Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

ON NON-LINEAR NETWORK EMBEDDING METHODS

by
Huong Yen Le

As a **linear** method, spectral clustering is the only network embedding algorithm that offers both a provably fast computation and an advanced theoretical understanding. The accuracy of spectral clustering depends on Cheeger ratio defined as the ratio between the graph conductance and the 2^{nd} smallest eigenvalue of its normalized Laplacian. In several graph families whose Cheeger ratio reaches its upper bound of $\Theta(n)$, the approximation power of spectral clustering is proven to perform poorly. Moreover, recent **non-linear** network embedding methods have surpassed spectral clustering by state-of-the-art performance with little to no theoretical understanding to back them.

The dissertation includes work that: (1) extends the theory of spectral clustering in order to address its weakness and provide ground for a theoretical understanding of existing non-linear network embedding methods.; (2) provides non-linear extensions of spectral clustering with theoretical guarantees, e.g., via different spectral modification algorithms; (3) demonstrates the potentials of this approach on different types and sizes of graphs from industrial applications; and (4) makes a theory-informed use of artificial networks.

Below is an overview of preliminary work.

1. **Spectral Clustering Theory Extension.** The proof that every graph G has a spectral maximizer H such that H is $\tilde{O}(1)$[1]-cut similar with G. All of the eigenvalues of H are $O(\log n)$ away from the maximum possible by the cuts of G. The maximizer eliminates *elongated features* of the graph (i.e. graph with high diameter) via long-range connections without changing its cuts. H achieves theoretically optimal Cheeger ratio that improves the cut of the original graph G .

2. **A Non-linear Extension of Spectral Clustering.** The developed theory holds for any graph $\tilde{H}$ which is spectral similar to H. We modify input G non-linearly into a graph $\tilde{H}$ similar to H and then apply standard spectral clustering on $\tilde{H}$ to obtain **favorable** Cheeger Ratio. **A framework for spectral modification** is based on a heuristic 'energy-based' tree decomposition approach. Construct M from G. M is cut-similar to G and is spectrally closer to maximizer H of G, relative to G itself. M can be computed in nearly-linear time.

3. **Demonstrate the Potentials of Spectral Modification.** Implementing spectral clustering on the approximation of H and mapping the output back to G has the potential to 'discover' dramatically different and improved cuts. Interesting enough, experiments have been performed on some popular networks with already high 2^{nd} eigenvalue in semi-supervised learning for multi-classification task and still yield very good accuracy scores during multi-fold cross-validation.

4. **Theory-informed Use of Artificial Networks for Network Embedding.** Geometric interpretations of spectral embedding using the baseline spectral method are demonstrated to reach state-of-the-art performance in semi-supervised classification tasks when boosted by a variety implementation of neural networks.

[1] $\tilde{O}(1)$ is used to hide factors algorithmic in n

ON NON-LINEAR NETWORK EMBEDDING METHODS

by
Huong Yen Le

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2021

APPROVAL PAGE

ON NON-LINEAR NETWORK EMBEDDING METHODS

Huong Yen Le

Dr. Ioannis Koutis, Dissertation Advisor — Date
Associate Professor of Computer Science, NJIT

Dr. Ali Mili, Committee Member — Date
Professor of Computer Science, NJIT

Dr. Jason T. L. Wang, Committee Member — Date
Professor of Computer Science, NJIT

Dr. Mihai Cucuringu, Committee Member — Date
Associate Professor of Statistics, University of Oxford, Oxford, United Kingdom

---

Dr. Senjuti Basu Roy, Committee Member — Date
Assistant Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Huong Yen Le

Degree: Doctor of Philosophy

Date: August 2021

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 2021

- Bachelor of Science in Finance
  Louisiana State University, Baton Rouge, LA, 2005

Major: Computer Science

Presentations and Publications:

Ioannis Koutis and Huong Le. Spectral Modification of Graphs for Improved Spectral Clustering. In *Advances in Neural Information Processing Systems*, Vancouver, BC/Canada: Curran Associates, Inc., Vol 32, 2019. NeurIPS.

'I hear and I forget. I see and I remember. I do and I understand.' An anonymous quote that summarizes my experience in academia. It has been a rewarding journey full of exciting surprises in knowledge, discovery and hard work. I can't wait to contribute to what lies ahead, knowing computer science will have such a large scale impact in our future.

Huong Yen Le

ACKNOWLEDGMENT

No words can describe the gratitude for my advisor Dr. Ioannis Koutis in providing me the guidance I needed to truly appreciate the beauty and necessity of Theoretical Computer Science, and in giving me the honor to work and study with him.

I also offer my earnest gratitude for the encouragement and guidance that my dissertation committee: Dr. Ali Mili, Dr. Jason T. L. Wang, Dr. Mihai Cucuringu, and Dr. Senjuti Basu Roy have given me to continue to explore different frontiers on my research topic.

