By designating a node $r \in V_T$ as the root, one can easily obtain from T a directed tree $\vec{T} = (V_{\vec{T}}, E_{\vec{T}}, r)$, with identical node set $V_{\vec{T}} = V_T$ and directed edges $E_{\vec{T}} = \{(j \rightarrow l)\}$ obtained from E_T by pointing edges away from r . Such tree structures provide succinct ways to capture complex dependencies that arise from a wide range of statistical applications.

In hierarchical modeling, directed trees represent multi-layer dependence structures underlying the observed data. From a generative perspective, we consider a \vec{T} where each node v is equipped with a parameter μ_v . Using this tree, we define a joint probability distribution for data y_1, \dots, y_n and parameters μ_1, \dots, μ_m conditioned on assignment labels $z_i \in (1, \dots, m)$:

$$L(y, \mu \mid \vec{T}, z) = \left\{ \prod_{i=1}^n \mathcal{F}(y_i \mid \mu_{z_i}) \right\} \left\{ \mathcal{R}(\mu_r) \prod_{(j \rightarrow l) \in E_{\vec{T}}} \mathcal{H}(\mu_l \mid \mu_j) \right\}, \quad (3.1)$$

where $\mathcal{H}(\mu_l | \mu_j)$ is the transition probability kernel from μ_j to μ_l , $\mathcal{R}(\mu_r)$ is the marginal kernel for μ_r in the root, and \mathcal{F} is the conditional kernel of the data y_i given the assignment z_i for that observation. There are potentially other parameters characterizing \mathcal{F} and \mathcal{H} , including dependence on covariates via decision trees (Chipman et al., 1998; Castillo and Ročková, 2021) or related ensemble methods (Chipman et al., 2010; Linero and Yang, 2018), but we suppress these temporarily for ease of notation. Model (3.1) induces a partition on $(1, \dots, n)$ via the latent assignments z . In contrast to traditional mixture models, which often assume the μ_k 's to be generated independently from a common distribution, (3.1) characterizes dependence in μ_j and μ_l through an ancestry tree. Ancestors of v include the tree nodes in the path from the root to v . This type of dependence is well motivated in many application areas. The tree can be interpreted as an inferred evolutionary/phylogenetic history in certain biological settings (Huelsenbeck and Ronquist, 2001; Suchard et al., 2001; Neal, 2003), or alternatively as a multi-layer partitioning of a dataset (Heller and Ghahramani, 2005).

It is also common to use a collection, or forest, of trees for flexibility in modeling. Consider

$$L(y | \vec{T}_1, \dots, \vec{T}_K) = \prod_{k=1}^K \left\{ \mathcal{R}(y_{r^{(k)}}) \prod_{(j \rightarrow l) \in E_{\vec{T}_k}} \mathcal{H}(y_l | y_j) \right\}, \quad (3.2)$$

where each $\vec{T}_k = \{V_{\vec{T}_k}, E_{\vec{T}_k}, r^{(k)}\}$ is a component tree, and $(V_{\vec{T}_1}, \dots, V_{\vec{T}_K})$ gives a K -partition of data index $(1, \dots, n)$, where $n = \sum_{k=1}^K |V_{\vec{T}_k}|$. The kernel \mathcal{R} describes how the first point in a group arises, and \mathcal{H} characterizes the conditional dependence of the subsequent points given the previous ones. The use of trees and forests in graphical modeling (Lauritzen, 2011) dates back at least to the single-linkage clustering algorithm (Gower and Ross, 1969), and is recently seen in dependence graph estimation (Duan and Dunson, 2023), contiguous spatial partitioning (Teixeira et al.,

2019; Luo et al., 2021, 2023), and model-based spectral clustering (Duan and Roy, 2023). In addition, likelihood (3.2) has been extended to a mixture of overlapping trees in Bayesian network estimation (Meilă and Jordan, 2000; Meilă and Jaakkola, 2006; Elidan and Gould, 2008).

While there is a rich literature on algorithmic approaches for obtaining point estimates of tree graphs (Prim, 1957; Kruskal, 1956), we are particularly interested in model-based Bayesian approaches. Such methods have the advantage of providing a characterization of uncertainty in estimating trees, while also inferring a generative probability model for the data. Uncertainty quantification is crucial in this context. Algorithms that produce a single tree estimate are ripe for over-interpretation and lack of reproducibility, since in most applications there are many different trees that are almost equally plausible for the data. Naturally, the success of such a Bayesian approach hinges on whether one can conduct inferences based on the posterior distribution of trees in a computationally efficient way.

Sampling from posterior distributions for trees is generally a difficult problem. Current Markov chain Monte Carlo samplers often navigate tree spaces using local modifications, such as pruning and growing moves. Since tree spaces are combinatorial and large in size, these samplers often exhibit poor mixing. A natural alternative is to rely on conjugacy to employ block updates on \vec{T} in Gibbs-type samplers. Our strategy is to view \vec{T} as a spanning tree $\vec{\mathcal{T}}$ under a complete and weighted auxiliary graph $G = (V_G, E_G)$. Here a spanning tree $\vec{\mathcal{T}}$ of G is simply a subtree of G that spans all the nodes of G with edges oriented away from some root node r . For the generative process in (3.1), one can consider a complete graph G with nodes $(1, \dots, m)$, with weighted adjacency matrix $Q \in [0, \infty)^{m \times m}$ of elements $q_{j,l} = \mathcal{H}(\mu_l | \mu_j)$ for every pair (j, l) . Observing that equation (3.1) can be written as $L(y; \mu | \vec{T}, z) = \mathcal{R}(\mu_r) \prod_{(j \rightarrow l) \in E_{\vec{T}}} \left\{ \prod_{z_i=l} \mathcal{F}(y_i | \mu_{z_i}) \mathcal{H}(\mu_l | \mu_j) \right\}$, a natural prior

distribution on the tree structure of $\vec{\mathcal{T}}$ arises:

$$\Pr(\vec{\mathcal{T}}|r) \propto \rho_r \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} q_{j,l}, \quad (3.3)$$

where $\Pr(r)$ is a discrete probability on $(1, \dots, m)$, and ρ_r is inversely proportional to the normalizing constant $\sum_{\vec{\mathcal{T}} \text{ rooted at } r} \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} q_{j,l}$, which could vary with r for directed trees. The resulting posterior distribution on $\vec{\mathcal{T}}$ shares a conjugate product-over-edges form with the prior $\Pr(\vec{\mathcal{T}})$ under the likelihood. The core imperative, then, is to efficiently sample from spanning tree distributions of the form (3.3). Notably, the uniform spanning tree distribution is a special case of (3.3).

The celebrated Aldous–Broder algorithm (Aldous, 1990; Broder, 1989) provides a tractable way to draw exact samples from (3.3). The algorithm proceeds by taking a random walk $X_t = (r, x_1, x_2, \dots, x_t)$ on V_G and stops at the cover time of the walk when all nodes have been visited. By collecting the first entrance edges of X_t , which are the set of edges via which the walk first visited each node, one obtains a spanning tree $\vec{\mathcal{T}}$ rooted at r . By separately specifying a distribution over the root node r , one obtains a distribution on the directed spanning trees $\vec{\mathcal{T}}$. For a graph G containing m nodes, Broder and Karlin (1989) shows an expected cover time of $O(m \log m)$ in well-connected graphs and $O(m^3)$ in the worst case.

The Aldous–Broder algorithm can be empirically slow. In practice, even if the number of nodes m in G is small, the algorithm can get stuck in certain subgraphs for a long time. This issue is inherent to the nature of random walks. Other popular algorithms, such as Wilson’s algorithm based on the loop-erased random walk (Wilson, 1996), also suffer from the same problem.

We formalize this pathological phenomenon using eigenvalues of a normalized Laplacian to characterize bottlenecks in the graph. We then propose a fast-forwarded cover algorithm for exact sampling from the distribution in (3.3) while bypassing

wasteful random walk steps. The key observation is that we do not need to simulate the entire random walk trajectory to obtain the first entrance edges to each node. We derive a closed form expression that allows for direct, fast-forwarded sampling of these first entrance edges via a marginalization argument. This algorithm leads to dramatically faster performance in simulations in the presence of bottlenecks.

Existing spanning tree samplers often perform transitions directly according to the underlying graph’s adjacency matrix Q , restricting their applicability to symmetric/undirected cases. We show that by using an auxiliary matrix W , which is generated from Q under certain transformations, to specify the random walk transition probabilities, our fast-forwarded cover algorithm can be extended to more general scenarios, including cases where the underlying graph has an asymmetric adjacency matrix or the induced Markov chains are irreversible. These results are of independent interest. We illustrate our sampler by fitting a Bayesian dendrogram model on a Massachusetts crimes and communities dataset. The resulting Gibbs sampler exhibits drastically improved mixing performance compared to a reversible jump sampler based on local moves.

3.2 Method

3.2.1 Background on Aldous–Broder Algorithm

We review and summarize results on the Aldous–Broder algorithm (Aldous, 1990; Broder, 1989) in this section. This algorithm samples a random spanning tree $\vec{\mathcal{T}}$ from the underlying graph G based on a random walk. Consider a weight matrix $W \in [0, \infty)^{m \times m}$ which is used to specify the random walk transition probabilities on G . For simplicity, it suffices for now to consider W as the graph’s weighted adjacency matrix Q in (3.3), an important special case that applies when Q is symmetric. However, our results hold for more general cases where the transition matrix W is some specific transformation of Q . A detailed discussion is in Section 2.4.

We use W to specify a random walk over the state space $V_G = (1, \dots, m)$. This random walk is a discrete-time Markov chain $X = (x_0, x_1, \dots)$ with transition probabilities:

$$p_{j,l} = \Pr(x_{t+1} = l \mid x_t = j) = \frac{w_{j,l}}{d_j}, \quad d_j = \sum_{v=1}^m w_{j,v}, \quad (3.4)$$

where $d_j > 0$ and w denotes entries of W . We assume the Markov chain is irreducible: for any pair of (j, l) , there exists $t > 0$ such that $\Pr(x_t = l \mid x_0 = j) > 0$. We denote the random walk up to time t by $X_t = (x_0, x_1, \dots, x_t)$, and the invariant distribution of $x_t : t \rightarrow \infty$ by (π_1, \dots, π_m) . The result of Broder (1989) further assumes reversibility of the Markov chain, $\pi_j p_{j,l} = \pi_l p_{l,j}$; however, this condition is not needed here.

The Aldous–Broder algorithm proceeds as follows: initialize the walk x_0 at root r ; perform the random walk until all nodes are visited at the cover time \hat{t} ; construct a directed spanning tree $\vec{\mathcal{T}}$ with the edge set as the set of first-entrance edges all pointed away from r . Formally,

$$E_{\vec{\mathcal{T}}} = \bigcup_{j \in (V_G \setminus r)} \left\{ (x_{(t_j-1)} \rightarrow x_{t_j}) : t_j = \min_{1 \leq t \leq \hat{t}} (t : x_t = j) \right\}. \quad (3.5)$$

The following theorem characterizes the induced distribution of $\vec{\mathcal{T}}$.

Theorem 1 (Extended Aldous–Broder). *Letting $\vec{\mathcal{T}}$ be generated as above, then*

$$\Pr(\vec{\mathcal{T}} \mid r) \propto \left\{ \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} \frac{p_{j,l} \pi_j}{\pi_l} \right\} \frac{1}{\pi_r} \propto \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} (p_{j,l} \pi_j), \quad (3.6)$$

where the probability is normalized over all $\vec{\mathcal{T}}$ rooted from r .

The first term in (3.6) is based on an adaptation of Theorem 3 of Fredes and Marckert (2023), where a directed tree with edges pointed towards the root is used, as is standard in the probability literature. While this is the opposite of our choice

of orientation, the mathematical results carry over. The second term is based on the fact that $\{\prod_{(j \rightarrow l) \in E_{\vec{T}}} \pi_l\} \pi_r = \prod_{j=1}^m \pi_j$, which is invariant to $E_{\vec{T}}$ and hence can be omitted.

In Section 2.4, we describe how one can choose W as some appropriate transformation of Q so that (3.6) becomes the product form shown in (3.3) under general settings without requiring matrix symmetry for W . For now in Section 2.2, we focus on computational aspects and discuss bottleneck effects that impact the cover time of the Aldous–Broder algorithm.

3.2.2 Bottlenecks in Random Walk Cover Algorithm

The Aldous–Broder algorithm is only efficient if there is a small cover time \hat{t} to reach all nodes in V_G . However, when there are bottlenecks in the graph that lead to close-to-zero probabilities to move from the visited nodes U to the unvisited nodes $\bar{U} = V_G \setminus U$, the random walk can spend a long time within U . We now characterize the consequences from this curse of bottlenecks.

Figure 3.1 illustrates three common challenges through example graphs. The first two graphs are either directed or undirected, while the third is directed. First, as in panel (a), there may be disjoint sets of nodes that are only weakly connected, in the sense that the transition probability between sets are small. Second, as in panel (b), there may be nodes that are isolated with few edges incident to them, and each edge having low transition probability. In this case, the random walk cover algorithm may visit most of nodes within a short time, but faces difficulties reaching the last few. Third, as in panel (c), in a directed graph, due to edge weight asymmetry, there may be a much higher probability to remain in a node set that has been visited.

Regardless of the specific scenario, the curse of bottlenecks can be understood as a close-to-zero probability to move to the unvisited set, marginalized over both the current node and the potential arrival node amongst unvisited nodes.

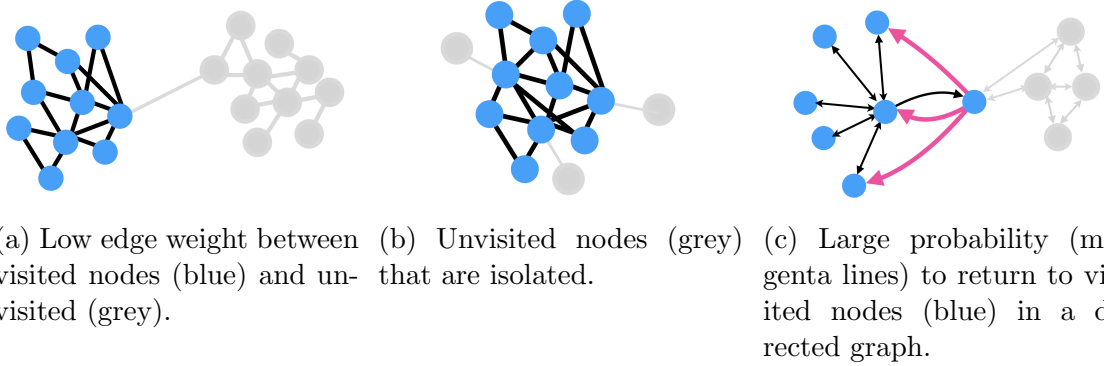


FIGURE 3.1: Illustration of some commonly encountered graphs with curse of bottlenecks — a low transition probability to move from the visited nodes (blue) to those unvisited (grey).

We now formally characterize this phenomenon. Consider a strictly increasing node set $(r) = U_1 \subset U_2 \subset \dots \subset U_m = V_G$, recording the history of visited nodes by the random walk, with $U_k = \cap_{t=0}^{\hat{t}_k} X_t$ and $\hat{t}_k = \min(t \geq 0 : |\cap_{i=0}^t X_i| = k)$, where we refer to \hat{t}_k as the partial cover time for k nodes. The probability of transitioning outside of U_k at time $t + 1$, conditional on the walk remaining within U_k from the beginning through time t is

$$\begin{aligned} \Pr(x_{t+1} \in \bar{U}_k \mid x_t \in U_k) &= \sum_{j \in U_k} \left\{ \frac{\sum_{l \in \bar{U}_k} w_{j,l}}{d_j} \Pr(x_t = j \mid x_{i \leq t} \in U_k) \right\} \leq \max_{j \in U_k} \left(\frac{\sum_{l \in \bar{U}_k} w_{j,l}}{d_j} \right) \\ &= \left(\min_{j \in U_k} \frac{\sum_{l' \in U_k} w_{j,l'}}{\sum_{l \in \bar{U}_k} w_{j,l}} + 1 \right)^{-1}, \end{aligned}$$

where the inequality uses the fact that the weighted average over $j \in U_k$ is less than or equal to the maximum, and the last equality is based on decomposing $d_j = \sum_{l \in \bar{U}_k} w_{j,l} + \sum_{l' \in U_k} w_{j,l'}$. Therefore, if the total outgoing weights to \bar{U}_k are dominated by the weights of staying within U_k , in the sense that $\sum_{l \in \bar{U}_k} w_{j,l} \ll \sum_{l' \in U_k} w_{j,l'}$, it is unlikely that $x_{t+1} \in \bar{U}_k$. With the above ingredients, we are ready to quantify the expected cover time.

Theorem 2. For a history of visited nodes (U_1, \dots, U_m) , the expected cover time $\hat{t} = \hat{t}_m$ has

$$\mathbb{E}(\hat{t}; U_1, \dots, U_m) \geq (m - 1) + \sum_{k=1}^{m-1} \min_{j \in U_k} \frac{\sum_{l' \in U_k} w_{j,l'}}{\sum_{l \in \bar{U}_k} w_{j,l}}.$$

Remark 3. The lower bound applies to any $W \in [0, \infty)^{m \times m}$ that leads to an irreducible Markov chain. In practice, when the random walk cover algorithm appears to be stuck at a certain U_k , one can directly use $\min_{j \in U_k} (\sum_{l' \in U_k} w_{j,l'}) / (\sum_{l \in \bar{U}_k} w_{j,l})$ as an estimate for the expected time to leave.

The above result is a quantification for a specific node history of (U_1, \dots, U_m) . One can obtain a worst case expected cover time by maximizing over all possible sequences of (U_1, \dots, U_m) . We are chiefly interested in W matrices that satisfy a circulation condition: $\sum_{l=1}^m w_{j,l} = \sum_{k=1}^m w_{k,j}$, $j = 1, \dots, m$, which includes symmetric W with $w_{j,l} = w_{l,j}$ as a special case.

Theorem 4. For a random walk X based on weight matrix W satisfying the circulation condition, the worst case expected cover time \hat{t} has

$$\max_{\text{all } (U_1, \dots, U_m)} \mathbb{E}(\hat{t}; U_1, \dots, U_m) \geq \frac{1}{M\sqrt{\lambda_2(\mathcal{L})}} + m - 2,$$

with $M = \sup_{t \geq 0} \sup_j \{Pr(x_t = j) / \pi_j\}$, and $\lambda_2(\mathcal{L})$ the second smallest eigenvalue of the normalized graph Laplacian \mathcal{L} .

Remark 5. Although the above lower bound on expected cover time is inspired by earlier works (Matthews, 1988; Lovász and Winkler, 1993; Levin and Peres, 2017), our result is distinguished by a directly computable lower bound. The constant M^{-1} has a lower bound $\min_j \pi_j$ for any root distribution for $x_0 = r$. In the special case where the root node is drawn from the invariant distribution $r \sim \pi$, $M = 1$ since $Pr(x_t = j) = \pi_j$ for any $t \geq 0$.

3.2.3 Skipping Bottlenecks via Fast-forwarding

With the curse of bottlenecks on the cover time of random walks established, we now exploit a useful fact — when transforming the random walk trajectory $X_{\bar{\mathcal{T}}}$ into the spanning tree edges $E_{\bar{\mathcal{T}}}$, one only needs the first entrance edges to each node. It is unnecessary to simulate entire random walk trajectories until reaching a new node, if the marginal distribution of the next first entrance edge can be directly sampled. We formalize this idea below.

We set up notation here. At any time t , suppose we have visited nodes in U , and let $e_j^t = 1$ if the walk is at node $j \in U$ at time t and first exits U at time $t + 1$. Let $e_j^t = 0$ otherwise. Given a starting location $x_{t_0} = j_0 \in U$ at some time t_0 , drawing the first entrance edge into \bar{U} at $t_0 + \delta$ for some fixed $\delta \in \mathbb{Z}_+$ can be understood as the result of the following events,

$$(e_{j_0}^{t_0} = 0) \Rightarrow (x_{t_0+1} = j_1 \in U) \Rightarrow (e_{j_1}^{t_0+1} = 0) \Rightarrow \cdots \Rightarrow (x_{t_0+\delta} = j' \in U) \Rightarrow (e_{j'}^{t_0+\delta} = 1).$$

The trajectory of the walk stays within U from time t_0 until $t_0 + \delta$ and exits from node $j' \in U$ to node $l' \in \bar{U}$ at time $t_0 + \delta + 1$. This implies the first-entrance edge $j' \rightarrow l'$. We write

$$\eta_j = \Pr(e_j^t = 1), \quad (1 - \eta_j) = \Pr(e_j^t = 0) = \frac{\sum_{k \in U} w_{j,k}}{d_j}.$$

Collecting the η 's into vector form, $\eta_U = (\eta_j : j \in U)$. Let $P_{U,U} = \{p_{j,l} : j \in U, l \in U\}$ represent the principle submatrix of P that corresponds to transitions from nodes in U back into U . Writing a state vector $s_t = \{\Pr(x_t = j' \mid x_{t_0} = j_0)\}_{j' \in U}$, we have the recursive relation $s_{t+1} = P_{U,U}^T s_t$, when the walk has not left U by time $t + 1$. The initial state s_{t_0} is a $|U|$ -dimensional vector with the element corresponding to j_0 set to 1 and all others equal to 0. We then have

$$\Pr(x_{t_0+\delta} = \cdot, x_{t_0+\delta+1} \in \bar{U} \mid x_{t_0} = j_0) = \text{diag}(\eta_U)(P_{U,U}^T)^{\delta-1} s_{t_0}.$$

Here, we started at the initial state at time t_0 , transitioned within the visited nodes U for $\delta - 1$ times, and visited \bar{U} at time $t + \delta + 1$ via $\text{diag}(\eta_U)$. This expression represents the distribution on the nodes that $x_{t_0 + \delta}$ can take. We can further marginalize the above by summing over $\delta \in \mathbb{Z}_{\geq 0}$. Since the summation involves a Neumann series, it has a closed-form for the marginal if the series converges. The following theorem shows a sufficient condition.