My wholehearted thanks to Dr. Vincent Oria for seeing the ability in me and his initial guiding me to follow my dream to pursue a Ph.D. degree in Computer Science.

I would also like to express my special thanks to ADP, LLC for their generous support to my study and research.

This journey has brought me so much realization of how lucky I am to get every possible support from my wonderful family, especially my mother, Oanh Nguyen, that allows me to work toward my academic goals and more.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Spectral clustering is a widely known family of algorithms that uses eigenvectors to partition the vertices of a graph into meaningful clusters. First introduced in the work of Donath and Hoffman [11] who proved the lower bounds for the bi-partitioning of random graphs, spectral clustering sees its popularity grow substantially thanks to Shi and Malik [45], who applied it toward computer vision and machine learning. While new clustering methods have since emerged including methods based on neural networks, spectral partitioning methods remain as a frequently used baseline and a serious contender to start-of-the-art graph embedding methods e.g., [38, 18, 56, 42].

The remarkable performance of spectral clustering outputs are backed by their approximation properties that are theoretically understood. Cheeger's inequality relates the eigenvalue λ_2 of the *normalized Laplacian* matrix to the *conductance* of graph G denoted $\phi(G)$. $\phi(G)$ is known as the "best threshold cut" and equal to the minimum value among all possible ratios of the number of edges cut over the volume of the smaller set after the partition.

Theorem 1.1.1 (Discrete Cheeger's Inequality [8])**.** For any graph G,

$$\lambda_2/2 \leq \phi(G) \leq \sqrt{2\lambda_2}$$

Both sides of the above inequality are tight. The left side is tight for the hypercube $\{0,1\}^k$ where $\lambda_2 \approx \phi(G) \approx k$ and the right side is tight for a cycle where $\lambda_2 \approx 1/n^2$ while $\phi(G) \approx 1/n$.

Theorem 1.1.1 shows that while λ_2 is never greater than ϕ, it can be as small as ϕ^2. It also implies that the spectral cut approximation can be a factor of (ϕ/λ_2) away

from the optimal value, which can be up to $\Theta(n)$ on both weighted and unweighted graphs.

There are known graph families where λ_2 is too small which in turn affect the ability to approximate the optimal cut [19] and thus, yield a cut far away from the solution. This motivates the work to research for a **spectral modification** method to 'raise' a graph spectrum (i.e., the set of graph eigenvalues of the Laplacian matrix of the graph) while preserving its cut structure approximately. In effect, the quality of spectral clustering is improved via the suppression of the ratio (ϕ/λ_2).

1.2 Spectral Modification: High-level Overview and Context

This section puts spectral modification in line with important spectral graph theory development and discoveries that prime a knowledge baseline for later technical details. Unless otherwise stated, all graphs discussed in this dissertation are undirected weighted graphs.

1.2.1 Spectral and Cut Similarities

Let $G = (V, E, w)$ be a graph. $|V| = n$. A is the $n \times n$ adjacency matrix of G such that each of the matrix entry $A(u, v)$ equal to the weight of the corresponding edge $(u, v) \in E$ and $u \neq v$. Let D denote the diagonal matrix with each diagonal entry equal to the sum of the weights of the adjacent edges incident with the corresponding vertex. The Laplacian matrix L_G of the graph G is defined to be $D - A$.

A $cut_G(S)$ is the total weights of edges crossing two separated vertex sets of V: the set S and its complement $\bar{S} = V - S$ s.t. $S \cap \bar{S} = V$ and $S \cup \bar{S} = \emptyset$.

Let $\mathcal{R}(A, \mathbf{x}) = \mathbf{x}^T A\mathbf{x}$ denote the **quadratic form** of a semi-positive definite matrix A. Clearly, the Laplacian L_G of a graph is a semi-positive definite matrix with $\mathbf{x}^T L_G \mathbf{x} \geq 0$ for any real vectors $\mathbf{x} \in \mathbb{R}^{|V|}$. Let G and H be graphs and $\rho = \alpha/\beta$.

Spectral Similarity G and H are **ρ-spectral similar** if

$$\alpha \cdot \mathcal{R}(L_H, \mathbf{x}) \leq \mathcal{R}(L_G, \mathbf{x}) \leq \beta \cdot \mathcal{R}(L_H, \mathbf{x})$$

Cut Similarity G and H are **ρ-cut similar** if

$$\alpha \cdot cut_H(S) \leq cut_G(S) \leq \beta \cdot cut_H(S)$$

ρ-spectral similarity implies ρ-cut similarity but not vice-versa [48].

1.2.2 Low-diameter Cut Approximators and Spectral Maximizers

Let $G = (V, E)$ be the unweighted path graph on n vertices, and for the sake of simplicity assume that n is a power of 2. Let $\mathcal{T} = (V \cup I, E)$ be the full binary tree, where V is the set of leaves being in one-to-one correspondence with the path vertices as illustrated in Figure 1.1, and I is the set of internal vertices. The weight of each edge of $\mathcal{T}$ is equal to the total weight of all edges that need cutting from the path G to separate the leaf nodes into separate clusters.

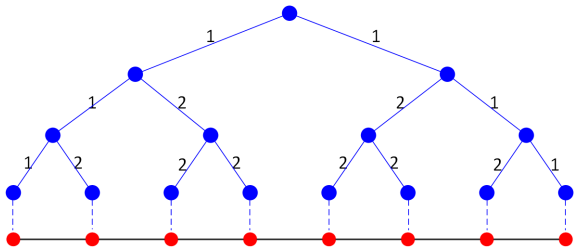


Figure 1.1 The unweighted path graph and its cut-approximating binary tree $\mathcal{T}$.

An interesting feature of $\mathcal{T}$ is that it provides a **cut approximator** for G, i.e., it contains information that allows estimating all cuts in G, within a factor of 2. Specifically, each edge of $\mathcal{T}$ carries the weight equal to the sum of the weights of crossing edges if the external vertices as part of V are separated in G. Section 4.1 describes how the cut approximator of $\mathcal{T}$ gives rise to a weighted complete graph $H = (V, E, w)$ on the original set of vertices V, via a canonical process of eliminating the internal vertices of $\mathcal{T}$.

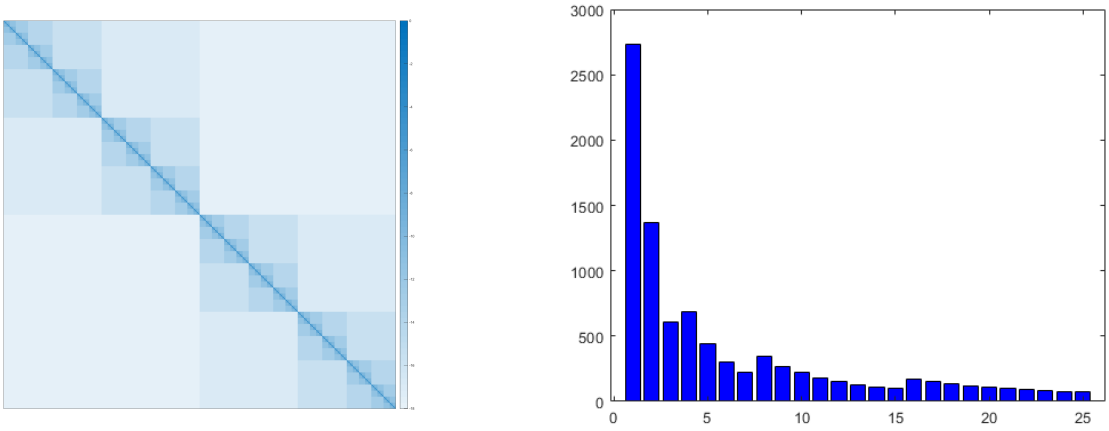


Figure 1.2 **Left:** Heat map of the log-entries of the adjacency matrix (Darker color denotes higher value) of the spectral maximizer H. **Right:** Ratios of the first 25 normalized eigenvalues of H and G in order.

Let G be the cross-product of a double binary tree graph with a path graph, the left of Figure 1.2 provides a glimpse to the edge weights of H in the case the number of vertices of G $|V_G| = n = 8196$. It can be seen that H is a dense graph that inherits the tri-diagonal path structure, but also has other long-range edges. H is $O(\log n)$-cut similar with G, but with a very different eigenvalue distribution, as illustrated on the right of Figure 1.2. More specifically, H has significantly larger eigenvalues. λ_2 of the normalized Laplacian of G is $\Theta(1/n^2)$, while that of H is $\Omega(1/(n \log n))$, essentially closing the gap with the conductance $\phi = \Theta(1/n)$. An alternative way of viewing this is that H has λ_2 which – up to an $O(\log n)$ factor – is the maximum possible, since the eigenvalue is always smaller than ϕ. In some sense, the same is true for all eigenvalues of H, which leads us to call H a **spectral maximizer** of G. These properties of H can be proven using only the logarithmic diameter of $\mathcal{T}$ and the fact that $\mathcal{T}$ is a cut-approximator.

In [43], Räcke showed that all graphs have low-diameter cut approximators. This backs the central claim of this thesis that every graph has a spectral maximizer. The example of the path graph and its cut approximator is a simplest case to demonstrate this claim while a vast generalization can be done with a small loss. Section 4.4

discusses Cheeger inequalities for spectral maximizers and shows that the inequality applies not only in the standard normalized cuts problem, but also in generalized cut problems that capture semi-supervised clustering problems.

These observations set the backdrop for the idea of spectral modification, which aims to modify the input graph G to arrive at its maximizer H. It is worth noting that, in some sense, the objective of spectral modification counters that of spectral graph sparsification, which aims to spectrally preserve a graph while approximating its sparse substitute [4].