Theorem 6. *If $P_{U,U}$ is irreducible (cannot be rearranged to a block upper-triangular matrix via row and column permutations), and there exists at least one $j \in U : \eta_j > 0$, then*

$$\Pr(x_{(\hat{t}_{|U|+1}-1)} = \cdot \mid x_{t_0} = j_0) = \text{diag}(\eta_U)(I - P_{U,U}^T)^{-1}s_{t_0}, \quad (3.7)$$

where $(\hat{t}_{|U|+1} - 1)$ is the time point before moving to a node in \bar{U} .

The above irreducibility condition is satisfied when $w_{j,l} > 0$ for all $j \neq l$. This theorem allows us to sample the first exit edge (j', l') from U to \bar{U} in a straightforward manner. We first draw $j' \in U$ according to the distribution specified by the vector in (3.7). We then take one random step from the drawn node j' via the transition specified by

$$\Pr(x_{\hat{t}_{|U|+1}} = l' \mid x_{(\hat{t}_{|U|+1}-1)} = j', e_j^{(\hat{t}_{|U|+1}-1)} = 1) = w_{j',l'} / \sum_{k \in \bar{U}} w_{j',k} \quad (3.8)$$

to obtain $l' \in \bar{U}$. We refer to steps (3.7) and (3.8) as the fast-forwarding steps.

Remark 7. *Direct calculation of the matrix inverse $(I - P_{U,U}^T)^{-1}$ has $O(|U|^3)$ cost, but solving the linear equation $x : (I - P_{U,U}^T)x = s_{t_0}$ has a low cost $O(\tilde{K}|U|)$ via iterative numerical methods, with \tilde{K} the number of iterations until convergence (Saad, 2003).*

A natural algorithm for drawing a spanning tree performs Aldous–Broder type random walk steps until a bottleneck is reached, and then uses fast-forwarding steps;

this is then repeated until visiting all nodes. We call this approach the fast-forwarded cover algorithm.

The fast-forwarded cover algorithm.

Initialize $\tau = 0$, $x_\tau = r$ and initialize the first entrance step tracker $\alpha = 0$

1. Set $\tau = 1$. Simulate one random walk step to x_τ
2. If x_τ has not been visited before, update $\alpha \leftarrow \tau$.
3. If the steps taken since first entrance step $(\tau - \alpha) \geq \kappa_0$,

a pre-set threshold,

update $x_{\tau+1}$ to j' sampled according to (3.7)

update $x_{\tau+2}$ to l' sampled according to (3.8)

update $\alpha \leftarrow \tau + 2$, and $\tau \leftarrow \tau + 3$. Go to Step 1.

If $(\tau - \alpha) < \kappa_0$, update $\tau \leftarrow \tau + 1$. Go to Step 1.

Repeat the above until all $\alpha = m$,

where m is the number of nodes of the underlying graph.

Collect and return the first entrance edges.

We use a new index τ above, since the iteration number will be different from the underlying random walk time index t , as soon as a fast-forwarding step is used. The algorithm completes when all nodes have been visited, and the total iterations needed is at most $\kappa_0(m - 1)$. In this article, we set $\kappa_0 = 1,000$, which leads to excellent performance in all our experiments.

Before presenting our empirical results, we generalize the random walk cover and our fast-forwarded modifications to be broadly applicable to sampling any directed spanning tree with probability in the form of (3.3). This is accomplished through a careful specification of the transition matrix P , or equivalently W up to row-wise normalization.

3.2.4 Generalizing the Applicability of Cover Algorithms

Given a graph with weighted adjacency matrix Q , our goal is to draw samples from (3.3) using cover algorithms, a terminology we use to include random walk cover and our fast-forwarded modifications. Several cases are of interest.

Case 1, Circulation: We start with the canonical case in which a simple condition $\sum_{l=1}^m q_{j,l} = \sum_{k=1}^m q_{k,j}$ holds for $j = 1, \dots, m$. If we view $q_{j,l}$ as a flow from node j to l , these equalities describe a type of conservation of flow, with Q describing a circulation matrix (Chung, 2005). This includes the special case where Q is symmetric. Under this circulation condition, setting the random walk transition probabilities $w_{j,l}$ as equal to the underlying graph's edge weights $q_{j,l}$ is sufficient for cover algorithms to produce samples satisfying

$$\Pr(\vec{\mathcal{T}} \mid r) \propto \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} q_{j,l}, \quad (3.9)$$

corresponding to $\rho_r = 1$. Therefore, we can draw the root from $\Pr(r)$, and use a cover algorithm with $W = Q$ to obtain samples distributed according to (3.3).

Case 2, General form: For a general $Q \in [0, \infty)^{m \times m}$, the stationary distribution vector π is a deterministic transform of a left eigenvector of the transition probability matrix P . Here, $(p_{j,l} \pi_j) \propto q_{j,l}$ may not hold for all (j, l) . To solve this problem, we introduce an auxiliary Markov chain as a mathematical tool for calculating P . Consider a chain $Y = (y_0, y_1, \dots)$ over V_G with transition kernel:

$$\tilde{p}_{l,j} = \Pr(y_{t+1} = j \mid y_t = l) = \frac{q_{l,j}^*}{\sum_{k=1}^m q_{l,k}^*}, \quad q_{l,j}^* = q_{j,l}.$$

We denote the invariant distribution of $y_t : t \rightarrow \infty$ by $(\pi_1^*, \dots, \pi_m^*)$, which can be numerically computed as a left eigenvector of the transition matrix formed by the $\tilde{p}_{l,j}$'s.

We now form the chain $X = (x_0, x_1, \dots)$ with the transition probability specified as:

$$p_{j,l} = \frac{\tilde{p}_{l,j} \pi_l^*}{\pi_j^*}. \quad (3.10)$$

Reversibility ($p_{j,l} = \tilde{p}_{j,l}$) is not needed; however, we can see that $\sum_j p_{j,l} \pi_j^* = \pi_l^*$ for all l . Therefore, X has the same invariant distribution $(\pi_1, \dots, \pi_m) = (\pi_1^*, \dots, \pi_m^*)$

as Y . As a result, we can run a cover algorithm using P from (3.10). The probability (3.6) in Theorem 1 becomes

$$\Pr(\vec{\mathcal{T}} \mid r) \propto \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} (\tilde{p}_{l,j} \pi_l^*) = \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} \frac{q_{j,l}}{\sum_{k=1}^m q_{k,l}} \pi_l^* \propto \left(\prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} q_{j,l} \right) \left(\frac{\sum_{k=1}^m q_{k,r}}{\pi_r^*} \right), \quad (3.11)$$

using the invariance to $\vec{\mathcal{T}}$ implied by

$$\pi_r^* / \left(\sum_{k=1}^m q_{k,r} \right) \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} \pi_l^* / \left(\sum_{k=1}^m q_{k,l} \right) = \prod_{l=1}^m \pi_l^* / \left(\sum_{k=1}^m q_{k,l} \right)$$

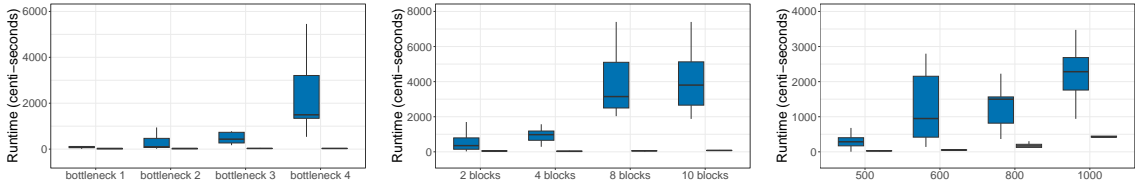
. The above result involves a non-constant $\rho_r = (\sum_{k=1}^m q_{k,r} / \pi_r^*)$, as in (3.3), which may pose inconveniences when one wants to draw from $\Pr(\vec{\mathcal{T}}) \propto h_r \prod_{(j \rightarrow l) \in E_{\vec{\mathcal{T}}}} q_{j,l}$ for some known kernel h_r . To solve this issue, one can first draw the root from $\Pr(r) \propto (h_r / \rho_r)$, and then use a cover algorithm to draw from $\Pr(\vec{\mathcal{T}} \mid r)$, so that ρ_r is effectively canceled.

3.3 Simulations

To investigate the gain in computational efficiency of our fast-forwarded cover algorithm over the random walk cover algorithm, we conduct several simulations and compare their runtime based on R code on an M1-chip laptop.

First, we assess the impact of size of bottlenecks, quantified via $1/\sqrt{\lambda_2(\mathcal{L})}$, on wall-clock runtime. We simulate a graph with 500 nodes, where we partition the nodes V into two blocks with $|V_1| = |V_2| = 250$. The graph is represented by a symmetric weight matrix $W \in \mathbb{R}^{500 \times 500}$ with each $w_{j,l} = u_{j,l} b_{j,l}$. The factor $u_{j,l}$ representing the size of the weight is simulated from $\text{Unif}(0, 1)$. We set $b_{j,l} = 1$ for those j and l that are in the same node partition, and $b_{j,l} \sim \text{Bern}(\zeta)$ for those j and l that are in different partitions. This creates two complete subgraphs connected

by a small number of edges having small weight. We conduct experiments under different edge densities with ζ from $(0.5, 0.1, 0.05, 0.01)$, corresponding to a range of bottleneck values near $1/\sqrt{\lambda_2(\mathcal{L})} = (249, 560, 786, 1738)$. Under each value of ζ , we draw 10 spanning trees using random walk cover algorithm, and 10 trees using fast-forwarded cover algorithm. Figure 3.2(a) shows that the runtime for the random walk algorithm increases rapidly as the bottleneck size increases, while the fast-forwarded algorithm runtime is almost unaffected by bottleneck size. Similar plots are shown in the supplementary materials, where we compare the Aldous–Broder algorithm and the fast-forwarded algorithm in the same setting but in number of iterations.



(a) Runtime for different bottlenecks $1/\sqrt{\lambda_2(\mathcal{L})}$. (b) Runtime over different numbers of blocks and hence bottlenecks. (c) Runtime over different numbers of nodes.

FIGURE 3.2: Comparisons of runtime between random walk cover (blue) and fast-forwarded cover (grey) algorithm. In each setting, the fast-forwarded algorithm runtime has a small mean and variance, so that each grey box appears close to a thin line.

Second, we assess effects of the number of blocks on runtime; more blocks implies more bottlenecks. We simulate $W \in \mathbb{R}^{600 \times 600}$ according to $w_{j,l} = u_{j,l} b_{j,l} c_{j,l}$. The factors $u_{j,l}$ and $b_{j,l}$ are simulated as above, except we partition the 600 nodes into K blocks (V_1, \dots, V_K) each of node size $(600/K)$, and use $b_{j,l} = b_{l,j} \sim \text{Bern}(0.001)$ and $c_{j,l} = 0.005/K$ for those $(j, l) : j \in V_k, l \in V \setminus V_k$. The factors $c_{j,l}$ are chosen so that empirically the size of $1/\sqrt{\lambda_2(\mathcal{L})}$ remains roughly the same. We vary K in $(2, 4, 8, 10)$, draw 10 spanning trees using the random walk algorithm, and 10 trees using the fast-forwarded cover algorithm. Figure 3.2(b) shows that the runtime of

random walk cover increases quickly as the number of blocks increases, while the fast-forwarded algorithm is again almost unaffected.

Lastly, we assess scalability for larger numbers of nodes ranging in (500, 600, 800, 1000) in the two block case using $b_{j,l} = b_{l,j} \sim \text{Bern}(0.1)$. Figure 3.2(c) shows that the runtime for the fast-forwarded cover algorithm increases at an approximately linear rate in node size m , while the runtime for the random walk cover algorithm increases much faster.

3.4 Bayesian Dendrogram Inference for Crime and Communities Data

We demonstrate the use of our algorithm in inferring a Bayesian dendrogram. We consider a hierarchical model in the form of (3.1) for continuous $y_i \in \mathbb{R}^{\vec{d}}$ with \vec{T} rooted at node 1, and we choose multivariate normal conditional densities as:

$$\mathcal{F}(y_i | \mu_{z_i}) = \phi(y_i; \mu_{z_i}, \Sigma), \quad \mathcal{H}(\mu_l | \mu_j, \Sigma) = \phi(\mu_l; \mu_j, \lambda^{-1}\Sigma).$$

We assign priors $\Pi_0(\vec{T}) \propto 1$, $z_i \stackrel{iid}{\sim} \text{Categorical}(\tilde{\pi}_1, \dots, \tilde{\pi}_{\tilde{K}})$, $(\tilde{\pi}_1, \dots, \tilde{\pi}_{\tilde{K}}) \sim \text{Dir}(\alpha, \dots, \alpha)$, $\Sigma \sim W^{-1}(\nu, \Sigma_0)$. Since \mathcal{F} and \mathcal{H} have a shared covariance Σ up to a scale shift $\lambda > 0$, a conjugate normal-inverse Wishart prior can be chosen for μ and Σ .

Our interest is inferring the posterior on \vec{T} , but we also have uncertainty in allocations z_i and component-specific parameters (μ_k) . The resulting joint posterior distribution is highly complex. Due to the computational challenges, the literature tends to avoid characterizing uncertainty in \vec{T} . For example, for Bayesian hierarchical clustering, Heller and Ghahramani (2005) uses a hypothesis testing criterion to iteratively merge clusters, whereas Heard et al. (2006) combines clusters based on a metric capturing inter-cluster closeness. Alternatively, one can use a Markov chain Monte Carlo sampling algorithm. Classical algorithms rely on making local changes in \vec{T} using reversible-jump Metropolis-Hastings (Chipman et al., 1998; Denison et al., 1998). Such algorithms tend to be very inefficient. Wu et al. (2007) considers adding

a move allowing for larger tree restructuring to improve mixing, but with high per iteration expense for large graphs.

Our new spanning tree sampler can be exploited to bypass these computational challenges. We take an overfitted modeling approach, by considering an encompassing tree $\vec{\mathcal{T}}$ with \tilde{m} nodes rooted at 1, with \tilde{m} sufficiently large to provide an upper bound on the true value $\tilde{m} \geq m$. This allows us to build a blocked Gibbs sampler based on drawing from $\Pi(z \mid y, \mu, \vec{\mathcal{T}})$, $\Pi(\mu \mid y, z, \vec{\mathcal{T}})$ and $\Pi(\vec{\mathcal{T}} \mid \mu)$, with $\vec{\mathcal{T}}$ a spanning tree for \tilde{m} nodes, in addition to steps updating other parameters.

After obtaining a posterior sample, we can marginalize out the redundant nodes and change each sampled spanning tree $\vec{\mathcal{T}}$ to a reduced dendrogram \vec{T} , while maintaining an equivalent generative model for the data. Given a sample of (z_1, \dots, z_n) , and \vec{T} initialized at $\vec{\mathcal{T}}$, we use the following pruning procedures corresponding to integrating out the densities related to j :

- If j is an empty leaf node, without any downstream edge ($j \rightarrow l$) and with $\sum_{i=1}^n 1(z_i = j) = n_j = 0$, we remove j and $(k \rightarrow j)$ from \vec{T} .
- If j only has two edges ($k \rightarrow j$) and ($j \rightarrow l$), and $n_j = 0$, we remove j and replace the two edges by $(k \rightarrow l)$ in \vec{T} .

We iterate the above pruning steps on the tree's nodes until we cannot reduce the size of \vec{T} any further. The spanning tree $\vec{\mathcal{T}}$ can be viewed as a latent variable that facilitates model specification and posterior computation for the dendrogram \vec{T} . We refer to this approach as a spanning tree-augmented dendrogram.

To illustrate this model, we consider an application to the Community and Crimes dataset from the University of California at Irvine Machine Learning Repository. The dataset consists of socioeconomic attributes from the 1990 United States Census, crime statistics from the 1995 Uniform Crime Report, and law enforcement attributes from the 1990 Law Enforcement Management and Administrative Statistics Survey.

We focus on economic data from the state of Massachusetts, where there are $n = 123$ relevant entries, each corresponding to a community in the state, with $\tilde{d} = 2$ continuous attributes: the community’s median income and median rent. These attributes are log-transformed and standardized to have sample mean 0 and marginal sample variance 1. Our focus is on characterizing variability in these economic attributes by grouping the communities hierarchically. A dendrogram is natural for this purpose.

We choose priors to favor node parameters μ_j that are broadly spread across the support of the data, with relatively few data points associated with each μ_j . This is achieved by choosing $\nu = n$, $\Sigma_0 = 0.2^2 I_{\tilde{d}}$, $\lambda = 0.25$ and $\tilde{m} = \lfloor n/4 \rfloor$. To favor effective deletion of extra unnecessary clusters, we follow common practice in the overfitted mixtures literature (Van Havre et al., 2015) and choose a symmetric Dirichlet with small concentration parameter ($\alpha = 0.1$) for the weights $\tilde{\pi}_1, \dots, \tilde{\pi}_{\tilde{m}}$. Since the data have been centered, we fix root choice $r = 1$ and $\mu_1 = 0$. There are implicit Bayesian Occam’s razor effects that favor the induced dendrogram \vec{T} to be small. As in other Bayesian mixture models, the marginal likelihood will tend to decrease if data are over clustered, favoring setting $n_k = 0$ to automatically remove some of the clusters. This tendency is furthered by our symmetric Dirichlet prior with precision close to zero for the cluster weights. In addition, the uniform prior on \vec{T} leads to higher weight on dendrograms with few nodes as such dendrograms have more ways to be marginalized (pruned) from a \tilde{m} -node \vec{T} .

Using a Gibbs sampler, we can update each term above using a closed-form full conditional distribution in each iteration. We provide the details in the supplementary material. To show computational advantages of this Gibbs sampler, we compare with sampling the posterior for a directly specified dendrogram using a reversible-jump Markov chain Monte Carlo sampler. The model is almost the same as our spanning tree-augmented version, with the same choice of \mathcal{F} and \mathcal{G} , priors for the parameters, and upper bound \tilde{m} on the number of nodes in \vec{T} . However, we allow

the number of nodes in \vec{T} to vary; hence some nodes amongst $(1, \dots, \tilde{m})$ might not be on \vec{T} . In the likelihood, we replace $\mathcal{F}(y_i | \mu_k)$ by zero if k is not a node of \vec{T} , and use the prior $\Pi_0(\vec{T}) \propto 0.1^{|\vec{T}|}$ to favor small dendrograms. Accordingly, we use birth/death proposals to add/remove nodes from \vec{T} , and a Metropolis-Hastings criterion to accept or reject each proposal. We provide details in the supplementary material.

We run each sampler for 5000 iterations, with a burn-in of 3500 iterations. In wall-clock time, both the Gibbs sampler using our fast-forward cover algorithm and the reversible-jump Markov chain Monte Carlo sampler run for approximately 0.5 to 1 hour. However, there are dramatic differences in mixing performance. The reversible-jump Markov chain Monte Carlo sampler tends to be stuck in certain states for a long time, while the spanning tree-based Gibbs sampler shows excellent mixing. Figure 3.3 compares the mixing for the number of non-empty leaves of the dendrogram, with results for other summaries shown in the supplementary material. We compare effective sample sizes per iteration in Table 3.4.

To quantify uncertainty around the obtained clusters, as well as to visualize the hierarchical structures obtained, we adopt a posterior similarity matrix approach (Fritsch and Ickstadt, 2009). We record whether communities share an ancestor node at depth 1, 2 and 3 of the sampled dendrogram, and average over the posterior samples to compute a probability for each such pairing. The results are shown in heatmaps in Figure 3.4. One can observe clear block-diagonal structures at all depths, which get increasingly noisy as the depth increases.

To interpret the inferred clusters, we visualize the communities in Massachusetts on a map. We identified the largest diagonal block obtained from the posterior similarity matrix at depth 1, and colored the map by membership. We juxtapose the cluster membership map with another map colored by whether the median rent in

the community is above the threshold of \$550 in Figure 3.5. There is a nearly identical correspondence between the two maps, suggesting that the clustering faithfully captures the variation in the data. We also observe a visible concentration of such higher income and rent communities at eastern Massachusetts near the coast. Details for computing the posterior similarity matrices and the maps are provided in the supplementary materials.