CHAPTER 2

RESEARCH CONTRIBUTIONS AND KEY APPLICATIONS

2.1 Contributions

The research motivation has allowed three contributions related to spectral graph theory: (1) Spectral clustering theory extension; (2) A non-linear extension of spectral clustering as an algorithm framework that modifies the input graph G into a graph $\mathcal{M}$ which is spectrally closer to maximizer H of G, which produces a much more accurate spectral clustering results; and (3) A theory-informed use of neural networks to boost the accuracy of spectral clustering.

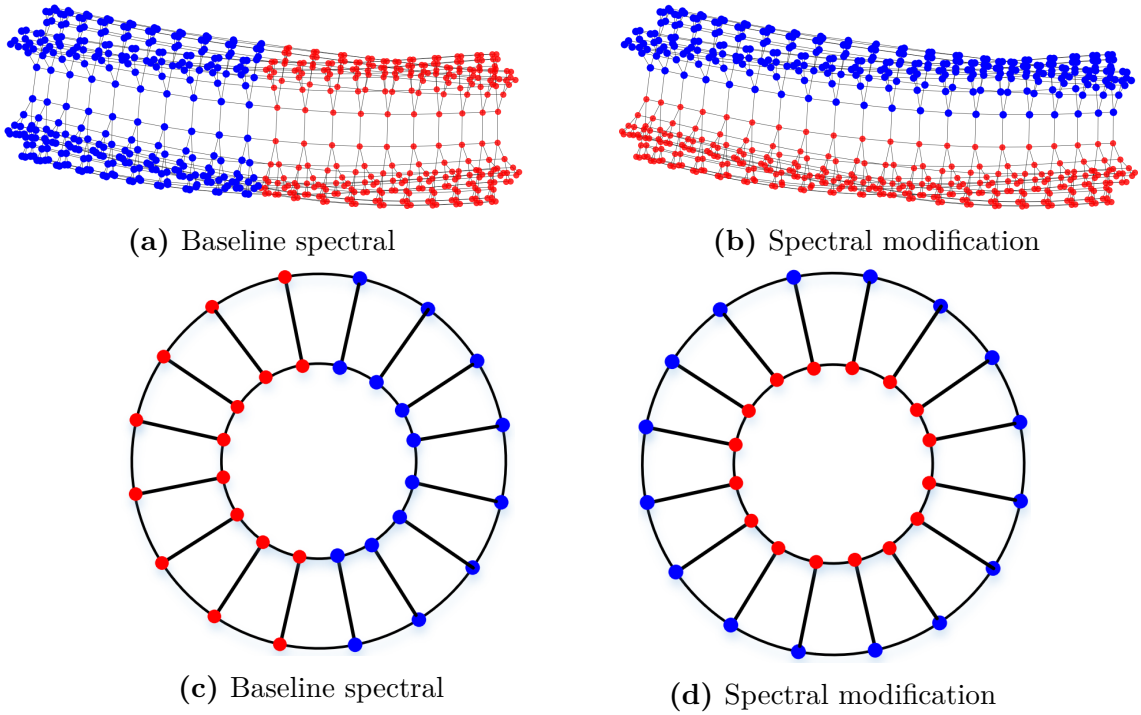
2.1.1 Spectral Clustering Theory Extension

The spectral partition of a graph G depends on how well λ_2 approximate the graph conductance ϕ. The quality of the cuts deteriorates as the Cheeger ratio ϕ/λ_2 increases. This ratio can be up to $\Theta(n)$ and leads to poor approximation of spectral clustering. Certain graph families demonstrates these *loose* bounds [19]. On the other hand, a graph H as a spectral maximizer of graph G will allow a *tight* Cheeger inequality. All eigenvalues of H are $O(\log n)$ away from the maximum allowed by the cuts of G. A spectral maximizer of a graph eliminates its 'elongated features' via long-range connections without changing its cuts. The proof that every graph has a spectral maximizer is presented in Section 4.5. The benefit of such tightness is a suppression in the ratio between ϕ/λ_2 to reach the optimal value of $O(\log n)$ so that the accuracy of the baseline spectral clustering method can be realized.

2.1.2 A Non-linear Extension of Spectral Clustering

The developed theory holds for any graph $\tilde{H}$ which is spectral similar to the spectral maximizer H. **Spectral modification** is the approach to modify any graph G non-

linearly to obtain its corresponding $\tilde{H}$. Due to the bounded spectral similarity, $\tilde{H}$ also has a favorable Cheeger ratio.



(a) Baseline spectral

(b) Spectral modification

(c) Baseline spectral

(d) Spectral modification

Figure 2.1 **Top:** Input G is a direct graph product of the path graph and a graph consisting of two binary trees with their roots connected [19]. Two way partitioning results. (a) By baseline spectral clustering (b) Improved With Graph Modification $\mathcal{M}$ of G. Spectral modification **sways** the lowest eigenvector away from the cut computed in G. The asymptotic improvement in the value of the cut is $O(n^{1/4})$. **Bottom:** Two unit-weight cycle joined by n edges of weight $100/n^2$ [19]. Two way partitioning results. (c) By baseline spectral clustering which cuts four unit edges and breaks both circles in half. (d) Modified spectral clustering cuts the n light edges and separates the two cycles ($O(n)$ asymptotic improvement).

Spectral modification starts with the construction of cut approximators. While constructing a cut approximator results in the costs of all possible cuts in a graph, it is, arguably, a waste if the only objective is to compute a k-clustering. This motivates the second contribution of a fast algorithm to construct graph maximizers with **low** run time and sized output so that any negative impact on the speed of spectral clustering is minimized.

[5][25][37][44] have improved and refined much of the original result of [43]. It is now possible to compute a cut approximator in nearly-linear time ($O(m \log^c m)$) where m is the number of edges in the graph [37]). This implies a similar time for the construction of a maximizer. $\tilde{H}$ can be constructed efficiently, without resorting to the single-tree cut approximators used for the proofs via the spectral modification framework. The non-linear extension has theoretical guarantees via different spectral modification algorithms discussed in Chapter 5. The framework for spectral modification is based on a heuristic 'energy-based' tree decompositions to obtain $\tilde{H}$ in nearly linear time.

2.1.3 Theory-informed Use of Neural Networks

Multi-layer convolutional networks have been around since the 70s. Several works explored conditions that deep networks perform better than shallow ones to reduce the complexity for approximation and learning [34][35][39]. Nevertheless, it is known that the approximation power of both shallow and deep networks are at an exponential cost [40]. While many networks achieve impressive accuracy results in semi-supervision tasks on standard datasets [38][56][42][18], tt is not well understood why they find such nice solutions despite the complexity of the loss function landscape. Moreover, a lack of theory backing makes the adoption of neural networks purely mechanical and experimental in nature. A shallow neural network coupled with the knowledge of geometric characteristics of spectral embeddings brings about state-of-the-art performance in semi-supervision tasks.

2.2 Some Key Applications

2.2.1 Improved Cuts with Graph Modification

Perhaps, the most astonishing benefit of working with graph $\mathcal{M}$ which preserves graph G spectral structure is the potential to correct and improve the answer to the

least expensive cut problem. Figure 2.1 shows that performing spectral clustering on the spectrally modified graph in practice solves graph families that are unsolvable by baseline spectral clustering [19].

2.2.2 Support for Regularized Spectral Clustering

It has been observed that adding a small copy of the identity matrix or the complete graph generated based on the input graph G to the input graph G improves the quality of spectral clustering [2][41]. This graph modification idea is termed as **regularized spectral clustering**. [22][57] explain this improved performance for block-stochastic models and stochastic social network graphs. Specifically, [57] shows that adding a complete graph suppresses the unbalanced sparse cuts in graph G caused by altering their cut ratios. The theoretical results of this research dissertation will help shed additional light on regularized spectral clustering. It is important to note that on its own, regularized spectral clustering does not improve cut result as shown in Figure 2.1.

CHAPTER 3

BACKGROUND

This section discusses fundamental and relevant results in spectral graph theory that motivates and support non-linear networking embedding via spectral modification. For more information about spectral graph theory fundamental, the reader is referred to [8].

3.1 Fundamental of Spectral Graph Theory

3.1.1 Graph Conductance

For a weighted graph $G = (V, E)$, $|V| = n$, $|E| = m$,$S \cap \bar{S} = V$, $S \cup \bar{S} = \emptyset$, $|S| \leq n/2$, the *edge boundary of* S is defined as the total weight of edges that cross over from set S to set $\bar{S}$ is denoted as

$$cut(S) = \sum_{u \in S, v \in \bar{S}} w(u, v)$$

The total degree of the vertices in S is denoted as

$$vol(S) = \sum_{v_i \in S} d(v_i)$$

where $d(v_i)$ is the degree of a vertex v_i in G and $d(v_i) = \sum_{v_j} w(v_i, v_j)$. The conductance of set S (a.k.a., the Cheeger constant) denoted by $\phi(S)$ reflecting the sparsity of the cut is defined to be

$$\phi(S) := \frac{cut(S)}{min(vol(S), vol(\bar{S}))}$$

For any set $S \subseteq V$, $0 \leq \phi(S) \leq 1$. If $\phi(S) \approx 0$, S represents a cluster in G. the conductance of G is the smallest conductance among all possible values of $\phi(S)$ with

at most half of the total volume

$$\phi(G) = \min_{S \subset V : vol(S) \leq vol(\frac{V}{2})} \phi(S)$$

The usual way to judge the quality of the split is via the calculation of the *cut* and the graph conductance ϕ [23].