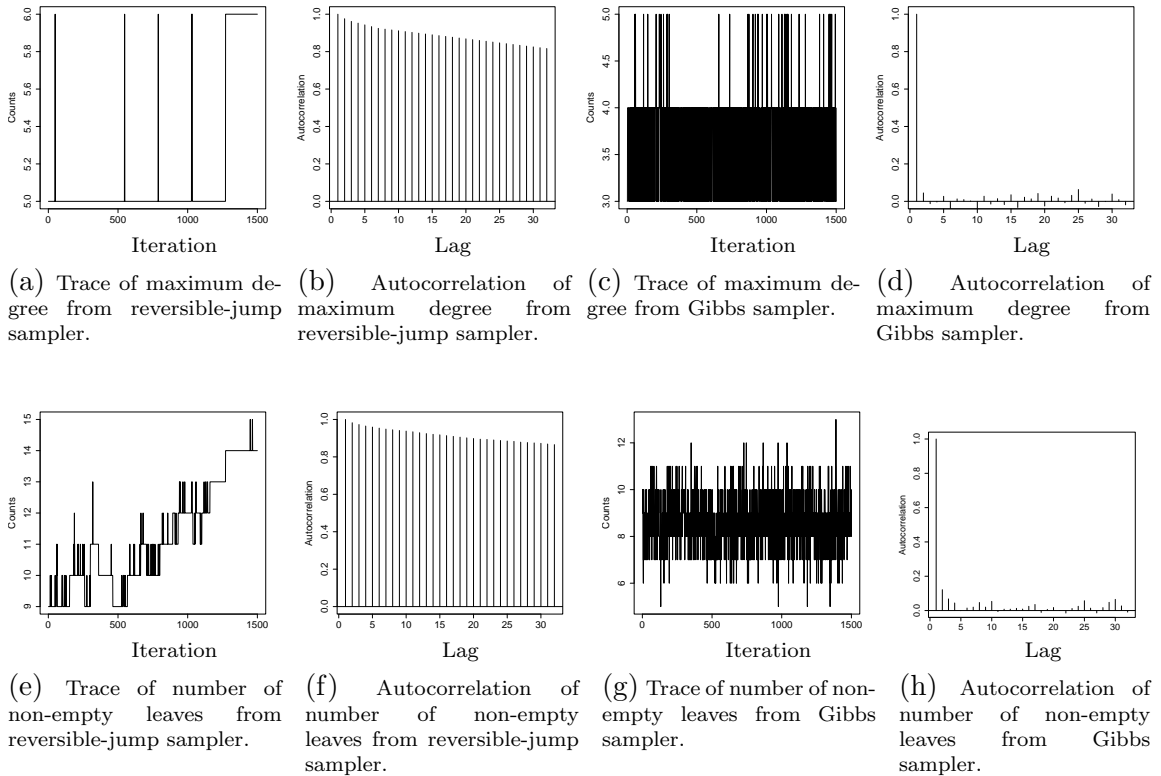


FIGURE 3.3: The Gibbs sampler for a spanning tree-augmented dendrogram model shows much faster mixing, compared to the reversible-jump Markov chain Monte Carlo sampler for a directly specified dendrogram model.

Parameters	Gibbs sampler	Reversible-jump sampler
Maximum degree	0.913	0.003
Maximum depth	0.29	0.007
Number of non-empty leaves	0.702	0.002

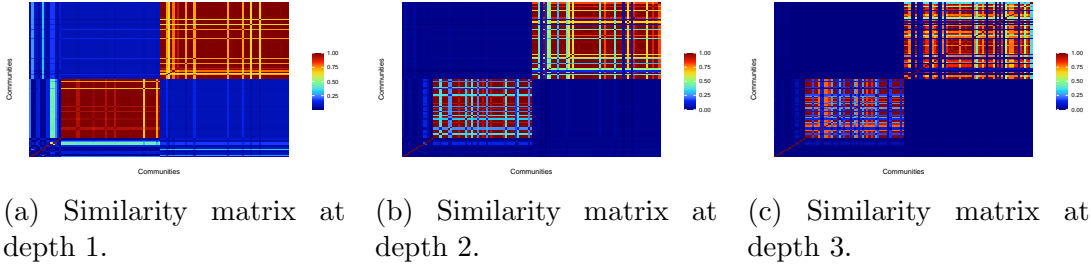


FIGURE 3.4: Posterior similarity matrices at different depths for crime and communities data .

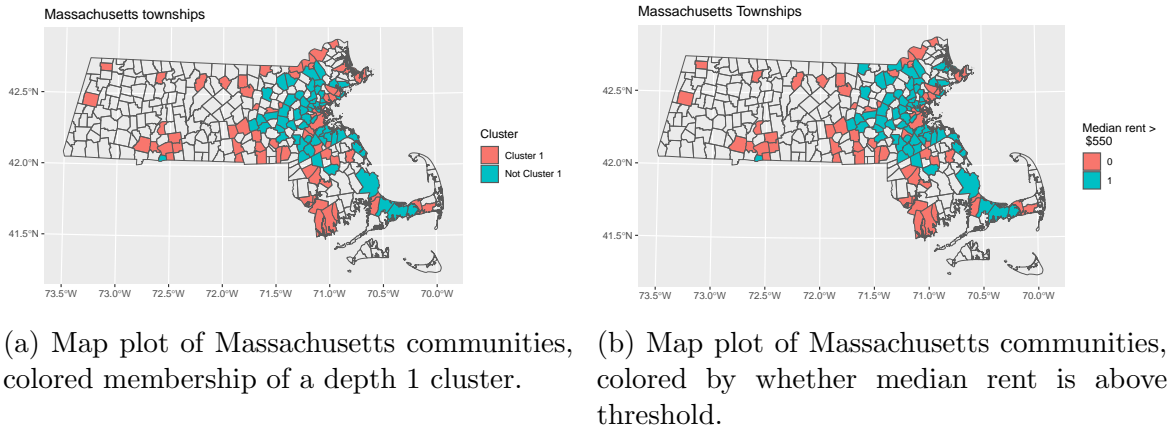


FIGURE 3.5: The communities in the largest cluster, red in panel (a), at depth 1 of the estimated dendrogram roughly match with those with median rent above \$550 in panel (b).

3.5 Discussion

Our work is motivated by improving computational efficiency of posterior inference for tree parameters. Expanding beyond the application to dendrograms, it is of interest to develop spanning tree-augmented models for more complex models, such as more elaborate discrete latent structure models (Zeng et al., 2023) or treed Gaussian process (Gramacy and Lee, 2008; Payne et al., 2024). For broader classes of tree models that may not admit a product-over-edges probability distribution, one

may modify our random walk cover algorithm to form a computationally efficient proposal-generating distribution. Some related Metropolis-Hastings algorithms for sampling graph partitions have been recently studied by Autry et al. (2023). For a tree with unknown node size or structural dependence on a latent arrival process, such as diffusion trees (Neal, 2003) or Bayesian phylogenetic trees (Huelsenbeck and Ronquist, 2001), it is interesting to explore extending the fast-forwarding algorithm to bypass wasteful random walks on an infinite graph, or under time-varying transition probabilities.

Appendix 1

Proofs

Proof for Theorem 2

Proof. Considering $(\hat{t}_{k+1} - \hat{t}_k)$ as the inter-arrival time, we want to show that

$$\mathbb{E}(\hat{t}_{k+1} - \hat{t}_k) \geq 1/\tilde{q}, \quad \tilde{q} = 1 + \min_{j \in U_k} \frac{\sum_{l \in U_k} w_{j,l}}{\sum_{l' \in \bar{U}_k} w_{j,l'}}.$$

Consider a sequence of independent Bernoulli events with success probabilities p_1, p_2, \dots , with all $p_i \leq \tilde{q}$, and denote the index on the first success in this sequence by T . If $\mathbb{E}T < \infty$ then $\mathbb{E}T \geq 1/\tilde{q}$. Since $T \geq 0$, we know $\mathbb{E}T = \sum_{t=0}^{\infty} \Pr(T > t) = \sum_{t=1}^{\infty} \prod_{i=1}^t (1 - p_i) \geq \sum_{t=1}^{\infty} \prod_{i=1}^t (1 - \tilde{q}) = 1/\tilde{q}$. Adding over $\hat{t} = \sum_{k=1}^m (\hat{t}_{k+1} - \hat{t}_k)$ with $\hat{t}_1 = 0$ yields the result. \square

Proof for Theorem 3

Proof. We first state the Cheeger inequality for circulation graphs [Chung (2005), Theorem 3]. For a directed and weighted graph with non-negative weight matrix W , having $\sum_{l=1}^m w_{j,l} = \sum_{l=1}^m w_{l,j}$ for all j , the second smallest eigenvalue $\lambda_2(\mathcal{L})$ satisfies

$$\sqrt{\lambda_2(\mathcal{L})} \geq \min_{U: 1 \leq |U| \leq m-1} \frac{\sum_{j \in U, l \in \bar{U}} w_{j,l}}{\min(\sum_{j \in U} d_j, \sum_{j \in \bar{U}} d_j)}.$$

Letting $U = \cap_{\tilde{t} \leq t} X_{\tilde{t}}$, $|U| \leq m - 1$, the probability of exiting U at time $t + 1$ is

$$\Pr(x_{t+1} \in \bar{U} \mid x_t \in U) = \sum_{j \in U} \left\{ \frac{\sum_{l \in \bar{U}} w_{j,l}}{d_j} S_t(j) \right\}.$$

Using $\tilde{e}_j = \sum_{l \in \bar{U}} w_{j,l}$, we obtain

$$\begin{aligned} \left[\sum_{j \in U} \left\{ \frac{\tilde{e}_j}{d_j} S_t(j) \right\} \right] \sum_{j \in U} d_j &= \sum_{j \in U} \left\{ \tilde{e}_j S_t(j) \frac{\sum_{j' \in U} d_{j'}}{d_j} \right\} \\ &\leq \sum_{j \in U} \tilde{e}_j \max_{j' \in U} \left\{ S_t(j) \frac{\sum_{j' \in U} d_{j'}}{d_j} \right\} \leq \left(\sum_{j \in U} \tilde{e}_j \right) \max_{j \in U \cup \bar{U}} \left\{ S_t(j) \frac{\sum_{j' \in U \cup \bar{U}} d_{j'}}{d_j} \right\}. \end{aligned}$$

After rearranging terms, we obtain:

$$\begin{aligned} \Pr(x_{t+1} \in \bar{U} \mid x_t \in U) &\leq \frac{\sum_{j \in U} \tilde{e}_j}{\sum_{j \in U} d_j} \max_{j \in U \cup \bar{U}} \left\{ S_t(j) \frac{\sum_{j'} d_{j'}}{d_j} \right\} \\ &\leq \frac{\sum_{j \in U} \tilde{e}_j}{\min(\sum_{j \in U} d_j, \sum_{j \in \bar{U}} d_j)} \max_{j \in U \cup \bar{U}} \left\{ S_t(j) \frac{\sum_{j'} d_{j'}}{d_j} \right\} \\ &\leq \frac{\sum_{j \in U} \tilde{e}_j}{\min(\sum_{j \in U} d_j, \sum_{j \in \bar{U}} d_j)} \max_{j \in U \cup \bar{U}} \left\{ S_t(j) \frac{\sum_{j'} d_{j'}}{d_j} \right\}. \end{aligned}$$

Using $\pi_j = d_j / \sum_{j'} d_{j'}$ and taking the minimum over both sides, we have

$$\min_{U: |U| \leq m-1} \Pr(x_{t+1} \in \bar{U} \mid x_t \in U) \leq \sqrt{\lambda_2(\mathcal{L})} \max_j S_t(j) / \pi_j.$$

Letting U_{k^*} be the solution to the above inequality, by a similar argument in the proof of Theorem 2,

$$\mathbb{E}(\hat{t}_{k^*+1} - \hat{t}_{k^*}^*) \geq \frac{1}{M \sqrt{\lambda_2(\mathcal{L})}}.$$

Adding the other $(m - 2)$ steps, each with $\hat{t}_{k'+1} - \hat{t}_{k'} \geq 1$, leads to the result. \square

Proof for Theorem 4

Proof. The Neumann series converges if the spectral radius $\lambda_1(P_{U,U}) < 1$ strictly. Since $P_{U,U}$ is a non-negative matrix, by Perron—Frobenius theorem, $\lambda_1(P_{U,U}) \leq \max_i \sum_j p_{i,j} \leq 1$.

Since $P_{U,U}$ is irreducible, by Perron—Frobenius theorem, there exists a unique vector $P_{U,U}^T \phi_* = \lambda_1(P_{U,U}) \phi_*$, with ϕ_* all positive and $1^T \phi_* = 1$. We follow Chapter 8 of Meyer (2023), and let Q be a non-negative matrix such that $P_{U,U}^T + Q$ has each column summable to 1, hence $1^T(P_{U,U}^T + Q)\phi \leq 1$ for any ϕ all positive and $1^T \phi = 1$. Since there exists at least $\eta_j > 0$, we know at least one $Q_{j,l} > 0$.

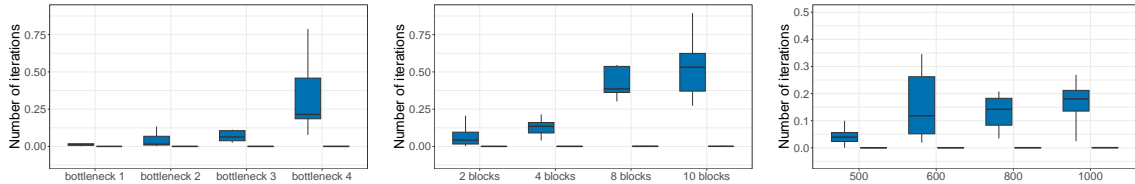
If $\lambda_1(P_{U,U}) = 1$, we would have

$$1^T(P_{U,U}^T + Q)\phi_* = 1^T \phi_* + 1^T Q \phi_* > 1,$$

which is a contradiction. Therefore, we know $\lambda_1(P_{U,U}) < 1$. □

Supplementary Materials

3.5.1 Comparisons of number of iterations



(a) Number of iterations to finish over different bottleneck rates $1/\sqrt{\lambda_2(\mathcal{L})}$. (b) Number of iterations to finish over different numbers of blocks (hence the numbers of bottlenecks). (c) Number of iterations to finish over different numbers of nodes.

FIGURE 3.6: Comparisons of number of iterations to finish between random walk cover (blue) and fast-forwarded cover (grey) algorithm.

3.5.2 Spanning tree-augmented dendrogram Gibbs sampler

The proposed Gibbs sampler iterates between the following sampling steps:

- Update the undirected version of the spanning tree T from the full conditional $\Pi(T \mid -)$, which is a spanning tree distribution of product-over-edges form, using the fast-forwarded cover algorithm. Form a directed tree \vec{T} from T by assigning 0 as the root, and having all edges pointed away from 0;
- For $i = 1, \dots, n$, update z_i from the full conditional $\Pi(z_i = k \mid -)$, a categorical distribution over $1, \dots, \tilde{m}$;
- Update $(\pi_1, \dots, \pi_{\tilde{m}})$ from the full conditional $\Pi(\pi_1, \dots, \pi_{\tilde{m}} \mid -)$, a Dirichlet distribution;
- Update the μ_k 's and Σ from the full conditional $\Pi(\mu_2, \dots, \mu_{\tilde{m}}, \Sigma \mid -)$, which takes the form of a normal-inverse-Wishart distribution.

Recall that the root parameter μ_1 is set at 0. Once tree samples have been collected, pruning steps outlined in section 3.4 are performed to obtain the reduced dendrograms.

3.5.3 Posterior similarity matrices and map

Posterior similarity matrices at depths 1, 2, 3 were generated for $n = 123$ Massachusetts communities available in the crimes and communities dataset. Each row/column in the matrix corresponds to a community. The ij th entry of the posterior similarity matrix at depth R corresponds to the fraction of Gibbs sampler iterations for which the two communities share an ancestor node at depth R . The posterior similarity matrix was computed on samples 3501 to 5000 with thinning factor 10. The depth 1 posterior similarity matrix is then reordered according to the output of a spectral biclustering algorithm to reveal block-diagonal structures. The same ordering is applied to all 3 posterior similarity matrices at depths 1, 2, 3. Maps data were obtained from the Massachusetts government website. The data from

the crimes and communities dataset do not include all Massachusetts towns/cities. Counties that are missing from the dataset are set to transparent on the map.

3.5.4 Details for reversible-jump Markov chain Monte Carlo

We use $V_{\vec{T}}$ as the set of nodes in the dendrogram, and $V_{\vec{T}}^C = (1, \dots, \tilde{m}) \setminus V_{\vec{T}}$ as the other nodes not on the dendrogram. Further, we use $V_{\vec{T}}^{\text{empty-leaf}}$ to denote the node set $\{k \in V_{\vec{T}} : k \text{ has no children, } n_k = 0\}$.

We use the same full conditional distributions as in the main text to update $(\pi_1, \dots, \pi_{\tilde{m}})$, and those μ_k with $k \in V_{\vec{T}}$. When updating z_i , we use the same multinomial distribution, except with $\phi(y_i; \mu_k, \Sigma)$ replaced by zero density if $k \in V_{\vec{T}}^C$.

For updating the dendrogram, we use the proposals of birth and death steps in each iteration. We draw a binary $u \sim \text{Bern}(p_0)$: if $u = 1$, we use the birth proposal unless $|V_{\vec{T}}| = n$; if $u = 0$, we use the death proposal unless $|V_{\vec{T}}| = 1$. Therefore, we have the birth probability $p_{\text{birth}}(\vec{T}) = p_0$ if $|V_{\vec{T}}| < n$, and 0 otherwise; and the death probability $p_{\text{death}}(\vec{T}) = 1 - p_0$ if $|V_{\vec{T}}| > 1$, and 0 otherwise. In our simulations, we set $p_0 = 0.1$.

- (Birth) Grow a new edge: draw j uniformly from $V_{\vec{T}}$, find the smallest index l from $V_{\vec{T}}^C$, draw $\mu_l \sim \mathcal{G}(\mu_l | \mu_j)$, and attach $(j \rightarrow l)$ to \vec{T} to form a proposed \vec{T}^* . Accept the \vec{T}^* with probability

$$\min \left\{ 1, \frac{0.01 p_{\text{death}}(\vec{T}^*) |V_{\vec{T}^*}^{\text{empty-leaf}}|^{-1}}{p_{\text{birth}}(\vec{T}) |V_{\vec{T}}|^{-1}} \right\}. \quad (3.12)$$

- (Death) Prune an empty leaf node: draw j uniformly from $V_{\vec{T}}^{\text{empty-leaf}}$, remove j from \vec{T} to form a proposed T^* . Accept the \vec{T}^* with probability

$$\min \left\{ 1, \frac{p_{\text{birth}}(\vec{T}^*) |V_{\vec{T}^*}|^{-1}}{0.01 p_{\text{death}}(\vec{T}) |V_{\vec{T}}^{\text{empty-leaf}}|^{-1}} \right\}, \quad (3.13)$$

provided that the current \vec{T} has $|V_{\vec{T}}^{\text{empty-leaf}}| > 0$. If $|V_{\vec{T}}^{\text{empty-leaf}}| = 0$ skip the step.

In the above, the Metropolis-Hastings acceptance ratio has a simple form because in the birth step, the product of the likelihood at \vec{T} and the proposal density for μ_l satisfy $L(y; (\mu_j)_{j \in V_{\vec{T}}}, \vec{T}, \cdot) \mathcal{G}(\mu_l | \mu_j) = L(y; (\mu_j)_{j \in V_{\vec{T}^*}}, \vec{T}^*, \cdot)$, hence canceling out the likelihood at the proposed state; and the transform $\{(\mu_j)_{j \in V_{\vec{T}}}, \mu_l\} \leftrightarrow (\mu_j)_{j \in V_{\vec{T}^*}}$ is one-to-one and has Jacobian determinant 1.

3.5.5 Modifications for Forest Models

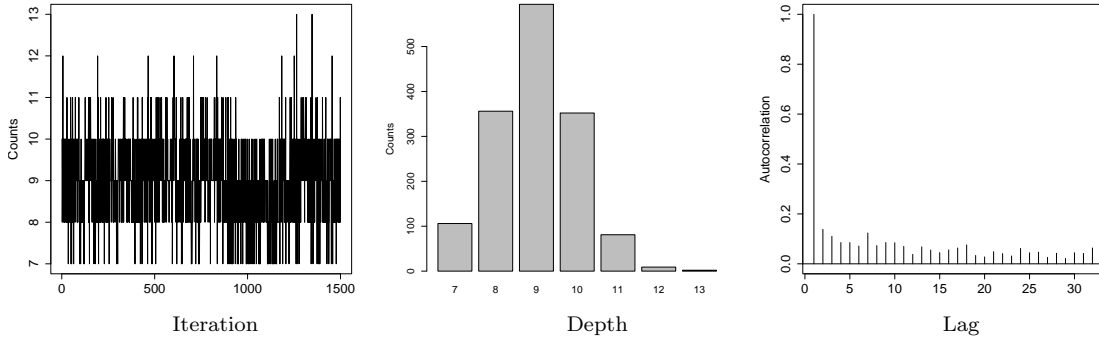


FIGURE 3.7: Trace, bar and autocorrelation plots for the depth of the tree samples from the Gibbs sampler.

We can view $\cup_{k=1}^K \vec{T}_k$ that arise from a forest model (3.2) as a spanning tree in the following manner. One can consider an auxiliary graph G with $(1, \dots, n+1)$ nodes, with $q_{j,l} = \mathcal{H}(\mu_l | \mu_j)$ for $j, l \leq n$ and $q_{n+1, r_k} = \mathcal{F}(y_{r_k})$ for $r_k \leq n$. Then, first specify \vec{T} as a large spanning tree rooted at $(n+1)$ and then cut all K edges adjacent to $(n+1)$ to obtain $\cup_{k=1}^K \vec{T}_k$. Our algorithm can be therefore be applied to the proposed class of forest models.

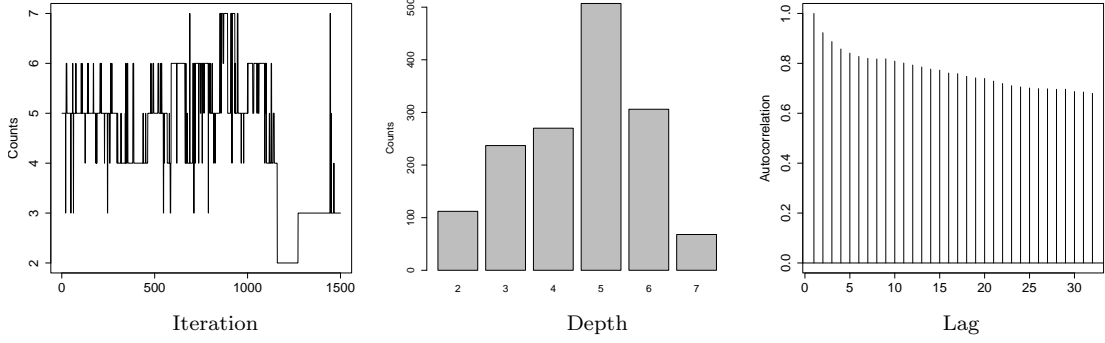


FIGURE 3.8: Trace, bar and autocorrelation plots for the depth of the tree samples from the reversible-jump Markov chain Monte Carlo sampler.