3.1.2 The Laplacian Matrix

The adjacency matrix A is defined as a $n \times n$ matrix whose entries corresponding to the weights between adjacent vertices

$$A_{u,v} = \begin{cases} w(u,v) & (u,v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

D is the diagonal degree matrix satisfying

$$D_{u,v} = \begin{cases} d(u) & u = v \\ 0 & \text{otherwise.} \end{cases}$$

L is the Laplacian matrix of G calculated by the subtraction of the adjacency matrix A from the diagonal degree matrix D.

$$L = D - A$$

L is better understood via its quadratic form, for all $\mathbf{x} \in \mathbb{R}^{|V|}$

$$\mathbf{x}^T L_G \mathbf{x} = \sum_{(u,v) \in E} w_{(u,v)} (\mathbf{x}(u) - \mathbf{x}(v))^2$$

This form expresses the smoothness of the function $\mathbf{x}$. It will be small if component values of $\mathbf{x}$ do not drastically change between any two vertices of any edge.

A cluster S in V can be represented by a vector χ_S where $\chi_S(u) = 1$ if $u \in S$ and $\chi_S(u) = 0$ otherwise, the total weight crossing from S to $\bar{S}$ in G can be expressed

as follows,

$$cut_G(S, \bar{S}) = \chi_S^T L_G \chi_S \tag{3.1}$$

The normalized Laplacian $\mathcal{N}$ of graph G is expressed as

$$\mathcal{N}_{i,j} := D^{-1/2} L D^{-1/2} = \begin{cases} 1 & \text{if } i = j,\ d_i \neq 0 \\ -\frac{w(v_i, v_j)}{\sqrt{d_i d_j}} & \text{if } i \neq j,\ d_i \neq 0,\ d_j \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

The Rayleigh quotient of a vector $\mathbf{x} \in \mathbb{R}^{|V|}$ with respect to a matrix A is the ratio

$$\frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$$

$\mathbf{x}$ is an eigenvector of A of eigenvalue λ if and only if

$$A\mathbf{x} = \lambda \mathbf{x} \tag{3.2}$$

Theorem 3.1.1 (Courant-Fischer)**.** Let A be a symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Then,

$$\lambda_k = \min_{\substack{S \subseteq \mathbb{R}^n \\ dim(S)=k}} \max_{\mathbf{x} \in S} \frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \max_{\substack{T \subseteq \mathbb{R}^n \\ dim(T)=n-k+1}} \min_{\mathbf{x} \in T} \frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$$

The full collection of eigenvalues of a matrix A makes up its spectrum. As $\mathbf{x}$ is an eigenvector of A, the Rayleigh quotient can be understood as

$$\frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \frac{\mathbf{x}^T \lambda \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \frac{\lambda \mathbf{x}^T \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \lambda$$

Set $\mathbf{y} = D^{-1/2}\mathbf{x}$. The generalized Rayleigh quotient for $\mathcal{N}$ can be presented as follows

$$\frac{\mathbf{x}^T \mathcal{N} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \frac{\mathbf{x}^T D^{-1/2} L D^{-1/2} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \frac{\mathbf{y}^T L \mathbf{y}}{\mathbf{y}^T D \mathbf{y}} \tag{3.3}$$

3.1.3 Cheeger's Inequality

In spectral graph theory, the number of connected components in an undirected graph is equal to the number of zero eigenvalues in the Laplacian matrix. Cheeger's inequality and its variants provide an approximate version of such fact, stating that a sparse cut exists in a graph if and only if there are at least two eigenvalues that are close to zero.

Let $\lambda_n \geq \lambda_{n-1} \geq \cdots \geq \lambda_2 \geq \lambda_1 = 0$ denote the eigenvalues of the normalized Laplacian. The graph conductance ϕ_G is related to λ_2 via the below discrete version of Cheeger's inequality,

$$\phi_G^2/2 \leq \lambda_2 \leq 2\phi_G \tag{3.4}$$

Equation (3.4) is equivalent to the expression $\lambda_2/2 \leq \phi_G \leq \sqrt{2\lambda_2}$ [8]. A deeper look into the proof following [53] that derives (3.4) helps clarify the relationship between graph conductance and the 2^{nd} smallest eigenvalue λ_2.

Let $s = vol(S)/vol(V)$ and $\chi_S = \{0,1\}^{|V|}$ s.t. $\chi_S(u) = 1$ if $u \in S$ and $\chi_S(u) = 0$ otherwise. Set $\mathbf{y} = \chi_S - s\mathbf{1}$. For $u \in S$, $\mathbf{y}(u) = 1 - s$. In all other cases, $\mathbf{y}(u) = -s$. $\mathbf{d}$ is the vector whose entries denote the degree of the vertices in the graph. By construction, $\mathbf{y}$ is orthogonal to both the constant vector $\mathbf{1}$ and $\mathbf{d}$

$$\mathbf{y}^T\mathbf{d} = \chi_S^T\mathbf{d} - s\mathbf{1}^T\mathbf{d} = (vol(S) - (vol(S)/vol(V))vol(V) = 0$$