3.5.6 Normalized Laplacian

For any non-negative weight matrix W , define the normalized Laplacian (Chung, 1997) as

$$\mathcal{L} = I - \frac{1}{2} \{ \Phi^{1/2} (D^{-1} W) \Phi^{-1/2} + \Phi^{-1/2} (W^T D^{-1}) \Phi^{1/2} \},$$

where $\Phi = \text{diag}(\pi_j)_{j=1}^m$ and $D = \text{diag}(d_j)_{j=1}^m$. Since $\sum_{l=1}^m \pi_j(w_{j,l}/d_j) = \sum_{k=1}^m \pi_k(w_{k,j}/d_k)$ holds for $\pi_j = d_j / (\sum_{k=1}^m d_k)$, the normalized Laplacian reduces to

$$\mathcal{L} = I - D^{-1/2} \frac{W + W^T}{2} D^{-1/2}.$$

The above form further reduces to $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$ if $W = W^T$.

Graph Comparison via Laplacian Embeddings

4.1 Introduction

Graphs and networks are ubiquitous objects in statistics, machine learning and related fields. A routine task in network analysis is graph comparison, which involves quantifying structural similarities between graphs. Graph comparison is of interest to a wide range of practitioners, ranging from neuroscientists to network scientists. An appropriate similarity measure on graphs can aid the analysis and interpretation of network data. It can also be used in a variety of important downstream tasks, such as graph classification.

One important challenge in graph comparison is to develop simple, fast and theoretically sound methods for comparing graphs that are of different orders, i.e. graphs with different numbers of vertices. This is a natural task that arises in many application domains. For example, sociologists are often interested in comparing the structures of social communities in different populations with vastly different orders (Grindrod and Lee, 2016), while biologists might be interested in comparing protein-protein interaction networks between different biological species (Jeong et al., 2016).

These applications often arise in cases where the modeler would like to compare networks that share structural similarities.

Different methods for graph comparison capture different types of structural similarities between graphs. One particularly important type of graph structure is community structure. We are motivated to compare graphs of different orders based on the similarities in the community structures that they exhibit. Methods that directly operate in graph space, e.g. via methods that are based on edit distances etc, often lead to a combinatorial explosion in computational complexity, due to the discrete nature of the objects under comparison. One natural way to bypass this issue is to compare continuous representations of graphs instead. This motivates our approach, which compares graphs based on Euclidean embeddings that capture their connectivity structures.

One natural mathematical tool well known to capture community structures within a graph is the eigendecomposition of the graph Laplacian. The Laplacian eigenvectors specify a Euclidean embedding of the graph’s vertices, which forms the foundation of popular algorithms such as spectral clustering (Ng et al., 2001; Shi and Malik, 2000; Von Luxburg, 2007). Extensive work in spectral graph theory have shown that the eigenvectors of the graph Laplacian encode important connectivity information within a graph (Spielman, 2019). In particular, the embedding based on the bottom k Laplacian eigenvectors encode important information about k -way partitions and sparse cuts of the graph’s vertices (Lee et al., 2014; Louis et al., 2012). It is therefore natural to compare the community structures of different graphs using their Laplacian eigenvectors embeddings as a starting point.

In this work we propose the embedded Laplacian discrepancy (ELD) as an simple and fast approach that compares graphs of different orders based on the similarities in the graphs’ community structures. A general outline of ELD is described below. The first step of ELD, similar to other eigendecomposition based machine learning

methods such as principal components analysis (PCA), is to select a dimension k . Intuitively, k can be thought of as a level of “resolution” that is small enough to encourage efficiency and parsimony, but also large enough to capture most of the interesting community structures that the modeler would like to compare. We give practical suggestions on how to choose the k in a principled manner in subsection 3.3. Once k is selected, we compute the bottom k Laplacian eigenvectors of each graph. This allows us to represent each graph as a set of points in a k -dimensional Euclidean space. This reduces our graph comparison problem into comparing points clouds on a common Euclidean space, which can be done in a straightforward manner using efficient tools (Wasserstein distances) from optimal transport. This resulting measure of similarity is the ELD. We show that the ELD is a well-defined pseudometric that is invariant under graph isomorphism. This indicates that ELD compares graphical structure only and is agnostic to permutations of vertex labelings.

The main challenge in using eigenvector embeddings for comparing graphs is the potential ambiguity that could arise due to sign and basis ambiguities Lim et al. (2022); Eastment and Krzanowski (1982); Rustomov et al. (2007); Bro et al. (2008). Much recent research, particularly in the graph neural networks literature, has been devoted to tackling these ambiguities. Sign ambiguities arise due to the fact that given any eigenvector \mathbf{v} , $-\mathbf{v}$ is also a valid eigenvector. This issue is neatly resolved by a symmetrization trick, i.e. using both \mathbf{v} and $-\mathbf{v}$ to define the embedding.

Basis ambiguities arise when the graphs under consideration have repeated eigenvalues, since the eigenspace under consideration can have infinitely many choices of basis vectors. Prior work in the literature have attempted to tackle similar problems by performing intermediate steps of optimization (e.g. over rotation matrices) (Lai and Zhao, 2017). Here, we take a different approach based on perturbation. Whenever, there are repeated eigenvalues, we perturb the Laplacian matrix of the graph with a small, symmetric noise matrix which will split the repeated eigenvalues

almost surely. We then compute the ELD on the perturbed graphs as a proxy for the comparison between the original graphs. We utilize results from matrix perturbation theory that give stability guarantees of the ELD under perturbations/noise, under mild conditions that are easily checkable in practice. As such, we bypass the difficult basis-symmetry ambiguity problem by paying the small price of (controllable) perturbation and randomness. We demonstrate the practical appeal of the ELD via simulations and experiments.

Our Contributions Our general idea is to utilize meaningful continuous graph representations for graph comparison. The continuous graph representation is provided by Laplacian embeddings, and comparison is provided by defining a natural Wasserstein-based distance on the embedded points. The resulting embedded Laplacian discrepancy (ELD) is, to the best of our knowledge, the first method that combines a spectral/Laplacian embedding and a Wasserstein approach for comparing collections of graphs of different orders. Any eigendecomposition-based embedding face potential ambiguities. We propose a symmetrization step to eliminate sign ambiguities, and for graphs with basis ambiguities we propose a perturbation step to bypass them without resorting to costly intermediate optimization/averaging procedures common in the literature. We provide theoretical guarantees that our approach satisfies natural properties and is stable under noise and perturbations. We provide empirical evidence on both simulated data and real graph datasets that demonstrates the efficacy of this novel approach.

4.1.1 Related Work

There is an immense literature on graph comparison. Due to space limitations we only highlight those methods that are most relevant to our paper, and delegate the interested reader to the excellent reviews (Donnat and Holmes, 2018; Emmert-Streib

et al., 2016; Wills and Meyer, 2020; Tantardini et al., 2019; Soundarajan et al., 2014) for further information.

A predominant approach for graph comparison is via defining a pairwise graph distance. Distances that are based on the graph spectrum (Wilson and Zhu, 2008) or matrix representations (Wills and Meyer, 2020; Koutra et al., 2013) are natural tools that have been extensively studied. However, they are generally confined to comparing graphs of the same orders. An alternative direction is to hand-pick a list of graph features, for example those based on spectral or topological descriptors (Kaiser, 2011; Dehmer et al., 2010; Berlingerio et al., 2012), and then compare the graphs based on such features. Much work has also been devoted to the popular graph edit distances (Sanfeliu and Fu, 1983; Gao et al., 2010; Bunke, 1997). They have been shown to be NP-hard in general to compute (Zeng et al., 2009).

Another approach for comparing graphs of different orders is by considering graphs as metric spaces and then leveraging distances between isometry classes of metric spaces (Wills and Meyer, 2020), e.g. the Gromov-Hausdorff distance (Gromov, 2007, 1981; Edwards, 1975) and its relaxations such as the Gromov-Wasserstein distance (Chowdhury and Needham, 2021). The Gromov-Hausdorff distance involves optimizing over isometric embeddings of metric spaces and is known to be NP-hard to compute (Chazal et al., 2009; Mémoli, 2007; Oles et al., 2019). A variety of methods using Wasserstein distances (Villani, 2009) etc. have been proposed for comparing graphs (Xu et al., 2019; Maretic et al., 2019; Vayer et al., 2018). (Lai and Zhao, 2017) compared point clouds in 3D Euclidean space utilizing a sliced Wasserstein approach on the eigensystem of the Laplace-Beltrami operator, optimizing over intermediate rotations to resolve the basis ambiguity problem.

Beyond graph distances, another predominant approach for graph comparison is graph kernels, which define an inner product between graphs, thus allowing use of kernel-based machine learning methods. Vishwanathan et al. (2010) provide a

unified framework for many graph kernel methods, and Borgwardt et al. (2020); Kriege et al. (2020); Nikolentzos et al. (2019); Ghosh et al. (2018) provide recent surveys in this area. There are many variants with different emphasis, such as the random walk kernel, graphlet kernels, and shortest path kernels, to name a few. In the case of comparing graphs of different orders, the direct product kernel (Gärtner et al., 2003; Ketkar et al., 2009) is often used. Given two graphs with m and n vertices respectively, this method constructs a direct product graph of order $O(mn)$, on which computation of the kernel could cost $O((mn)^3)$.

Some existing methods that address similar problems to ours are outlined below. In the context of comparing graphs of different orders, one of the most recent methods is the Network Portrait Divergence (NPD) (Bagrow and Bollt, 2019), which uses information theoretic approaches to compare network portraits and is computationally fast. (Tsitsulin et al., 2018) proposes NetLSD, a descriptor that utilizes the trace of the Laplacian heat kernel matrix as a summary for graph comparison. We compare the ELD against the NPD and NetLSD in our experiments.

Generally, methods that are based on global graph structure, such as the graph spectrum, do not directly extend to the scenario where graphs of different orders are compared. Methods that are based on more local, combinatorial structures, such as those that are based on matching subgraphs, graphlets, edit operations etc, usually run into computational bottlenecks, since at the core of these approaches lies the difficult problem of (sub)graph isomorphism.

4.1.2 Organization

The paper is organized as follows. In section 2, we set up the notation and preliminary notions from spectral graph theory and optimal transport that are needed for this paper. In section 3, we formally introduce the ELD and the perturbed ELD. In section 4, we provide a theoretical characterization of the ELD. In section 5, we per-

form experiments on simulated and real graph data to illustrate the practical appeal of this approach. In section 6, we discuss the future directions and implications of this work.

4.2 Preliminaries

4.2.1 Notation and Setup

Let the triple $G = (V, E, w)$ denote a simple, undirected, weighted graph, where V is the set of vertices, $E \subset V \times V$ represents the edges of the graph, and $w : E \rightarrow \mathbb{R}_0^+$ is a weight function that assigns to each edge a non-negative real number. This framework subsumes the unweighted case, where we simply pick w to be a constant function on the edges. We use n to denote the number of vertices of G . Without loss of generality, we use the natural numbers to label the vertices, and denote by E_{ij} that edge that connects vertices i and j . We use $[n] := \{1, 2, \dots, n\}$ to denote the positive natural numbers up to n .

The graph's adjacency matrix \mathbf{A} is the matrix with entries $\mathbf{A}_{ij} := w_{ij} := w(E_{ij})$ if there is an edge between nodes i and j , and 0 otherwise. The degree matrix \mathbf{D} is the diagonal matrix with $\mathbf{D}_{ii} = \sum_{j=1}^n \mathbf{A}_{ij}$. The combinatorial Laplacian matrix \mathbf{L} is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. We use λ and \mathbf{v} to denote Laplacian eigenvalues and eigenvectors respectively. All matrices considered in the paper are symmetric, hence all corresponding eigenvalues are real. In particular, the combinatorial Laplacian matrix is positive-semidefinite, and its eigenvalues are presented in ascending order, so $0 \leq \lambda_i \leq \lambda_j$ for $i < j$. When appropriate, we use subscripts/superscripts on the above notations, such as \mathbf{L}_G and \mathbf{v}^G , to highlight dependency on the graph G .

In this paper we only consider graphs that are connected. This has essentially no loss of generality since we are mainly interested in studying/comparing community structures of graphs, and in the case of disconnected graphs, the community structures are apparent and each connected component can be analyzed separately.

4.2.2 Laplacian Embeddings

The core mathematical object in this paper is the combinatorial Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$. It is used widely in spectral graph theory and manifold learning for tasks such as clustering and dimensionality reduction. One useful way to understand the graph Laplacian is to look at its eigendecomposition $\mathbf{L} = \sum_{i=1}^n \lambda_i \mathbf{v}_i \mathbf{v}_i^T$. As outlined in the seminal paper Belkin and Niyogi (2003), this decomposition provides a natural embedding of a graph’s vertices in Euclidean space: pick the first k eigenvectors (corresponding to the k smallest non-zero eigenvalues) of \mathbf{L} (where $k \in \mathbb{N}^+$ is a hyperparameter), and use the entries of these eigenvectors as Euclidean coordinates for embedding the vertices. In other words, the i^{th} entry of the k^{th} eigenvector, $\mathbf{v}_k(i)$, provides the k^{th} coordinate of vertex i .

One simple interpretation of the Laplacian embedding is to see it as the solution to a natural optimization problem, due to Belkin and Niyogi (2003). Let $\mathbf{Y} := [\mathbf{y}_1 \cdots \mathbf{y}_k]$ be a $n \times k$ matrix, where \mathbf{Y}_{ij} represents the j^{th} Euclidean coordinate of the i^{th} vertex. If we want an embedding where vertices that are “more connected” with each other are embedded closely, a natural objective function to minimize is $\sum_{(i,j) \in E} w_{ij} \|\mathbf{y}(i) - \mathbf{y}(j)\|^2 = \text{Tr}(\mathbf{Y}^T \mathbf{L} \mathbf{Y})$, which is simply a weighted least squares minimization problem. We then have the following result:

Theorem 8 (Optimal Linear Embedding (Belkin and Niyogi, 2003)). *Given a connected graph $G = (V, E, w)$, the solution $\mathbf{Y} = [\mathbf{y}_1 \cdots \mathbf{y}_k]$ of the following optimization problem*

$$\arg \min_{\mathbf{Y}; \mathbf{Y}^T \mathbf{Y} = \mathbf{I}} \text{Tr}(\mathbf{Y}^T \mathbf{L} \mathbf{Y})$$

corresponds to the first k Laplacian eigenvectors.

Here the constraint $\mathbf{Y}^T \mathbf{Y} = \mathbf{I}$ enforces orthonormality. The proof’s arguments are based on standard spectral graph theory and the Rayleigh-Ritz variational char-

acterization of eigenvalues, and can be found in references such as Belkin and Niyogi (2003). Optionally, one can ignore the first eigenvector, which will always be constant.

The loss function above is set up in such a way that the Laplacian embedding respects graph connectivity, since vertices that are “more connected” with each other are given more weight and hence embedded more closely. To see this more concretely, consider the Laplacian spectral clustering algorithm Ng et al. (2001); Von Luxburg (2007); Shi and Malik (2000). The intuition is that since “more connected” vertices are closer together in the embedding space, one can just run a Euclidean clustering algorithm (e.g. K-means) in the embedding space to recover graph clusters. This can also be understood from a graph cut perspective, where one is trying to find a partition of the vertices of a graph such that the edges crossing between clusters have low weight and the edges within clusters have high weight. The (unnormalized) Laplacian embeddings can be viewed as the solution of a continuous relaxation of the a particular kind of graph cut known as NCut, which in its original discrete form is NP hard to solve (Von Luxburg, 2007; Wagner and Wagner, 1993).

Yet another way to show that Laplacian embeddings capture cluster structures is via results in spectral graph theory known as higher-order cheeger’s inequalities Lee et al. (2014); Louis et al. (2012). Such results imply that if the vertices of a graph admit k sparse cuts, then the resulting k partitions can be captured by the first k Laplacian eigenvectors of the graph.

In summary, if we want to compare graphs using continuous representations that reflect their connectivity patterns (in particular whether the graphs exhibit similar clustering structures), it is natural to consider the Laplacian embedding as a starting point. The embedding dimension k will serve as a hyperparameter that controls the number of clusters we are taking into consideration. We give suggestions on how to choose k later in the paper.

4.2.3 Wasserstein Distances

Wasserstein distances define metrics between probability measures. They are intimately connected to optimal transport (Villani, 2009), and have been used in graph settings increasingly often (Saad-Eldin et al., 2021; Kolouri et al., 2020).

Given two measure μ and ν on a k -dimensional Euclidean space, define the Wasserstein p -distance between them as:

$$\mathcal{W}_p(\mu, \nu) := \inf_{J \in \mathcal{J}(\mu, \nu)} \left(\int_{\mathbb{R}^k \times \mathbb{R}^k} d(x, y)^p dJ(x, y) \right)^{1/p}$$

where $\mathcal{J}(\mu, \nu)$ denotes the set of all couplings of μ and ν , and $p \geq 1$ specifies the order of the moment used.

When $p = 1$, then \mathcal{W}_1 is also known as the earth mover distance, and admits the following elegant equivalent representation:

$$\mathcal{W}_1(\mu, \nu) = \sup_{f \in \mathcal{F}} \left| \int f(x) d\mu(x) - \int f(y) d\nu(y) \right|$$

where $\mathcal{F} := \{f : \mathbb{R}^k \rightarrow \mathbb{R} \mid |f(x) - f(y)| \leq \|x - y\|_2 \forall x, y \in \mathbb{R}^k\}$ denotes the set of all 1-Lipschitz functions from \mathbb{R}^k to \mathbb{R} . Due to this useful representation, in the rest of the paper we set $p = 1$ for all Wasserstein distances.

The Wasserstein distance is often used to compare different sets of points in Euclidean space, by simply specifying μ and ν to be the empirical measures of the two sets of points that one wants to compare.

For cases where the dimension $k > 1$, computing the exact Wasserstein distance can involve time-consuming optimization that hampers computational performance. This computational problem disappears in the case when comparing 1-dimensional distributions. In particular, for $k = 1$, if μ and ν are both measures on the one-

dimensional Euclidean space, we have the following simple characterization:

$$\mathcal{W}_1(\mu, \nu) = \int_0^1 |F_\mu^{-1}(z) - F_\nu^{-1}(z)| dz$$

where F_μ and F_ν are the cumulative distribution functions of μ and ν respectively. This formulation allows for one-dimensional Wasserstein distances to be computed very efficiently. The above facts and formulations can be found in standard optimal transport references, such as (Kolouri et al., 2017; Villani, 2009).

Wasserstein distances are principled and natural mathematical tools for comparing probability distributions. This motivates us to find ways to leverage Wasserstein distances in defining the ELD, with a particular focus on exploiting the computational efficiency of 1-dimensional Wasserstein distances. In the next section, we utilize the Laplacian embedding and Wasserstein distances in a specific way that is simple and efficient to define the Embedded Laplacian Discrepancy.

4.3 The Embedded Laplacian Discrepancy (ELD)

We now formally introduce the ELD. We divide our definition and analysis into two parts: 1. comparing graphs with simple spectrums (no repeated eigenvalues) and 2. comparing graphs with non-simple spectrums (with repeated eigenvalues).

4.3.1 ELD for Graphs with Simple Spectrums

Let \mathcal{G} denote the space of all simple, undirected, weighted, connected graphs with finite numbers of vertices and simple spectrums. Our goal is to construct a mapping $\rho : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}_0^+$ that quantifies the structural similarity/dissimilarity between two graphs.

We will do so via the following general recipe. 1. Compute the first k Laplacian eigenvectors of the two graphs under comparison. 2. Use the entries of the k Laplacian eigenvectors as well as their negated counterpart (for symmetrization) to

represent the nodes of the two graphs as two point clouds in a common k -dimensional Euclidean space. 3. Treating the two points clouds as two empirical measures on the same space, leverage Wasserstein distances to compare them.

We setup some notation here. Given the r th eigenvector \mathbf{v}_r^G of a graph G with n nodes, we associate with it a one-dimensional empirical measure

$$\mu_r^G := \frac{1}{2n} \sum_{i=1}^n [\delta(\lambda_r^G \mathbf{v}_r^G(i)) + \delta(-\lambda_r^G \mathbf{v}_r^G(i))]$$

where δ is the Dirac measure and λ_r^G is the r th eigenvalue. Note that the negative of the eigenvector is also included to symmetrize the embedding. In section 4, we show that this leads to embeddings that are invariant to sign-flips of eigenvectors. Notationally, we will use both μ and ν below to denote the empirical measures associated with the eigenvectors of different graphs, with ν defined analogously to μ .

Definition 9 (Embedded Laplacian Discrepancy). *Consider $G_1 = (V_1, E_1, w_1) \in \mathcal{G}$ and $G_2 = (V_2, E_2, w_2) \in \mathcal{G}$, with orders $n_1 := |V_1|$ and $n_2 := |V_2|$ and Laplacians L_{G_1} and L_{G_2} respectively. Without loss of generality we assume $n_1 \leq n_2$. Given a dimension hyperparameter $k \leq n_1$, define the embedded Laplacian discrepancy as*

$$\rho(G_1, G_2) := \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$$

Several remarks are in order. First, Laplacian embeddings have traditionally been used in the context of analyzing vertices/clusters in a single graph. One of the unique features of the ELD is it utilizes Laplacian embeddings to project collections of graphs with different orders onto a common Euclidean space, as illustrated in Figure 4.1. Second, we weigh each point inside empirical measures by the corresponding Laplacian eigenvalues, which puts the entries of the Laplacian eigenvectors under their natural scalings.