The numerator of Equation (3.3) equals to

$$\mathbf{y}^T L\mathbf{y} = \sum_{(u,v)\in E} w(u,v)((\chi_S(u) - s) - (\chi_S(v) - s))^2 = cut(S)$$

The denominator of Equation (3.3) is computed as follows

$$\begin{aligned}\mathbf{y}^T D\mathbf{y} &= \sum_{u\in S}\mathbf{d}(u)(1-s)^2 + \sum_{u\notin S}\mathbf{d}(u)s^2 \\ &= vol(S)(1-s)^2 + vol(V-S)s^2 \\ &= vol(S) - 2vol(S)s + vol(V)s^2 \\ &= vol(S) - vol(S)s \\ &= vol(S)vol(V-S)/vol(V)\end{aligned}$$

The larger of $(vol(S), vol(V-S))$ is at least half of $vol(V)$. The eigenvector of λ_2 of the normalized Laplacian $\mathcal{N}$ is given by

$$\lambda_2 = \text{argmin}_{\mathbf{x}\perp\mathbf{d}^{1/2}}\frac{\mathbf{x}^T\mathcal{N}\mathbf{x}}{\mathbf{x}^T\mathbf{x}} = \min_{\mathbf{y}\perp\mathbf{d}}\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \tag{3.5}$$

So,

$$\lambda_2 \leq \frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} = \frac{cut(S)vol(V)}{vol(S)vol(V-S)} \leq 2\frac{cut(S)}{min(vol(S), vol(V-S))} = 2\phi_S$$

Thus, the right hand side of Equation (3.4) can be generalized to every subset S of V

$$\phi_S \geq \lambda_2/2$$

The left hand side of Equation (3.4) means that there exists a set of small conductance S_t such that $S_t = \{u : \mathbf{y}(u) < t \text{ for } t \in R\}$ satisfies

$$\phi(S_t) \leq \sqrt{2\lambda_2} \leq \sqrt{2\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}} \tag{3.6}$$

The eigenvector of eigenvalue λ_0 of $\mathcal{N}$ is $\mathbf{d^{1/2}}$ since

$$D^{-1/2}LD^{-1/2}\mathbf{d^{1/2}} = D^{-1/2}L\mathbf{1} = D^{-1/2}\mathbf{0} = 0\mathbf{d^{1/2}} = \mathbf{0}$$

$$\mathbf{x}^T\mathbf{d^{1/2}} = \mathbf{y}^T D^{1/2}\mathbf{d^{1/2}} = \mathbf{y}^T\mathbf{d}$$

Without loss of generality, the vertices can be renumbered so that $\mathbf{y}(1) \leq \mathbf{y}(2) \leq \cdots \leq \mathbf{y}(n)$. Let j be the least number for which

$$\sum_{u=1}^{j} \mathbf{d}(u) \geq vol(V)/2$$

Let vector $\mathbf{z}$ be centered at j

$$\mathbf{z} = \mathbf{y} - \mathbf{y}(j)\mathbf{1}$$

The arrangement of $\mathbf{y}$ allows the maintance of the relation: $\mathbf{z}(1) \leq \mathbf{z}(2) \leq \cdots \leq \mathbf{z}(n)$. In addition, $\mathbf{z}(1) \leq \mathbf{z}(j) = 0 \leq \mathbf{z}(n)$. $\mathbf{z}$ is multiplied by a constant so that

$$\mathbf{z}^2(1) + \mathbf{z}^2(n) = 1$$

To satisfy (3.6), it is necessary to prove the existing of a distribution on t such that $S_t = \{u : z(u) \leq t\}$. The value of t is chosen from the probability density function $2|t|$ so that

$$\int_{\mathbf{z}(1)}^{\mathbf{z}(n)} 2|t| = \int_{\mathbf{z}(1)}^{0} 2|t| + \int_{0}^{\mathbf{z}(n)} 2|t| = \mathbf{z}^2(1) + \mathbf{z}^2(n) = 1$$

An edge (u, v) with $\mathbf{z}(u) \leq \mathbf{z}(v)$ is part of the $cut(S)$ if

$$\mathbf{z}(u) \leq t < \mathbf{z}(v)$$

With $\mathbf{z}(j) = 0$, the probability that t lies in between two vertices (u, v) is

$$\int_{u}^{v} 2|t| = \begin{cases} |\mathbf{z}^2(u) - \mathbf{z}^2(v)| & \text{if } \mathbf{z}(u)\mathbf{z}(v) \geq 0 \\ \mathbf{z}^2(u) + \mathbf{z}^2(v) & \text{if } \mathbf{z}(v)\mathbf{z}(j) < 0 \end{cases}$$

The centering by satisfying $\mathbf{z}(j) = 0$ guarantees that

$$t < 0 \rightarrow vol(S) = min(vol(S), vol(V - S)), \text{ and}$$
$$t \geq 0 \rightarrow vol(V - S) = min(vol(S), vol(V - S))$$