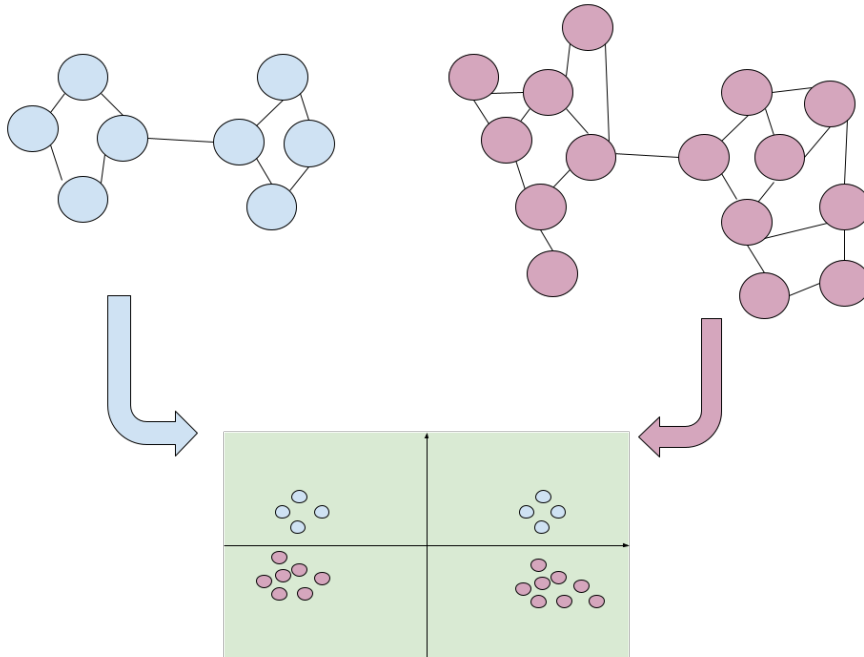


FIGURE 4.1: Depiction of Laplacian Embeddings

4.3.2 Design Choices

The Third, we defined the ELD via aggregating the one-dimensional Wasserstein distances along each eigenvector/canonical Euclidean axis. This can be naturally justified by noting the orthogonality of the Laplacian eigenvectors. Alternatively, from the spectral graph theory and graph signal processing literature (Spielman, 2019; Shuman et al., 2016; Ricaud et al., 2019), each Laplacian eigenvector can be thought of as encoding a certain “frequency” of the graph, and our approach then amounts to comparing the ordered frequency components of each graph one by one. This approach allows us to leverage the efficient computation of one-dimensional Wasserstein distances and directly leads to the ELD being isomorphism invariant (see section 4).

4.3.3 Computation of the ELD

We provide a procedure for implementing the ELD in Algorithm 1. In terms of computational complexity, the dominating step is the eigendecomposition, which is of order $O(n_2^3)$ under a naive theoretical analysis. In practice, most of the time $k \ll n$ and the eigendecomposition can be sped up via sparse or approximate numerical schemes. Similar pseudocode for the Perturbed ELD can be found in subsection A.5 of the appendix.

- 1: **Input:** Two graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ with $|V_1| = n_1$ and $|V_2| = n_2$. If perturbation step employed, then also noise matrix distribution P_N and noise variance ϵ
- 2: **Output:** A non-negative real number $\rho(G_1, G_2)$ that quantifies the structural similarity between G_1 and G_2
- 3: **Hyperparameters:** Positive integer $k \leq \min(n_1, n_2)$
- 4: **Algorithm:**
- 5: Compute the Laplacians \mathbf{L}_{G_1} and \mathbf{L}_{G_2} from the adjacency matrices \mathbf{A}_{G_1} and \mathbf{A}_{G_2}
- 6: Compute the eigendecomposition of \mathbf{L}_{G_1} and \mathbf{L}_{G_2} to obtain the Laplacian eigenvalues/eigenvectors of each graph $\{\lambda_i^{G_1}, \mathbf{v}_i^{G_1}\}$ and $\{\lambda_j^{G_2}, \mathbf{v}_j^{G_2}\}$
- 7: Construct the k empirical measures for each graph, i.e. construct $\mu_r^{G_1} := \frac{1}{2n_1} \sum_{i=1}^{n_1} [\delta(\lambda_r^{G_1} \mathbf{v}_r^{G_1}(i)) + \delta(-\lambda_r^{G_2} \mathbf{v}_r^{G_1}(i))]$ and $\nu_r^{G_2} := \frac{1}{2n_2} \sum_{i=1}^{n_2} [\delta(\lambda_r^{G_2} \mathbf{v}_r^{G_2}(i)) + \delta(-\lambda_r^{G_2} \mathbf{v}_r^{G_2}(i))]$ for each r in $[k]$
- 8: For each $r \in [k]$, compute the 1D Wasserstein distance $\mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$
- 9: Return the average $\rho(G_1, G_2) = \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$

Algorithm 2: Embedded Laplacian Discrepancy (no repeated eigenvalues)

Here, the δ notation indicates Dirac's delta measure.

4.3.4 Perturbed ELD for Graphs with Repeated Eigenvalues

The ELD framework introduced in the prior subsection is defined formally only on graphs in \mathcal{G} , i.e. graphs that have no repeated eigenvalues. When repeated eigenvalues are present in the graph(s) under comparison, naively applying the ELD approach above would not be satisfactory. This is due to the basis symmetries that arise from repeated eigenvalues, leading to an infinite number of potential choices

of basis vectors that span the eigenspace and ambiguity over which basis to use for the ELD. Specifically, if λ is an eigenvalue with multiplicity larger than 1, and \mathbf{v}_1 and \mathbf{v}_2 are two linearly independent eigenvectors with eigenvalue λ , then any linear combination of \mathbf{v}_1 and \mathbf{v}_2 will also be an eigenvector with eigenvalue λ . Prior work in the literature (Lai and Zhao, 2017) often uses intermediate optimization procedures over the orthogonal group to overcome the issue, which could potentially be computationally expensive.

We propose a simple alternative, called Perturbed ELD, that bypasses the need for any intermediate optimization step.

The Perturbed ELD is based on the following two insights. First, results from matrix perturbation theory show that the eigenvalues and eigenvectors (under mild, easily checkable conditions) of matrices are stable under small perturbations. Second, probability theory tells us adding continuous, symmetric noise to a symmetric matrix would split any repeated eigenvalues almost surely.

Utilizing these ideas, for each graph G with repeated Laplacian eigenvalues that is under comparison, we add a hollow, symmetric noise matrix to the adjacency matrix of G , where each upper-diagonal entry of the noise matrix is the absolute value of an independent Gaussian random variable with variance ϵ . Using the perturbed adjacency matrix, we obtain a new Laplacian matrix that has no repeated eigenvalues almost surely, and the ELD analysis can proceed as usual.

Definition 10 (Perturbed Embedded Laplacian Discrepancy). *Consider two connected graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$, with orders $n_1 := |V_1|$ and $n_2 := |V_2|$, adjacency matrices A_{G_1} and A_{G_2} , as well as Laplacians L_{G_1} and L_{G_2} respectively.*

Suppose both G_1 and G_2 have repeated Laplacian eigenvalues. Without loss of generality we assume $n_1 \leq n_2$. Given a dimension hyperparameter $k \leq n_1$, define

the embedded Laplacian discrepancy as

$$\rho(G_1, G_2) := \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$$

Perturbed ELD works because eigenvector embeddings are stable objects under small perturbations. We use perturbed, simple graphs as a proxy for the comparison between the original graphs with repeated eigenvalues. Precise stability guarantees for this approach, as well as discussions on the choice of noise distributions, will be given in section 4.

The mild spectral gap conditions that ground this approach are easily variable in practice. This, combined with the fact that we can pick the noise variance ϵ to be arbitrarily small, leads to a natural procedure that avoids any intermediate optimization step, retains the computational efficiency, and works excellently in practice (see section 5). The price that we pay is that the perturbed ELD leads to a random quantity, rather than a deterministic one. In practice, one can average over repeated samplings of perturbed ELD to obtain mean estimates to the desired level of statistical accuracy.

4.3.5 Computation of the Perturbed ELD

One could optionally average the results of Algorithm 2 under repeated realizations of \mathbf{N} .

4.3.6 Selection of hyperparameter k

Both the ELD and the perturbed ELD require the user to select a hyperparameter k , which represents the number of eigenvectors to consider for both graphs/the dimension of the embedding Euclidean space. A necessary requirement is that when comparing two graphs with n_1 and n_2 vertices respectively, k is less than or equal to $\min(n_1, n_2)$.

- 1: **Input:** Two graphs $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ with $|V_1| = n_1$ and $|V_2| = n_2$. If perturbation step employed, then also noise matrix distribution P_N and noise variance ϵ
- 2: **Output:** A non-negative real number $\rho(G_1, G_2)$ that quantifies the structural similarity between G_1 and G_2
- 3: **Hyperparameters:** Positive integer $k \leq \min(n_1, n_2)$
- 4: **Algorithm:**
- 5: Compute the Laplacians \mathbf{L}_{G_1} and \mathbf{L}_{G_2} from the adjacency matrices \mathbf{A}_{G_1} and \mathbf{A}_{G_2}
- 6: Compute the eigendecomposition of \mathbf{L}_{G_1} and \mathbf{L}_{G_2} to obtain the Laplacian eigenvalues/eigenvectors of each graph $\{\lambda_i^{G_1}, \mathbf{v}_i^{G_1}\}$ and $\{\lambda_j^{G_2}, \mathbf{v}_j^{G_2}\}$
- 7: **for** each graph G under consideration **do**
- 8: **if** \mathbf{L}_G has repeated eigenvalues **then**
- 9: Sample symmetric, hollow noise matrix \mathbf{N} with entries taken i.i.d. from specified noise matrix distribution P_N
- 10: Add \mathbf{N} to \mathbf{A}_G to obtain the perturbed adjacency matrix $\tilde{\mathbf{A}}_G$
- 11: Compute the perturbed Laplacian matrix $\tilde{\mathbf{L}}_G$ from $\tilde{\mathbf{A}}_G$
- 12: Compute the perturbed Laplacian eigenvalues and eigenvectors $\{\tilde{\lambda}_i^G, \tilde{\mathbf{v}}_i^G\}$ from $\tilde{\mathbf{L}}_G$
- 13: Use $\{\tilde{\lambda}_i^G, \tilde{\mathbf{v}}_i^G\}$ in all subsequent calculations
- 14: **end if**
- 15: **end for**
- 16: Construct the k empirical measures for each graph, i.e. construct $\mu_r^{G_1} := \frac{1}{2n_1} \sum_{i=1}^{n_1} [\delta(\lambda_r^{G_1} \mathbf{v}_r^{G_1}(i)) + \delta(-\lambda_r^{G_2} \mathbf{v}_r^{G_1}(i))]$ and $\nu_r^{G_2} := \frac{1}{2n_2} \sum_{i=1}^{n_2} [\delta(\lambda_r^{G_2} \mathbf{v}_r^{G_2}(i)) + \delta(-\lambda_r^{G_1} \mathbf{v}_r^{G_2}(i))]$ for each r in $[k]$
- 17: For each $r \in [k]$, compute the 1D Wasserstein distance $\mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$
- 18: Return the average $\rho = \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \nu_r^{G_2})$

Algorithm 3: Perturbed Embedded Laplacian Discrepancy (with repeated eigenvalues)

From the practice of spectral clustering, it is often noted that the existence of a large spectral gap between adjacent Laplacian eigenvalues λ_r and λ_{r+1} indicates the "appropriate" number of clusters in the graph is r . Recent theoretical work on higher-order Cheeger's inequality (for example Theorem 1.3 in (Lee et al., 2014)) provides theoretical support for this observation. Hence, for general applications of the ELD, we recommend users to first draw scree plots to inspect the number of meaningful clusters in both graphs, and then pick k accordingly. In general, a larger k provides a finer grain comparison between graphs. When using the perturbed

ELD, it is important to pick k so that there are no large spectral gaps in the first k eigenvalues of either graph in order for the eigenvectors to be stable. In the case where a repeated eigenvalue is splitted into b different eigenvalues after perturbation, k should be picked in such a way that all b resulting perturbed eigenvectors are included/excluded together. For downstream applications in supervised learning where the ELD is used as an intermediate step, k can be chosen via cross-validation. For subject-specific applications where the modeler has strong prior information, k can be chosen as the number of clusters that are of interest to the modeler.

4.4 Theoretical Properties

4.4.1 Theoretical Properties of ELD

We now demonstrate some mathematical properties of the ELD in this section. We first show that the ELD is invariant to graph isomorphisms, i.e. permutations of the vertices that preserve edge structure. Given any permutation $\sigma : [n] \rightarrow [n]$, we can associate to it a permutation matrix \mathbf{P}_σ , and we use G_σ to denote the graph obtained by applying σ on the vertices of G .

Theorem 11 (Invariance to Graph Isomorphisms). *The embedded Laplacian discrepancy mapping $\rho : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}^+$ is invariant to permutations. I.e. given two graphs G and H with n_1 and n_2 vertices respectively and simple spectrums, given permutations σ on $[n_1]$ and ω on $[n_2]$, we have:*

$$\rho_k(G, H) = \rho_k(G_\sigma, H_\omega)$$

for any integer $k \leq \min(n_1, n_2)$.

Proof. See section A.2 in the appendix. □

The invariance under isomorphism property ensures that the ELD is only capturing the desired graph connectivity structure, rather than non-structural charac-

teristics of the graphs that depend on arbitrary vertex labels.

We also show that the ELD is invariant to the any sign configuration of the eigenvectors.

Theorem 12 (Invariance to Sign Configurations). *The embedded Laplacian discrepancy mapping $\rho : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}^+$ is invariant to sign configurations of the eigenvectors.*

Proof. See section A.3 in the appendix. □

When we restrict ourselves to compare connected graphs that do not have repeated eigenvalues, the ELD provides a pseudometric on such graphs.

Theorem 13 (Pseudometric Property). *The embedded Laplacian discrepancy mapping $\rho : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}^+$ is a pseudo-metric.*

Proof. See section A.1 in the appendix. □

For our purposes of comparing graphs of different orders, the pseudometric property is appropriate and it is in general not feasible to upgrade the ELD to a metric. Since the goal of the ELD is to capture graph structure, isomorphic graphs will have a distance of 0 between each other under the ELD, thus preventing the realization of the identity of indiscernibles property of metrics. Even if we only consider defining a distance between equivalent classes of isomorphic graphs, the fact that the graphs could be of different orders renders the identity of indiscernibles property generally not possible since the smaller graph has less degrees of freedom than the larger graph.

4.4.2 Theoretical Properties of Perturbed ELD

Here we record some relevant theoretical properties of the perturbed ELD procedure. We first demonstrate that the perturbation procedure leads to a graph with distinct eigenvalues almost surely, thus resolving any ambiguity from basis symmetries.

Theorem 14 (Perturbation Splits Laplacian Eigenvalues Almost Surely). *Given a graph G with n vertices, adjacency matrix \mathbf{A} and Laplacian matrix \mathbf{L} . Define the perturbed adjacency matrix $\tilde{\mathbf{A}} := \mathbf{A} + \mathbf{N}$, where \mathbf{N} is a hollow, symmetric matrix where $\mathbf{N}_{ij} \sim |\text{Normal}(0, \epsilon)|$ for every $i < j$ and $\mathbf{N}_{ij} = \mathbf{N}_{ji}$. Define the perturbed diagonal matrix $\tilde{\mathbf{D}}$ as the matrix with $\tilde{\mathbf{D}}_{ii} = \sum_{j=1}^n \tilde{\mathbf{A}}_{ij}$ and 0 off the diagonals. Then the perturbed Laplacian matrix $\tilde{\mathbf{L}} := \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$ has no repeated eigenvalues almost surely.*

Proof. See section A.4 in the appendix. □

We remark here that the choice of the distribution for the entries of the noise matrix is chosen to be a (folded) Gaussian distribution for its simplicity and sharp concentration. The absolute value operation is employed here to avoid inducing any negative weights on the graph. In general, any non-negative, continuous distribution would suffice in splitting the eigenvalues. Other types of sub-Gaussian distributions (e.g. uniform distributions on non-negative intervals) would also retain similar sharp concentration properties.

Next, we record two well-known results in matrix perturbation theory that lends theoretical support to the stability of the perturbed ELD.

Theorem 15 (Weyl’s inequality (Chen et al., 2021; Horn and Johnson, 2012)). *Given any $n \times n$ symmetric matrix \mathbf{L} and any $n \times n$ symmetric perturbation matrix \mathbf{N} , we have that for any $1 \leq i \leq n$,*

$$|\lambda_i(\mathbf{L} + \mathbf{N}) - \lambda_i(\mathbf{L})| \leq \|\mathbf{N}\|$$

where $\|\cdot\|$ denotes the spectral norm.

Weyl’s inequality upper bounds the difference between the eigenvalues of a matrix before and after a perturbation. The Davis-Kahan inequality provides an analogous result for eigenvectors/eigenspaces. To state the theorem, we first set up an appropriate notion of distance between eigenspaces of matrices. Given any $n \times n$ symmetric

matrix L , let $\mathbf{V} := \sum_{r=1}^k \mathbf{v}_r \mathbf{v}_r^T$ denote the orthogonal projection operator projecting to the eigenspace spanned by \mathbf{L} 's eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_k$. Let \mathbf{N} be an $n \times n$ symmetric perturbation matrix. Let $\tilde{\mathbf{L}} := \mathbf{L} + \mathbf{N}$. Let $\tilde{\mathbf{V}} := \sum_{r=1}^k \tilde{\mathbf{v}}_r \tilde{\mathbf{v}}_r^T$ denote the orthogonal projection operator projecting to the eigenspace spanned by the eigenvectors of $\tilde{\mathbf{L}}$, $\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_k$. We are interested in upper bounding the quantity $\|\tilde{\mathbf{V}} - \mathbf{V}\|$, which characterizes the magnitude of the change in eigenspace of \mathbf{L} under the perturbation \mathbf{N} . Subject to the condition that the eigengap $\Delta := \lambda_{k+1} - \lambda_k > 0$ (where the $\lambda \geq 0$ here correspond to the ordered eigenvalues of \mathbf{L}), the following result is standard.

Theorem 16 (Davis-Kahan inequality (Davis and Kahan, 1970; Chen et al., 2021)).

If $\|\mathbf{N}\| \leq (1 - 1/\sqrt{2})\Delta$, we then have

$$\|\tilde{\mathbf{V}} - \mathbf{V}\| \leq \frac{\sqrt{2}\|\mathbf{N}\|}{\Delta}$$

where $\|\cdot\|$ denotes the spectral norm.

Both Weyl's inequality and the Davis-Kahan inequality are standard results whose proofs can be found in references such as (Chen et al., 2021). Note that the definition of the ELD/perturbed ELD depends only on the graphs under consideration only through their eigenvalues and eigenvectors. Weyl's inequality states that eigenvalues are stable under perturbation. The Davis-Kahan inequality states that the eigenspace spanned by the top k eigenvectors is stable under perturbation, provided that the corresponding spectral gap δ is not too small. In particular, both bounds depend on the norm of the noise matrix \mathbf{N} . Since the user has control over the noise parameter ϵ , in practice the norm of \mathbf{N} could be made small. Moreover, in practice the eigengap δ of each graph under comparison can be inspected via a simple scree plot. Hence, these results together provide guarantees of the stability of the perturbed ELD procedure in practice.

4.5 Experiments and Results

To gauge the efficacy of the ELD approach in practice, we implemented the ELD algorithm in Python 3.8 and tested it on both simulated and real datasets. All experiments were conducted on a Unix machine with 32GB of RAM and an Apple Silicon M1 Max chip. The software and data to reproduce all experiments in this paper will be made available after anonymous review. Our implementation detects any repeated eigenvalues in the graphs and automatically switches between the exact ELD and the perturbed ELD seamlessly.

We divide our simulations and experiments into several parts: 1. Comparing deterministic graphs via rings of cliques. 2. Comparing random graphs via stochastic blockmodels. 3. Comparing connectome graphs across different organisms. 4. Evaluating results in a downstream graph classification task where the ELD was used as a tool in a K-nearest neighbor classifier.

We consider two main competitors to the ELD for the multiscale graph comparison task: the Network Portrait Divergence (NPD) Bagrow and Bollt (2019) and the NetLSD Tsitsulin et al. (2018). These two competitors are selected based on their popularity, computational efficiency, and applicability to the multiscale case. Many other potential competitor methods for graph comparison do not extend to the case of comparing graphs with different orders. Other popular methods, such as the influential graph edit distance (GED) (Sanfeliu and Fu, 1983), are prohibitively slow. We discovered that the computational time for the GED (using an implementation from the popular NetworkX library (Hagberg et al., 2008)) exceeded our computational budget even for networks of moderate order, e.g. graphs with only several hundred vertices. In terms of computational speed in practice, the ELD exceeds the GED by orders of magnitude. We therefore only limit our comparisons to NPD and NetLSD.

Simulated Data: Rings of Cliques Since there is no ground truth notion of similarity between graphs, the most natural way to evaluate the ELD is by applying it to a collection of graphs that exhibit known structural similarities and observing whether the ELD captures these patterns.

Ring of cliques are a family of graphs that consist of cliques that are connected through single edges. They exhibit very clear community structures. In figures 1, 2, and 3, we plot heatmaps that represent dissimilarity/discrepancy matrices computed on a collection of ring of cliques graphs using the ELD ($k = 9$), NPD and NetLSD. We use $R(a, b)$ to denote a ring of cliques graph with a cliques, each clique consisting of b vertices.

In these figures we observe patterns in the ELD matrix that fit our intuitive understanding of graph structures. Clear block structures in the ELD heatmap reveal the pattern where ring of cliques graphs with similar number of clusters are closer together, whereas the ELD grows larger as the difference between the number of cliques in the graphs grows larger. This confirms our understanding that the ELD capture meaningful cluster/community structures in graphs.

On the other hand, the NPD and NetLSD heatmaps both exhibited no block structures. Both exhibit a highly checkered pattern instead, indicating a potentially different type of graph structure that the NPD and NetLSD are capturing. These results show that the ELD is able to capture meaningful cluster/community structures in graphs that its competitors cannot.

Simulated Data: Stochastic Blockmodels Stochastic blockmodels (SBM) are generative random graph models that naturally carries community structures.

In figure 4, 5 and 6, we plot heatmaps that represent dissimilarity/discrepancy matrices computed on a collection of stochastic blockmodel graphs using the ELD ($k = 9$), NPD and NetLSD. We use $SBM(a, b)$ to denote an SBM model with a blocks

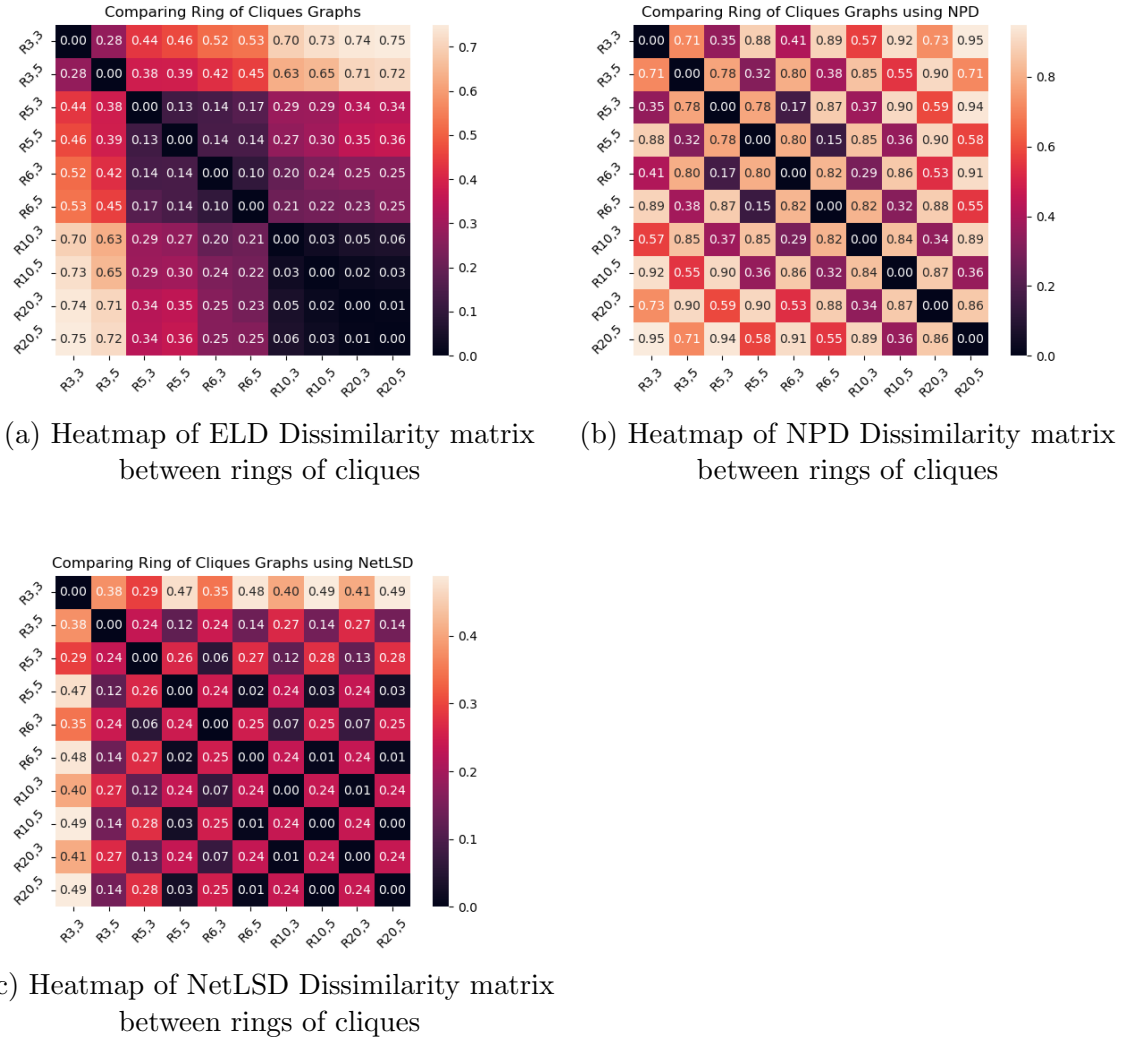


FIGURE 4.2: Comparison between ELD, NPD and NetLSD on rings of cliques

and $b//a$ nodes in each block, with between-block connection probability 0.99 and within-block connection probability 0.01. Each entry in the heat map represents the mean value of the discrepancy/dissimilarity measure computed over 20 independent samples of the given SBM models.

We observe that, similar to the ring of cliques results, the ELD heatmap exhibited clear and gradual block structures that indicated the capturing of community structures, whereas the NPD and NetLSD heatmaps continue to exhibit a checkered

pattern that do not capture community structures.

Computational Time Comparison In figure ?? we compared the computational cost of the ELD, NPD and NetLSD. We used these methods to compare ring of cliques graphs, varying the number of vertices of the graphs compared. To ensure robustness, we repeated these comparisons 20 times and reported the total computer run time (in seconds) of each method. We can observe from figure 7 that all three methods are computationally fast. As the number of vertices increases, all three methods showed a similar growth pattern in run time, with NetLSD being the fastest, ELD being in the middle, and NPD being the slowest. Remark that for graphs of smaller orders (below 40 vertices) the ELD performs just as fast as NetLSD.

Real Data: Connectome graphs For real network data, we use a connectome dataset hosted at NeuroData (Vogelstein et al., 2018).

The connectome data we used consisted of graphs characterizing different regions of the brain from different organisms, including two graphs from mice (Bock et al., 2011), three from rats (Bota et al., 2012) and two from the nematode *P. pacificus* (Bumbarger et al., 2013). The connectome graphs are directed, and we take their undirected underlying graphs for comparison in figure ??.

We observe that the ELD captures relevant graph structural information, since there are clear block structures that distinguishes between the connectome graphs of different organisms in figure 8.

Graph Classification: MUTAG dataset MUTAG-2 Debnath et al. (1991) is a dataset on the mutagenicity on *Salmonella typhimurium* of collection of chemicals know as nitroaromatics. It is a benchmark dataset for graph classificaiton. The dataset consists of 188 chemical graphs, each with a binary label y taking the values 1 and

Table 4.1: Training and Testing Accuracies on MUTAG

	ELD	NPD	NetLSD
Training Accuracy	0.872	0.840	0.829
Testing Accuracy	0.800	0.800	0.733

-1 indicating whether the compound is mutagenic. We obtained the dataset and the dataloader code using the PSCN python package Niepert et al. (2016).

We use a K -nearest neighbor (KNN) classifier (with $K = 3$) to perform binary classification on the MUTAG dataset, using the ELD, NPD and NetLSD as the dissimilarity/distance measures of the KNN algorithm. We report both training and testing error in table 1. For training error, we trained and evaluated on the entire dataset. For testing error, we trained on 158 data points and tested on 30 hold-out data points.

Results indicate that these graph distances/discrepancies are capturing meaningful structural information in these chemical graphs that can be leveraged for predictive tasks. In particular, ELD has the best performance overall in terms of both training and testing accuracy.

4.6 Discussion and Future Directions

The motivation of the ELD is to offer a principled and practical way for comparing graphs that might have similar structures but different orders, which often arises in applications. Note that comparing graphs of the same order is simply a special case of the ELD framework we presented, so all of our theoretical guarantees and computer programs carry over. The framework that we presented mainly deals with (weighted) simple, undirected graphs, but ELD could be potentially extended to many settings. We discuss two of these directions below.

Directed Graphs ELD could potentially be generalized to comparing directed graphs. One way is to apply the ELD approach on the underlying undirected graph of the digraph, but this discards orientation information. Another approach is to take the directed version of the Laplacian (Chung, 2005) and develop analogous embeddings.

Labelled Graphs ELD could also be extended to comparing vertex-labelled graphs, where there are known correspondences between vertices. The most natural way to do this is to retain the vertex correspondences after performing the Laplacian embedding, and compute a distance in Euclidean space between the matching vertices.

Computational efficiency is one of the advantages of adopting the ELD approach. The most computationally intensive step in ELD is eigendecomposition, which we have implemented using standard numerical libraries and methods. It is possible to achieve substantial speedups in practice by using sparse numerical techniques or a myriad of other approximation/optimization schemes. For applications with repeated calculations on the same graphs (e.g. calculating a distance matrix), the embedded representation can be stored and retrieved for more efficient computation.

One major design choice made in the definition of the ELD is by using the combinatorial Laplacian rather than the normalized Laplacian $\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The ELD definition and implementation described above could easily be extended to the normalized Laplacian case, and virtually all of our theoretical results will carry over. While the combinatorial Laplacian approach naturally takes scale information of the graphs into account, the normalized Laplacian does not. Depending on the context of the application, users might find either forms of the Laplacian to be more appropriate.

In the experiments, we observed that the NPD/NetLSD dissimilarity matrices did not show block structure, but instead shows an interesting checkerboard structure for comparing rings-of-cliques/SBMs. One hypothesis that this suggests those distances

might be capturing local motifs/structural information rather than the number of cliques/communities globally.

The focus of this paper is mainly to introduce the ELD as a novel discrepancy between graphs of different scales. There are many further downstream machine learning/statistics tasks that one could potentially perform with the ELD, and we delegate these investigations to future research.

4.7 Appendix

4.7.1 Proof of pseudometric property of ELD

Proof. To show that the ELD $\rho_k(\cdot, \cdot)$ is a pseudometric, we have to check four properties. Our arguments hold for all admissible k as specified in the definition of the ELD in section 3.

1. Nonnegativity: $\rho_k(\cdot, \cdot) \geq 0$. Note that $\mathcal{W}_1(\cdot, \cdot) \geq 0$ since it is a metric. Since $\rho_k(G, H)$ is simply an average of $\mathcal{W}_1(\cdot, \cdot)$ terms, ρ_k will also be non-negative.
2. Symmetry: $\rho_k(G, H) = \rho_k(H, G)$ for any graphs G, H . Note that since $\mathcal{W}_1(\cdot, \cdot)$ is a metric, it is also symmetric. Since $\rho_k(G, H)$ is just the average of $\mathcal{W}_1(\cdot, \cdot)$ terms, with the first argument of \mathcal{W}_1 dependent only on G and the second argument of \mathcal{W}_1 dependent only on H , conclude that $\rho_k(G, H)$ is also symmetric.
3. Identity of indiscernibles: $\rho_k(G, G) = 0$. (since this is a pseudometric, we allow for the possibility that there exist $G \neq H$ where $\rho_k(G, H) = 0$). To show this, simply note that \mathcal{W}_1 is a metric, hence $\mathcal{W}_1(a, a) = 0$ for any $a = a$. Since $\rho_k(G, G)$ is just the average of $\mathcal{W}_1(\cdot, \cdot)$ terms where the two arguments in each \mathcal{W}_1 term will be identical, conclude that each of the \mathcal{W}_1 terms will be 0 and hence $\rho_k(G, G) = 0$.
4. Triangle inequality: $\rho_k(G_1, G_2) + \rho_k(G_2, G_3) \geq \rho_k(G_1, G_3)$ for graphs G_1, G_2, G_3 .

To show the triangle inequality, again leverage the fact that \mathcal{W}_1 is a metric to obtain the inequality $\mathcal{W}_1(a, b) + \mathcal{W}_1(b, c) \geq \mathcal{W}_1(a, c)$ for any a, b, c .

Get:

$$\begin{aligned}
& \rho(G_1, G_2) + \rho(G_2, G_3) \\
&= \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \mu_r^{G_2}) + \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_2}, \mu_r^{G_3}) \\
&= \frac{1}{k} \sum_{r=1}^k (\mathcal{W}_1(\mu_r^{G_1}, \mu_r^{G_2}) + \mathcal{W}_1(\mu_r^{G_2}, \mu_r^{G_3})) \geq \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^{G_1}, \mu_r^{G_3}) \\
&= \rho(G_1, G_3)
\end{aligned}$$

□

4.7.2 Proof of invariance under graph isomorphism

Proof. Recall that the ELD between two graphs G and H with n_1 and n_2 vertices respectively is given by:

$$\rho_k(G, H) = \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^G, \nu_r^H)$$

where $\mu_r^G := \frac{1}{2n_1} \sum_{i=1}^{n_1} [\delta(\lambda_r^G \mathbf{v}_r^G(i)) + \delta(-\lambda_r^G \mathbf{v}_r^G(i))]$ and $\nu_r^H := \frac{1}{2n_2} \sum_{j=1}^{n_2} [\delta(\lambda_r^H \mathbf{v}_r^H(j)) + \delta(-\lambda_r^H \mathbf{v}_r^H(j))]$.

The ELD by definition computes a distance between the empirical measures associated with the Laplacian embeddings of two respective graphs. Hence, to show that the ELD is unaffected by graph isomorphisms, it suffices for us to show that the empirical measure associated with the Laplacian embedding of a graph is invariant under graph isomorphisms/permutaitons.

Given a graph G , a permutation on its vertices σ , and the corresponding permutation matrix \mathbf{P}_σ , it is well known fact (Spielman, 2019) that

$$L_{G_\sigma} = \mathbf{P}_\sigma \mathbf{L}_G \mathbf{P}_\sigma^T$$

It then follows directly by associativity that:

$$\mathbf{L}_{G_\sigma} \mathbf{v} = \lambda \mathbf{v} \Leftrightarrow (\mathbf{P}_\sigma \mathbf{L}_G \mathbf{P}_\sigma^T)(\mathbf{P}_\sigma \mathbf{v}) = \lambda(\mathbf{P}_\sigma \mathbf{v}) \quad (4.1)$$

This shows that the Laplacian eigenvectors of the permuted graph is simply the eigenvectors of the original graph permuted the same way. This also shows that eigenvalues are unchanged by graph isomorphisms/permutations.

Based on the above, note that the effect of the permutation on the empirical measure of a graph is just changing the order of addition of the Dirac measures, and since we are only considering finite sums, the order of addition does not affect the sum.

This shows that the empirical measure associated with the Laplacian embedding of a graph is invariant under graph isomorphisms, which proves the desired claim. \square

4.7.3 Proof of invariance under sign configurations of eigenvectors

Proof. Recall that the ELD between two graphs G and H with n_1 and n_2 vertices respectively is given by:

$$\rho_k(G, H) = \frac{1}{k} \sum_{r=1}^k \mathcal{W}_1(\mu_r^G, \nu_r^H)$$

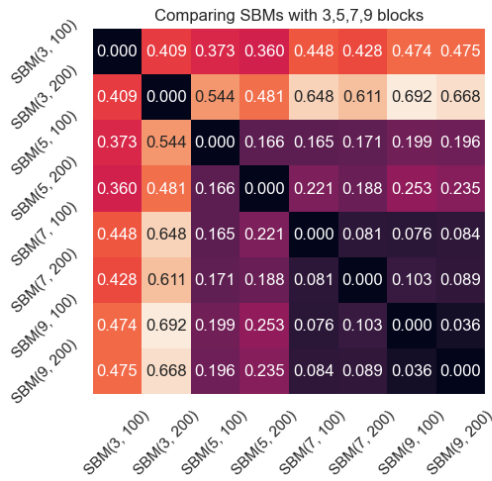
where $\mu_r^G := \frac{1}{2n_1} \sum_{i=1}^{n_1} [\delta(\lambda_r^G \mathbf{v}_r^G(i)) + \delta(-\lambda_r^G \mathbf{v}_r^G(i))]$ and $\nu_r^H := \frac{1}{2n_2} \sum_{j=1}^{n_2} [\delta(\lambda_r^H \mathbf{v}_r^H(j)) + \delta(-\lambda_r^H \mathbf{v}_r^H(j))]$.

It suffices that we show the empirical measure of a graph is invariant under sign configurations of eigenvectors. Pick an arbitrary eigenvector \mathbf{v}_r from either graph for

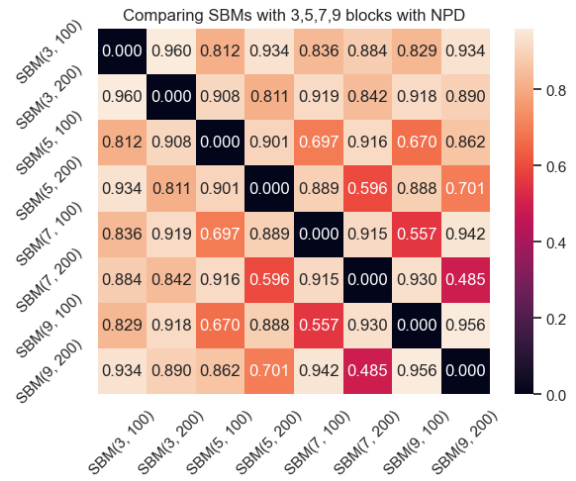
any $1 \leq r \leq k$. Now simply note that if we flip the sign of the chosen eigenvector from \mathbf{v}_r to $-\mathbf{v}_r$, the corresponding empirical measure, as defined, remains unchanged due to the symmetry in the term $\delta(\lambda_r \mathbf{v}_r(j)) + \delta(-\lambda_r \mathbf{v}_r(j))$ for every entry j . \square

4.7.4 Proof of perturbation almost surely splitting eigenvalues

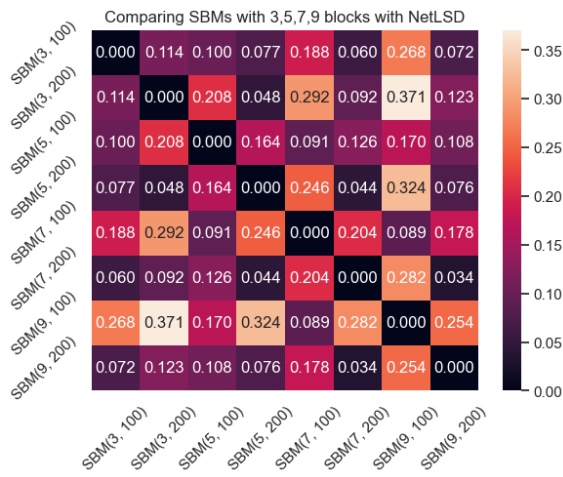
Proof. Note that each entry of the perturbed Laplacian matrix $\tilde{\mathbf{L}}$ is a random variable following a continuous distribution. It is a well-known mathematical fact that the eigenvalues of a matrix are continuous functions of the entries of the matrix. This is due to the characterization of eigenvalues via the characteristic polynomial (for further reference, see Zedek (1965)). Combining the two facts above, conclude that each eigenvalue of $\tilde{\mathbf{L}}$ is continuously distributed. Conclude that the event where two eigenvalues are exactly equal to each other has measure 0 under any continuous distribution. \square



(a) Heatmap of ELD Dissimilarity matrix between SBM graphs

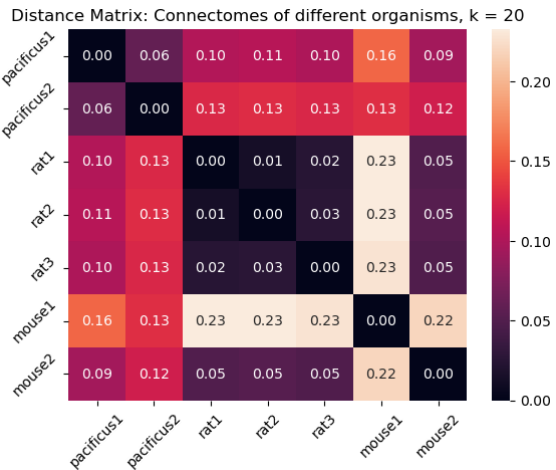
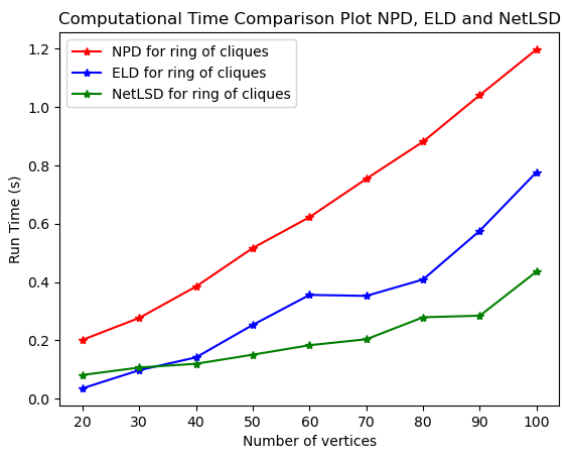


(b) Heatmap of NPD Dissimilarity matrix between SBM graphs



(c) Heatmap of NetLSD Dissimilarity matrix between SBM graphs

FIGURE 4.3: Comparison between ELD, NPD and NetLSD on SBM graphs



(a) Computational time comparison (b) ELD comparisons between connectomes

FIGURE 4.4: ELD running time analysis and connectomes heatmap

On the Statistical Capacity of Deep Generative Models

5.1 Introduction

Deep generative models, such as variational autoencoders and generative adversarial networks, are routinely used for a wide variety of applications. Despite these models' empirical success in generating apparent samples from complex, high-dimensional distributions, their statistical properties are not well understood. A common assumption amongst practitioners is that with enough training data and a sufficiently large neural network, such deep generative models can sample from any continuous target distribution. We debunk this assumption by showing that broad classes of deep generative models, including variational autoencoders and generative adversarial networks, are not universal generators. We show that under typical Gaussian and uniform distributions for their latent variables, these models can only sample from certain distributions with light tails. We give quantitative characterizations on the expressivity of the learned distribution using functional inequalities. Our results caution against the cavalier use of such models for inferential tasks under heavy tailed

data.

5.2 Introduction

Sampling is foundational to statistics. Markov chain Monte Carlo methods remain the gold standard for a wide variety of sampling tasks in Bayesian modeling and beyond. However, modern applications, ranging from physical simulations to computer vision, necessitate sampling from increasingly high dimensional and complex target distributions. Under these scenarios with complicated geometry, Markov chain Monte Carlo methods often require extensive tuning and can exhibit poor mixing.

A powerful alternative sampling paradigm utilizing deep neural networks has emerged in recent years. The core idea is to generate samples $f(z)$ by transforming simple latent variables z with a sufficiently expressive learned function f . As long as the law of $f(z)$ approximates the target distribution, $f(z)$ can be treated as a sample. This turns the sampling problem into a function approximation problem. When training samples are available, deep neural networks are natural candidates for modeling f . Given the immense flexibility of neural networks, it is common to set z as Gaussian or uniform, and let the learned neural network \hat{f} capture the complexity of the target distribution. This approach encapsulates popular models such as generative adversarial networks and variational auto-encoders.

These deep generative models have achieved state-of-the-art results in a wide variety of applications, ranging from medical imaging to drug design. The synthetic data generated can also find use in many downstream machine learning and engineering tasks, such as anomaly detection.

Owing to the status of neural networks as universal function approximators, there is a folklore assumption in the literature that, given enough training data, and a sufficiently large neural network, such transformation-based deep generative models can sample from any continuous target distributions. Our work here debunks this as-

sumption. We show that, for a broad range of commonly adopted distributions for z , such as the Gaussian or the uniform, the law of $f(z)$ will have light, sub-Gaussian-like tails. Leveraging geometric tools, we show that when z follows a log-concave distributions, $f(z)$ exhibits sub-exponential-like tails. This shows that even though neural networks are universal function approximators, deep generative models like generative adversarial networks and variational autoencoders are not universal generators. We complement the results above with more quantitative characterizations on upper bounds of the entropy and variances of $f(z)$ using functional inequalities, and generalize to the manifold setting.

Since the mean and median of $f(z)$ remains completely flexible, it is unsurprising that the typical sample from such deep generative models empirically resembles samples from the target distribution. However, due to the light tailedness of $f(z)$, when the target distribution is heavy tailed, inferential tasks from such samples will tend to underestimate uncertainty. Given the prevalence of the use of these deep generative models for tasks such as anomaly detection, these observations suggest conservatism to the practitioner when interpreting the samples from such models.

5.3 Deep Generative Models

5.3.1 Deep neural networks

Deep neural networks play a crucial role in deep generative modeling. We consider fully-connected feed-forward neural networks. Let L be the number of layers, also known as the depth, of the neural network. Given input $z \in \mathbb{R}^d$, define the feed-forward neural network via the composition $f(z) = h_L(h_{L-1}(\cdots(h_1(z))\cdots))$, where $h_l(z) = \sigma_l(W_l z)$, σ_l is a non-linear activation function operating elementwise on the l th layer and W_l is the weight matrix corresponding to the l th layer. Note that the dimensions of W_l , as well as the choice of activation functions, can vary layer to layer. Let $\dim(W_l)$ denote the number of columns or the number of rows of the matrix W_l ,

whichever is larger. We call $\max_{l=1}^L \dim(W_l)$ the width of the neural network. For additional details, see the excellent review by ?.

We use d to denote the latent variable dimension and p to denote the output dimension. Let $f(z) \in \mathbb{R}^p$. The function f is Lipschitz if $\sup_{x \neq y} \|f(x) - f(y)\| / \|x - y\| \leq \mathcal{L}$ for some finite constant $\mathcal{L} > 0$, where $\|\cdot\|$ denotes the Euclidean norm. We write S for the collection of activation functions that are Lipschitz. S includes the rectified linear unit function $\sigma_{ReLU}(x) = \max(0, x)$, the logistic function $\sigma_{logistic}(x) = \{1 + \exp(-x)\}^{-1}$, the hyperbolic tangent function $\tanh(x)$, and many more.

We introduce the notion of finite feed-forward neural networks below:

Definition 17 (Finite feed-forward neural networks). *We say a feed-forward neural network f is finite if 1. the depth L of the neural network is finite 2. the width of the neural network is finite 3. all entries in the matrices $W_{l=1}^L$ are finite 4. all activation functions $\sigma_{l=1}^L$ are members of S . We denote the class of finite feed-forward neural networks as \mathcal{F} .*

In practice, due to the finite memory of computers, all neural networks encountered are going to have finite depth, width and weight entries. Virtually all of the commonly used activation functions are Lipschitz, since activation functions that are more irregular can often lead to the exploding gradients and vanishing gradients problem during neural network training.

Proposition 18. *Finite feed-forward neural networks are Lipschitz.*

Proof. A finite neural network consists of finitely many compositions of linear transformations by W_l and non-linear activations by σ_l . Since the Lipschitz property is preserved under finite composition, and since $\sigma_l \in S$, it suffices to show that linear transformations by W_l are Lipschitz. The Lipschitz constant of the linear function $g(x) = W_l x$ is given by the spectral norm $\|W_l\|_2$. The spectral norm $\|W_l\|_2$ is up-

per bounded by the Frobenius norm $\|W_l\|_F$, which is finite since matrix entries and dimensions are finite. \square

Note that many commonly encountered neural network operations, such as dropout, pooling and batch normalization have finite Lipschitz constants. Our results throughout could be straightforwardly extended to neural networks that incorporate a finite number of these operations.

A broad class of foundational deep generative models uses neural networks to transform latent variables for sampling. Generative adversarial networks, as well as variational autoencoders, fall within this regime. To generate a sample from these models after training, one first samples the latent variables $z \in R^d$ from a prescribed prior distribution, such as a standard multivariate Gaussian, and then apply the trained neural network f to obtain the sample $f(z)$. While we focus on generative adversarial networks and variational autoencoders here, the framework developed applies to other models that transform latent variables by a Lipschitz function, such as certain classes of Lipschitz diffusion models.

5.4 Isoperimetry and Concentration of Deep Generative Models

We recall the definition of a sub-Gaussian random variable.

Definition 19 (Sub-Gaussian). *We say a random variable x with mean $E(x) = \mu$ is σ^2 -sub-Gaussian if the following concentration inequality is satisfied.*

$$P(|X - \mu| \geq t) \leq 2 \exp(-t^2/\sigma^2) \tag{5.1}$$

where $C, c > 0$ are constants.

Note that to ease notation, throughout the paper we follow the convention where we use $C, c > 0$ to denote positive constants whose values can change line to line. Equivalently, x is sub-Gaussian if the Orlicz norm $\|x\|_{\psi_2} = \{\inf_{c>0} E(\exp((x -$

$E(x)^2/c^2) \leq 2\}$ is finite. A random vector z is sub-Gaussian if $\sup_{v \in S^{d-1}} \|v^T z\|_{\psi_2} < \infty$, where S^{d-1} is the unit sphere.

A sub-Gaussian random variable have tails that decay at least as fast as that of a Gaussian random variable (i.e. in a squared exponential fashion), and hence is sharply concentrated about its mean/median. Important examples of sub-Gaussian random variables include Gaussian random variables, bounded random variables, uniform random variables on compact sets, and any finite mixtures thereof.

Let \mathcal{G} denote the set of all sub-Gaussian distributions on \mathbb{R}^p . Note that the dimensions p and d here are treated as constants in the generative modeling context, and the trained neural network f is treated as a fixed learned function after training. We now move to our first concentration inequality regarding deep generative models.

Theorem 20 (Concentration with Gaussian Latent Variables). *Let z be a Gaussian random vector with mean μ and variance Σ . Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^p$ be any finite neural network function with Lipschitz constant \mathcal{L} . Then for any unit vector $v \in \mathbb{R}^p$, we have $P(v^T |f(z) - E(f(z))| > t) \leq Cp \exp(-cdt^2)$ for some $C, c > 0$. In other words, the centered random variable $f(z) - E(f(z))$ is sub-Gaussian.*

Note that the the mean $E(f(z))$ can be substituted by the median $\text{med}(f(z))$ with only changes to the constant C and c . Throughout this work, the finite neural network function f is a learned function that is fixed at generation time, so its Lipschitz constant, as well as its input and output dimensions, can be treated as constants in our context. The variance of the latent variable distribution is also a fixed constant at generation time. The free parameter in the concentration inequality is t . As such, we sometimes include the Lipschitz factor and the variance factor into the constants $c, C > 0$, which eases notation and does not impact the interpretation.

In other words, $\mathcal{F}(z) \subset \mathcal{G}$. The above concentration form of the Gaussian isoperimetric inequality is relevant since standard normal priors are often used in deep gen-

erative contexts. This result demonstrates that deep generative models that push-forward Gaussian measures can only model light tailed, sub-Gaussian distributions.

We now generalize the above result to the sub-Gaussian prior context.

Theorem 21 (Sub-Gaussian case). *Let z be a random vector in \mathbb{R}^d with mean $E(z)$ such that $z - E(z)$ is a σ^2 -sub-Gaussian distributed random vector. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^p$ be any finite neural network function with Lipschitz constant \mathcal{L} , then for any unit vector $v \in \mathbb{R}^p$, we have $P(v^T |f(z) - E(f(z))| > t) \leq Cp \exp(-cdt^2)$ for some constants $C, c > 0$.*

In other words, the centered random vector $f(z) - E(f(z))$ is sub-Gaussian.

Proof. We first show the case for $f : \mathbb{R} \rightarrow \mathbb{R}$. Let y be an independent copy of x . Consider the Orlicz norm of $f(x)$, which corresponds to the smallest $q > 0$ such that $E \exp\{f(x) - E(f(x))\}^2/q^2$ is upper bounded by 2. This is equivalent to $E \exp\{f(x) - E(f(y))\}^2/q^2 = E \exp\{f(x) - f(y)\}^2/q^2 \leq E \exp\{L^2(x - y)^2/q^2\}$. By the inequality $2x^2 + 2y^2 \geq (x - y)^2$, the above is $\leq E \exp\{2L^2x^2 + 2L^2y^2/q^2\} = E \exp\{4L^2x^2/q^2\}$. Picking $q = 2L\sigma$ makes the above Orlicz norm upper bounded by 2, which completes the proof. Generalization of the above to the case from \mathbb{R}^d to \mathbb{R}^p follows again from simple triangle inequality arguments. \square

The above theorem generalizes our results in a similar flavor to the case of sub-Gaussian priors on the latent variables. This includes the important cases of finite mixtures of Gaussians and sub-Gaussians, uniform distributions, and any bounded random variables.

One can further generalize the above results to the log-concave case, using tools from high-dimensional geometry.

Theorem 22 (Log-concave concentration). *Let $z \in \mathbb{R}^d$ be centered random vector with isotropic log-concave probability density. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a finite neural*

network function that is \mathcal{L} -Lipschitz. Then

$$\Pr(|f(z) - E(f(z))| > t) \leq C \exp(-c\Psi_z t/\mathcal{L})$$

for some constants $C, c > 0$, where Ψ_z is the Cheeger's constant of the density of z .

The above theorem is due to ?. We adopt the form used in ?. Note that due to recent progress on the Kannan-Lovasz-Simonovitz conjecture, the Cheeger's constant involved in the above upper bound can be replaced by a polylogarithmic function of the input dimension. Generalizing the output dimension to p is straightforward.

5.4.1 Manifold Setting

The manifold hypothesis suggests that often times the latent variables under consideration lie on a low dimensional manifold. The above concentration of measure perspective allows us to generalize the above results to the manifold setting via the Gromov-Levy inequality. The following result is due to ?, and we use the versions in ??.

Theorem 23 (Gromov-Levy). *Let (M, g) be a compact, connected d -dimensional Riemannian manifold ($d \geq 2$) with positive Ricci curvature bounded below everywhere. Let ν be its normalized volume element. Let $z \sim \nu$. Let $f : M \rightarrow \mathbb{R}$ be a \mathcal{L} -Lipschitz finite neural network function. Then there exists $m \in \mathbb{R}$ such that*

$$\nu(|f(z) - m| > t) \leq C \exp(-(d-1)\mathcal{L}t^2)$$

for some constant $C > 0$.

5.5 Quantification of capacity

The above results utilizing concentration of measure and isoperimetric inequalities give a qualitative characterization of the tail behavior of the learned distribution $f(z)$.

For the case of Gaussian latent variables, we can also give more quantitative bounds on the variance and entropy of the learned distribution via functional inequalities.

Theorem 24 (Gaussian Poincare Inequality). *Let $f : \mathbb{R}^p \rightarrow \mathbb{R}$ be a finite neural network function with Lipschitz constant \mathcal{L} . Let z be a standard Gaussian random vector. Then $\text{Var}(f(z)) \leq \mathcal{L}^2$.*

The result above gives an upper bound on the variance of the individual entries of the samples output from the deep generative models.

We could also obtain

Theorem 25 (Gaussian log-Sobolev Inequality). *Let $f : \mathbb{R}^p \rightarrow \mathbb{R}^d$ be a finite neural network function with Lipschitz constant \mathcal{L} . Let z be a standard Gaussian random vector. Then $\text{Ent}(f(z)^2) \leq 2d\mathcal{L}^2$, where $\text{Ent}(\cdot)$ denotes the joint differential entropy.*

The above is a simple application of the classical Gaussian log-Sobolev inequality, with the chain rule for differential entropy. The above results show that if the Lipschitz constant of the neural network is too small, the richness the output samples will be constrained. Note that both the Gaussian log-Sobolev inequality and the Gaussian Poincare inequality are known math results that we are applying to the generative modeling context.

5.6 Discussion

There is a large body of literature focused on specifying parametric forms of generative models. Classical approaches to clustering, such as the Gaussian mixture model, as well as classifiers such as the Gaussian naïve Bayes classifier, are both examples. Most of the traditional literature on shallow models can only handle data that lies in low dimensions. It is generally impossible to specify shallow parametric models that capture the true hierarchical and highly interdependent nature that generates data as complex as images.

There has been enormous interest in the generative modeling of high dimensional, complex data, such as images and audio recordings. Recently, with the advent of powerful function approximators such as deep neural networks, deep generative models have been increasingly popular. In application areas such as computer vision where large corpuses of unlabelled datasets are available, these deep generative models have found enormous success in modeling high dimensionanl datasets. This is reflected by the ability of such generative models to generate synthetic samples that visually and qualitatively resembles that of the training data, e.g. in the case of generating images of human faces.

Our results here show that despite the apparent empirical success of such deep generative models, we have to employ caution against the cavalier interpretation of generated samples from such models.

We gave a theoretical characterization of the statistical limitations of certain broad classes of foundational deep generative models. Whether this constraint on variational autoencoders and generative adversarial networks can be broken via alternative priors/architectures remains an interesting direction of investigation. Notably, other types of deep generative models, such as non-Lipschitz diffusion models and energy based models, might be free from the constraints outlined in this paper.

6

Conclusion

This thesis developed novel methodology and theory in regularization and sampling using tools from spectral graph theory, isoperimetry and concentration of measure.

These encouraging initial results indicate the promise of this research direction. Some future directions are outlined as follows.

On the Fiedler regularization front, how to use similar techniques for more general statistical models, such as graphical models, is an open problem. Only the Fiedler value has been investigated in this work, but more general spectral objects, such as multiple Laplacian eigenvalues, or the spectrum of matrices other than the Laplacian, can be considered. On the Bayesian side, the development of priors that correspond to Fiedler regularization is an interesting direction.

For graph comparison, utilizing continuous representations other than the Laplacian matrix, such as the adjacency matrix, is a natural direction to investigate. Generalizing this comparison method to avenues where spectral method less naturally applies, such as in the case of directed graphs and hypergraphs, is a challenging direction.

For spanning tree samplers, an interesting theoretical question is whether one can

perform similar marginalization arguments to accelerate other popular random walk based spanning tree samplers, such as the celebrated Wilson’s algorithm. Another question is other graphical structures beyond spanning trees can be sampled using similar approaches.

For the work on the limitations of deep generative models, one wonders whether we could come up with novel priors/architectures that circumvent the Lipschitz/light-tailed limitations of existing variational autoencoder/generative adversarial network models. It would also be interesting to investigate whether similar limitations can be derived for other classes of deep generative models beyond the Lipschitz pushforward type of models that we investigate in this work.

Bibliography

- Aldous, D. J. (1990), ‘The random walk construction of uniform spanning trees and uniform labelled trees,’ *SIAM Journal on Discrete Mathematics*, 3, 450–465.
- Arbel, J., Beraha, M., and Bystrova, D. (2021), ‘Bayesian block-diagonal graphical models via the Fiedler prior,’ in *SFdS-52 Journées de Statistique de la Société Française de Statistique*, pp. 1–6.
- Autry, E., Carter, D., Herschlag, G. J., Hunter, Z., and Mattingly, J. C. (2023), ‘Metropolized forest recombination for Monte Carlo sampling of graph partitions,’ *SIAM Journal on Applied Mathematics*, 83, 1366–1391.
- Bagrow, J. P. and Bollt, E. M. (2019), ‘An information-theoretic, all-scales approach to comparing networks,’ *Applied Network Science*, 4, 1–15.
- Bartlett, P. L. and Mendelson, S. (2002), ‘Rademacher and Gaussian complexities: Risk bounds and structural results,’ *Journal of Machine Learning Research*, 3, 463–482.
- Batson, J., Spielman, D. A., and Srivastava, N. (2012), ‘Twice-ramanujan sparsifiers,’ *SIAM Journal on Computing*, 41, 1704–1721.
- Belkin, M. and Niyogi, P. (2003), ‘Laplacian eigenmaps for dimensionality reduction and data representation,’ *Neural computation*, 15, 1373–1396.
- Berlingerio, M., Koutra, D., Eliassi-Rad, T., and Faloutsos, C. (2012), ‘Netsimile: A scalable approach to size-independent network similarity,’ *arXiv preprint arXiv:1209.2684*.
- Bock, D. D., Lee, W.-C. A., Kerlin, A. M., Andermann, M. L., Hood, G., Wetzel, A. W., Yurgenson, S., Soucy, E. R., Kim, H. S., and Reid, R. C. (2011), ‘Network anatomy and in vivo physiology of visual cortical neurons,’ *Nature*, 471, 177–182.
- Borgwardt, K., Ghisu, E., Llinares-López, F., O’Bray, L., and Rieck, B. (2020), ‘Graph Kernels: State-of-the-Art and Future Challenges,’ *arXiv preprint arXiv:2011.03854*.

- Bota, M., Dong, H.-W., and Swanson, L. W. (2012), ‘Combining collation and annotation efforts toward completion of the rat and mouse connectomes in BAMS,’ *Frontiers in neuroinformatics*, 6, 2.
- Bousquet, O., Boucheron, S., and Lugosi, G. (2003), ‘Introduction to statistical learning theory,’ in *Summer School on Machine Learning*, pp. 169–207, Springer.
- Boyd, S. (2006), ‘Convex optimization of graph Laplacian eigenvalues,’ in *Proceedings of the International Congress of Mathematicians*, vol. 3, pp. 1311–1319.
- Bro, R., Acar, E., and Kolda, T. G. (2008), ‘Resolving the sign ambiguity in the singular value decomposition,’ *Journal of Chemometrics: A Journal of the Chemometrics Society*, 22, 135–140.
- Broder, A. Z. (1989), ‘Generating random spanning trees,’ in *30th Annual Symposium on Foundations of Computer Science*, pp. 442–447.
- Broder, A. Z. and Karlin, A. R. (1989), ‘Bounds on the cover time,’ *Journal of Theoretical Probability*, 2, 101–120.
- Bumbarger, D., Riebesell, M., Sommer, R., et al. (2013), ‘System-wide rewiring underlies behavioral differences in predatory and bacterial-feeding nematodes,’ *Cell*, 152, 109–119.
- Bunke, H. (1997), ‘On a relation between graph edit distance and maximum common subgraph,’ *Pattern Recognition Letters*, 18, 689–694.
- Candes, E. J., Wakin, M. B., and Boyd, S. P. (2008), ‘Enhancing sparsity by reweighted ℓ_1 minimization,’ *Journal of Fourier analysis and applications*, 14, 877–905.
- Carmichael, I. (2021), ‘The folded concave Laplacian spectral penalty learns block diagonal sparsity patterns with the strong oracle property,’ arXiv preprint arXiv:2107.03494.
- Castillo, I. and Ročková, V. (2021), ‘Uncertainty quantification for Bayesian CART,’ *The Annals of Statistics*, 49, 3482–3509.
- Chazal, F., Cohen-Steiner, D., Guibas, L. J., Mémoli, F., and Oudot, S. Y. (2009), ‘Gromov-Hausdorff stable signatures for shapes using persistence,’ in *Computer Graphics Forum*, vol. 28, pp. 1393–1403, Wiley Online Library.
- Chen, Y., Chi, Y., Fan, J., Ma, C., et al. (2021), ‘Spectral methods for data science: A statistical perspective,’ *Foundations and Trends® in Machine Learning*, 14, 566–806.

- Chipman, H. A., George, E. I., and McCulloch, R. E. (1998), ‘Bayesian CART model search,’ *Journal of the American Statistical Association*, 93, 935–948.
- Chipman, H. A., George, E. I., and McCulloch, R. E. (2010), ‘BART: Bayesian additive regression trees,’ *The Annals of Applied Statistics*, 4, 266 – 298.
- Chowdhury, S. and Needham, T. (2021), ‘Generalized spectral clustering via Gromov-Wasserstein learning,’ in *International Conference on Artificial Intelligence and Statistics*, pp. 712–720, PMLR.
- Chung, F. (2005), ‘Laplacians and the Cheeger inequality for directed graphs,’ *Annals of Combinatorics*, 9, 1–19.
- Chung, F. R. (1997), *Spectral graph theory*, no. 92, American Mathematical Soc.
- Davis, C. and Kahan, W. M. (1970), ‘The rotation of eigenvectors by a perturbation. III,’ *SIAM Journal on Numerical Analysis*, 7, 1–46.
- Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991), ‘Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity,’ *Journal of medicinal chemistry*, 34, 786–797.
- Dehmer, M. M., Barbarini, N. N., Varmuza, K. K., and Graber, A. A. (2010), ‘Novel topological descriptors for analyzing biological networks,’ *BMC structural biology*, 10, 1–17.
- Denison, D. G., Mallick, B. K., and Smith, A. F. (1998), ‘A Bayesian CART algorithm,’ *Biometrika*, 85, 363–377.
- Donath, W. E. and Hoffman, A. J. (1972), ‘Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices,’ *IBM Technical Disclosure Bulletin*, 15, 938–944.
- Donnat, C. and Holmes, S. (2018), ‘Tracking network dynamics: A survey of distances and similarity metrics,’ *arXiv preprint arXiv:1801.07351*.
- Duan, L. L. and Dunson, D. B. (2023), ‘Bayesian spanning tree: estimating the backbone of the dependence graph,’ *The Journal of Machine Learning Research*, 24, 1–44.
- Duan, L. L. and Roy, A. (2023), ‘Spectral clustering, Bayesian spanning forest, and forest process,’ *Journal of the American Statistical Association*, pp. 1–24.
- Duchi, J., Hazan, E., and Singer, Y. (2011), ‘Adaptive subgradient methods for online learning and stochastic optimization,’ *Journal of machine learning research*, 12, 2121–2159.

- Eastment, H. and Krzanowski, W. (1982), ‘Cross-validators choice of the number of components from a principal component analysis,’ *Technometrics*, 24, 73–77.
- Edwards, D. A. (1975), ‘The structure of superspace,’ in *Studies in topology*, pp. 121–133, Elsevier.
- Elidan, G. and Gould, S. (2008), ‘Learning bounded treewidth Bayesian networks,’ *Advances in Neural Information Processing Systems*, 21.
- Emmert-Streib, F., Dehmer, M., and Shi, Y. (2016), ‘Fifty years of graph matching, network alignment and network comparison,’ *Information sciences*, 346, 180–197.
- Fan, J. and Li, R. (2001), ‘Variable selection via nonconcave penalized likelihood and its oracle properties,’ *Journal of the American statistical Association*, 96, 1348–1360.
- Fan, J., Ma, C., and Zhong, Y. (2019), ‘A selective overview of deep learning,’ arXiv preprint arXiv:1904.05526.
- Fredes, L. and Marckert, J.-F. (2023), ‘A combinatorial proof of Aldous–Broder theorem for general Markov chains,’ *Random Structures & Algorithms*, 62, 430–449.
- Fritsch, A. and Ickstadt, K. (2009), ‘Improved criteria for clustering based on the posterior similarity matrix,’ *Bayesian Analysis*, 4, 367 – 391.
- Gao, X., Xiao, B., Tao, D., and Li, X. (2010), ‘A survey of graph edit distance,’ *Pattern Analysis and applications*, 13, 113–129.
- Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1974), ‘Some simplified NP-complete problems,’ in *Proceedings of the sixth annual ACM symposium on Theory of computing*, pp. 47–63.
- Gärtner, T., Flach, P., and Wrobel, S. (2003), ‘On graph kernels: Hardness results and efficient alternatives,’ in *Learning theory and kernel machines*, pp. 129–143, Springer.
- Ghosh, S., Das, N., Gonçalves, T., Quaresma, P., and Kundu, M. (2018), ‘The journey of graph kernels through two decades,’ *Computer Science Review*, 27, 88–111.
- Godsil, C. and Royle, G. F. (2013), *Algebraic graph theory*, vol. 207, Springer Science & Business Media.
- Golowich, N., Rakhlin, A., and Shamir, O. (2018), ‘Size-independent sample complexity of neural networks,’ in *Conference On Learning Theory*, pp. 297–299, PMLR.

- Gower, J. C. and Ross, G. J. (1969), ‘Minimum spanning trees and single linkage cluster analysis,’ *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18, 54–64.
- Gramacy, R. B. and Lee, H. K. H. (2008), ‘Bayesian treed Gaussian process models with an application to computer modeling,’ *Journal of the American Statistical Association*, 103, 1119–1130.
- Graves, A. (2013), ‘Generating sequences with recurrent neural networks,’ arXiv preprint arXiv:1308.0850.
- Grindrod, P. and Lee, T. (2016), ‘Comparison of social structures within cities of very different sizes,’ *Royal Society open science*, 3, 150526.
- Gromov, M. (1981), ‘Groups of polynomial growth and expanding maps,’ *Publications Mathématiques de l’Institut des Hautes Études Scientifiques*, 53, 53–78.
- Gromov, M. (2007), *Metric structures for Riemannian and non-Riemannian spaces*, Springer Science & Business Media.
- Hagberg, A., Swart, P., and Schult, D. (2008), ‘Exploring network structure, dynamics, and function using NetworkX,’ Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
- Hagen, L. and Kahng, A. B. (1992), ‘New spectral methods for ratio cut partitioning and clustering,’ *IEEE transactions on computer-aided design of integrated circuits and systems*, 11, 1074–1085.
- Hall, K. M. (1970), ‘An r -dimensional quadratic placement algorithm,’ *Management science*, 17, 219–229.
- Heard, N. A., Holmes, C. C., and Stephens, D. A. (2006), ‘A quantitative study of gene regulation involved in the immune response of anopheline mosquitoes: an application of Bayesian hierarchical clustering of curves,’ *Journal of the American Statistical Association*, 101, 18–29.
- Heller, K. A. and Ghahramani, Z. (2005), ‘Bayesian hierarchical clustering,’ in *Proceedings of the 22nd International Conference on Machine Learning*, pp. 297–304.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012), ‘Improving neural networks by preventing co-adaptation of feature detectors,’ arXiv preprint arXiv:1207.0580.
- Horn, R. A. and Johnson, C. R. (2012), *Matrix analysis*, Cambridge university press.
- Horn, R. A., Rhee, N. H., and Wasin, S. (1998), ‘Eigenvalue inequalities and equalities,’ *Linear Algebra and its Applications*, 270, 29–44.

- Huelsenbeck, J. P. and Ronquist, F. (2001), ‘MRBAYES: Bayesian inference of phylogenetic trees,’ *Bioinformatics*, 17, 754–755.
- Jeong, H., Qian, X., and Yoon, B.-J. (2016), ‘Effective comparative analysis of protein-protein interaction networks by measuring the steady-state network flow using a Markov model,’ in *BMC bioinformatics*, vol. 17, pp. 15–27, BioMed Central.
- Jiang, B. and Lin, D. (2018), ‘Graph Laplacian Regularized Graph Convolutional Networks for Semi-supervised Learning,’ arXiv preprint arXiv:1809.09839.
- Kaiser, M. (2011), ‘A tutorial in connectome analysis: topological and spatial features of brain networks,’ *Neuroimage*, 57, 892–907.
- Ketkar, N. S., Holder, L. B., and Cook, D. J. (2009), ‘Faster computation of the direct product kernel for graph classification,’ in *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 267–274, IEEE.
- Kingma, D. P. and Ba, J. (2014), ‘Adam: A method for stochastic optimization,’ arXiv preprint arXiv:1412.6980.
- Kipf, T. N. and Welling, M. (2016), ‘Semi-supervised classification with graph convolutional networks,’ arXiv preprint arXiv:1609.02907.
- Kolouri, S., Park, S. R., Thorpe, M., Slepcev, D., and Rohde, G. K. (2017), ‘Optimal mass transport: Signal processing and machine-learning applications,’ *IEEE signal processing magazine*, 34, 43–59.
- Kolouri, S., Naderializadeh, N., Rohde, G. K., and Hoffmann, H. (2020), ‘Wasserstein embedding for graph learning,’ arXiv preprint arXiv:2006.09430.
- Koutra, D., Vogelstein, J. T., and Faloutsos, C. (2013), ‘Deltacon: A principled massive-graph similarity function,’ in *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 162–170, SIAM.
- Kriege, N. M., Johansson, F. D., and Morris, C. (2020), ‘A survey on graph kernels,’ *Applied Network Science*, 5, 1–42.
- Krogh, A. and Hertz, J. A. (1992), ‘A simple weight decay can improve generalization,’ in *Advances in neural information processing systems*, pp. 950–957.
- Kruskal, J. B. (1956), ‘On the shortest spanning subtree of a graph and the traveling salesman problem,’ *Proceedings of the American Mathematical Society*, 7, 48–50.
- Lai, R. and Zhao, H. (2017), ‘Multiscale Nonrigid Point Cloud Registration Using Rotation-Invariant Sliced-Wasserstein Distance via Laplace–Beltrami Eigenmap,’ *SIAM Journal on Imaging Sciences*, 10, 449–483.

- Langford, J., Li, L., and Zhang, T. (2009), ‘Sparse online learning via truncated gradient,’ in *Advances in neural information processing systems*, pp. 905–912.
- Lauritzen, S. L. (2011), ‘Elements of graphical models,’ *Lectures From the XXXVIth International Probability Summer School in St-Flour, France*.
- Ledoux, M. and Talagrand, M. (2013), *Probability in Banach Spaces: isoperimetry and processes*, Springer Science & Business Media.
- Lee, J. R., Gharan, S. O., and Trevisan, L. (2014), ‘Multiway spectral partitioning and higher-order cheeger inequalities,’ *Journal of the ACM (JACM)*, 61, 1–30.
- Levin, D. A. and Peres, Y. (2017), *Markov Chains and Mixing Times*, vol. 107, American Mathematical Soc.
- Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. (2022), ‘Sign and basis invariant networks for spectral graph representation learning,’ *arXiv preprint arXiv:2202.13013*.
- Linero, A. R. and Yang, Y. (2018), ‘Bayesian regression tree ensembles that adapt to smoothness and sparsity,’ *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 80, 1087–1110.
- Louis, A., Raghavendra, P., Tetali, P., and Vempala, S. (2012), ‘Many sparse cuts via higher eigenvalues,’ in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pp. 1131–1140.
- Lovász, L. and Winkler, P. (1993), ‘A note on the last new vertex visited by a random walk,’ *Journal of Graph Theory*, 17, 593–596.
- Luo, Z. T., Sang, H., and Mallick, B. (2021), ‘A Bayesian contiguous partitioning method for learning clustered latent variables,’ *The Journal of Machine Learning Research*, 22, 1748–1799.
- Luo, Z. T., Sang, H., and Mallick, B. (2023), ‘A nonstationary soft partitioned Gaussian process model via random spanning trees,’ *Journal of the American Statistical Association*, 0, 1–12.
- Maretic, H. P., Gheche, M. E., Chierchia, G., and Frossard, P. (2019), ‘GOT: An optimal transport framework for graph comparison,’ *arXiv preprint arXiv:1906.02085*.
- Massart, P. (2000), ‘Some applications of concentration inequalities to statistics,’ in *Annales de la Faculté des sciences de Toulouse: Mathématiques*, vol. 9, pp. 245–303.
- Matthews, P. (1988), ‘Covering problems for Markov chains,’ *The Annals of Probability*, 16, 1215–1228.

- Meilă, M. and Jaakkola, T. (2006), ‘Tractable Bayesian learning of tree belief networks,’ *Statistics and Computing*, 16, 77–92.
- Meilă, M. and Jordan, M. I. (2000), ‘Learning with mixtures of trees,’ *Journal of Machine Learning Research*, 1, 1–48.
- Mémoli, F. (2007), ‘On the use of Gromov-Hausdorff distances for shape comparison,’ .
- Meyer, C. D. (2023), *Matrix Analysis and Applied Linear Algebra*, vol. 188, SIAM.
- Neal, R. M. (2003), ‘Density modeling and clustering using Dirichlet diffusion trees,’ *Bayesian Statistics*, 7, 619–629.
- Neyshabur, B., Bhojanapalli, S., and Srebro, N. (2017), ‘A pac-bayesian approach to spectrally-normalized margin bounds for neural networks,’ arXiv preprint arXiv:1707.09564.
- Ng, A., Jordan, M., and Weiss, Y. (2001), ‘On spectral clustering: Analysis and an algorithm,’ *Advances in neural information processing systems*, 14.
- Niepert, M., Ahmed, M., and Kutzkov, K. (2016), ‘Learning convolutional neural networks for graphs,’ in *International conference on machine learning*, pp. 2014–2023, PMLR.
- Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. (2019), ‘Graph kernels: A survey,’ arXiv preprint arXiv:1904.12218.
- Oles, V., Lemons, N., and Panchenko, A. (2019), ‘Efficient estimation of a Gromov–Hausdorff distance between unweighted graphs,’ arXiv preprint arXiv:1909.09772.
- Payne, R. D., Guha, N., and Mallick, B. K. (2024), ‘A Bayesian survival treed hazards model using latent Gaussian processes,’ *Biometrics*, 80, ujad009.
- Petersen, K. B., Pedersen, M. S., et al. (2008), ‘The matrix cookbook,’ *Technical University of Denmark*, 7, 510.
- Polson, N. G. and Ročková, V. (2018), ‘Posterior concentration for sparse deep learning,’ in *Advances in Neural Information Processing Systems*, pp. 930–941.
- Prim, R. C. (1957), ‘Shortest connection networks and some generalizations,’ *The Bell System Technical Journal*, 36, 1389–1401.
- Ricaud, B., Borgnat, P., Tremblay, N., Gonçalves, P., and Vandergheynst, P. (2019), ‘Fourier could be a data scientist: From graph Fourier transform to signal processing on graphs,’ *Comptes Rendus Physique*, 20, 474–488.

- Ruder, S. (2016), ‘An overview of gradient descent optimization algorithms,’ arXiv preprint arXiv:1609.04747.
- Rustamov, R. M. et al. (2007), ‘Laplace-Beltrami eigenfunctions for deformation invariant shape representation,’ in Symposium on geometry processing, vol. 257, pp. 225–233.
- Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems*, SIAM.
- Saad-Eldin, A., Pedigo, B. D., Priebe, C. E., and Vogelstein, J. T. (2021), ‘Graph Matching via Optimal Transport,’ arXiv preprint arXiv:2111.05366.
- Sanfeliu, A. and Fu, K.-S. (1983), ‘A distance measure between attributed relational graphs for pattern recognition,’ *IEEE transactions on systems, man, and cybernetics*, pp. 353–362.
- Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. (2017), ‘Group sparse regularization for deep neural networks,’ *Neurocomputing*, 241, 81–89.
- Shalev-Shwartz, S. and Ben-David, S. (2014), *Understanding machine learning: From theory to algorithms*, Cambridge university press.
- Shi, J. and Malik, J. (2000), ‘Normalized cuts and image segmentation,’ *IEEE Transactions on pattern analysis and machine intelligence*, 22, 888–905.
- Shuman, D. I., Ricaud, B., and Vandergheynst, P. (2016), ‘Vertex-frequency analysis on graphs,’ *Applied and Computational Harmonic Analysis*, 40, 260–291.
- Soundarajan, S., Eliassi-Rad, T., and Gallagher, B. (2014), ‘A guide to selecting a network similarity method,’ in *Proceedings of the 2014 Siam international conference on data mining*, pp. 1037–1045, SIAM.
- Spielman, D. (2019), ‘Spectral and Algebraic Graph Theory,’ .
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014), ‘Dropout: a simple way to prevent neural networks from overfitting,’ *The journal of machine learning research*, 15, 1929–1958.
- Suchard, M. A., Weiss, R. E., and Sinsheimer, J. S. (2001), ‘Bayesian selection of continuous-time Markov chain evolutionary models,’ *Molecular Biology and Evolution*, 18, 1001–1013.
- Sun, J., Boyd, S., Xiao, L., and Diaconis, P. (2006), ‘The fastest mixing Markov process on a graph and a connection to a maximum variance unfolding problem,’ *SIAM review*, 48, 681–699.
- Tam, E. and Dunson, D. (2020), ‘Fiedler Regularization: Learning Neural Networks with Graph Sparsity,’ arXiv preprint arXiv:2003.00992.

- Tantardini, M., Ieva, F., Tajoli, L., and Piccardi, C. (2019), ‘Comparing methods for comparing networks,’ *Scientific reports*, 9, 1–19.
- Teixeira, L. V., Assunção, R. M., and Loschi, R. H. (2019), ‘Bayesian space-time partitioning by sampling and pruning spanning trees,’ *The Journal of Machine Learning Research*, 20, 1.
- Tibshirani, R. (1996), ‘Regression shrinkage and selection via the lasso,’ *Journal of the Royal Statistical Society: Series B (Methodological)*, 58, 267–288.
- Tsitsulin, A., Mottin, D., Karras, P., Bronstein, A., and Müller, E. (2018), ‘Netlsd: hearing the shape of a graph,’ in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2347–2356.
- Tuck, J., Hallac, D., and Boyd, S. (2019), ‘Distributed majorization-minimization for Laplacian regularized problems,’ *IEEE/CAA Journal of Automatica Sinica*, 6, 45–52.
- Van Havre, Z., White, N., Rousseau, J., and Mengersen, K. (2015), ‘Overfitting Bayesian mixture models with an unknown number of components,’ *PloS One*, 10, e0131739.
- Vayer, T., Chapel, L., Flamary, R., Tavenard, R., and Courty, N. (2018), ‘Optimal Transport for structured data with application on graphs,’ *arXiv preprint arXiv:1805.09114*.
- Villani, C. (2009), *Optimal transport: old and new*, vol. 338, Springer.
- Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010), ‘Graph kernels,’ *Journal of Machine Learning Research*, 11, 1201–1242.
- Vladimirova, M., Verbeek, J., Mesejo, P., and Arbel, J. (2018), ‘Understanding priors in Bayesian neural networks at the unit level,’ *arXiv preprint arXiv:1810.05193*.
- Vogelstein, J. T., Perlman, E., Falk, B., Baden, A., Roncal, W. G., Chandrashekhar, V., Collman, F., Seshamani, S., Patsolic, J. L., Lillaney, K., et al. (2018), ‘A community-developed open-source computational ecosystem for big neuro data,’ *Nature methods*, 15, 846–847.
- Von Luxburg, U. (2007), ‘A tutorial on spectral clustering,’ *Statistics and computing*, 17, 395–416.
- Wagner, D. and Wagner, F. (1993), ‘Between min cut and graph bisection,’ in *Mathematical Foundations of Computer Science 1993: 18th International Symposium, MFCS’93 Gdańsk, Poland, August 30–September 3, 1993 Proceedings* 18, pp. 744–750, Springer.

- Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013), ‘Regularization of neural networks using dropconnect,’ in International conference on machine learning, pp. 1058–1066.
- Weinstein, J. N., Collisson, E. A., Mills, G. B., Shaw, K. R. M., Ozenberger, B. A., Ellrott, K., Shmulevich, I., Sander, C., Stuart, J. M., Network, C. G. A. R., et al. (2013), ‘The cancer genome atlas pan-cancer analysis project,’ *Nature genetics*, 45, 1113.
- Wills, P. and Meyer, F. G. (2020), ‘Metrics for graph comparison: a practitioner’s guide,’ *PloS one*, 15, e0228728.
- Wilson, D. B. (1996), ‘Generating random spanning trees more quickly than the cover time,’ in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 296–303.
- Wilson, R. C. and Zhu, P. (2008), ‘A study of graph spectra for comparing graphs and trees,’ *Pattern Recognition*, 41, 2833–2841.
- Wu, Y., Tjelmeland, H., and West, M. (2007), ‘Bayesian CART: prior specification and posterior simulation,’ *Journal of Computational and Graphical Statistics*, 16, 44–66.
- Xu, H., Luo, D., Zha, H., and Carin, L. (2019), ‘Gromov-wasserstein learning for graph matching and node embedding,’ in International conference on machine learning, pp. 6932–6941, PMLR.
- Zedek, M. (1965), ‘Continuity and location of zeros of linear combinations of polynomials,’ *Proceedings of the American Mathematical Society*, 16, 78–84.
- Zeng, J., Pang, J., Sun, W., and Cheung, G. (2019), ‘Deep graph Laplacian regularization for robust denoising of real images,’ in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 0–0.
- Zeng, Z., Tung, A. K., Wang, J., Feng, J., and Zhou, L. (2009), ‘Comparing stars: On approximating graph edit distance,’ *Proceedings of the VLDB Endowment*, 2, 25–36.
- Zeng, Z., Gu, Y., and Xu, G. (2023), ‘A tensor-EM method for large-scale latent class analysis with binary responses,’ *Psychometrika*, 88, 580–612.
- Zhang, C.-H. (2010), ‘Nearly unbiased variable selection under minimax concave penalty,’ .
- Zou, H. (2006), ‘The adaptive lasso and its oracle properties,’ *Journal of the American statistical association*, 101, 1418–1429.

Biography

Edric Tam attended Johns Hopkins University for his undergraduate degree. He triple majored in Biomedical Engineering, Applied Mathematics and Statistics, as well as Neuroscience, with a minor in Computational Medicine. He worked in the laboratory of Alex Kolodkin at the School of Medicine and at the Center for Imaging Science with Michael I. Miller and J. Tilak Ratnanather. He obtained his first Master of Science degree from Yale University in Biomedical Engineering, where he worked in the lab of John Carlson. He obtained his second Master of Science degree from the University of Chicago in Computer Science, where he worked with Veronika Ročková. He started his PhD in Statistical Science at Duke in 2018 and completed it in 2024. His PhD advisor is David Dunson.

He has performed research at various industry venues, including NextCapital (acquired by Goldman Sachs), Google (Search Team), Amazon Science, Apple AIML, Facebook and Pinterest Labs. His teaching and mentoring efforts at Duke were recognized by the Statistical Science department 3 years in a row with awards. After his PhD graduation, he will be the Warren Alpert Fellow in AI and Computational Biology at Stanford University, where he will be mentored by Barbara Engelhardt.