The expected value of the minimum between the $vol(S_t)$ and $vol(V - S_t)$ is the total sum of the probability of $(u) < t < 0$ and $(v) > t \geq 0$ while u is a vertex that is either behind or in front of j.

$$\begin{aligned} E[\min(vol(S), vol(V - S))] &= \sum_{u<j} Pr[\mathbf{z}(u) < t < 0]\mathbf{d}(u) + \sum_{u \geq j} Pr[\mathbf{z}(u) > t \geq 0]\mathbf{d}(u) \\ &= \sum_{u<j} \mathbf{z}^2(u)\mathbf{d}(u) + \sum_{u \geq j} \mathbf{z}^2(u)\mathbf{d}(u) \\ &= \sum_{u} \mathbf{z}^2(u)\mathbf{d}(u) = \mathbf{z}^T D\mathbf{z} \end{aligned}$$

The expected $cut(S_t)$ is the sum of the probabilities of choosing t such that t is in the middle of two vertices (u, v) where only once of such vertex is part of S_t.

$$\begin{aligned} E[cut(S_t)] &= \sum_{(u,v) \in E} Pr_t[(\mathbf{z}(u)\mathbf{z}(v) < 0)]w_{u,v} \leq \sum_{(u,v) \in E} |\mathbf{z}(u) - \mathbf{z}(v)|(|\mathbf{z}(u)| + |\mathbf{z}(v)|)w_{u,v} \\ &\leq \sqrt{\sum_{(u,v) \in E} w_{u,v}(\mathbf{z}(u) - \mathbf{z}(v))} \sqrt{\sum_{(u,v) \in E} w_{u,v}(|\mathbf{z}(u)| + |\mathbf{z}(v)|} \text{ (Cauchy-Schwartz)} \\ &\leq \sqrt{\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \sum_u \mathbf{z}^2(u)\mathbf{d}(u)} \sqrt{2 \sum_u \mathbf{z}^2(u)\mathbf{d}(u)} \\ &= \sqrt{2\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \sum_u \mathbf{z}^2(u)\mathbf{d}(u)} \sqrt{\sum_u \mathbf{z}^2(u)\mathbf{d}(u)} \\ &= \sqrt{2\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}} E[min(vol(S_t), vol(V - S_t))] \end{aligned}$$

Move the expected value of the minimum of the total degrees $E[min(vol(S_t), vol(V - S_t))]$ to the left hand side of the previous Equation, the conductance of the graph is shown to meet the Equation $\phi_G \leq \phi(S_t) \leq \sqrt{2\frac{\mathbf{y}^T L\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}}$.

For a path graph P_n of n vertices, ϕ_P is $1/\lfloor (n-1)/2 \rfloor$, $\lambda_2 \approx \frac{\pi^2}{2(n-1)^2}$. Cheeger inequality $\lambda_2 > \phi^2/2$ is best possible up to a constant factor. For a hypercube Q_n, ϕ_Q is equal to its $\lambda_2 = 2/n$, which is tight for $2\phi \geq \lambda_2$.

3.1.4 Network Partitioning

Multiple types of data can be naturally represented as a network (i.e., graph). Network partitioning is generally understood as the process of decomposing a graph into mutually exclusive sets of vertices. The partitioning problem may be posed with or without clustering constraints. As the only input, graph can contain information of edges, their corresponding weights, and geometric information about the locations of the vertices. Clustering constraints, if available, reflect domain knowledge under the form of must-link (ML) indicating certain vertices must belong to the same group and cannot-link (CL) mapping out which vertices cannot be in the same group [3].

For a balanced partition problem, graph partitioning aims to separate the graph into pieces of roughly equal number of vertices by removing either edges or vertices called *edge*, and *vertex separator* respectively. The goal is to find small separators whose cuts are as sparse as possible.

While the problem of bisecting a graph into two halves is NP-complete [15], graph partitioning as a general problem is NP-hard. Therefore, its solutions are derived from heuristics and approximation algorithms. Combinatorial or geometric methods are the two broad categories of algorithms that solve graph partitioning.

Spectral Clustering Spectral clustering [11] belongs to a family of combinatorial algorithms which perform graph partitioning based on only graph connectivity information. Spectral partitioning is well-known as the most successful class of algorithms to the sparsest-cut problem, partitioning graphs and matrices [14][8][47] utilizing invariant properties of the graph spectrum.

Spectral partitioning solves the two-way clustering problem to bisect a graph by using the 2^{nd} smallest eigenvalue λ_2 of the Laplacian, which reveals the connectivity of the graph. The eigenvector $\mathbf{x}_2$ (a.k.a., the Fiedler vector) corresponding to λ_2 contains information about the relative distances between vertices [14]. Spectral clustering is implemented by calculating $\mathbf{x}_2$, sorting the entries, and partitioning the corresponding vertices about a splitting value s. Popular choices for s includes (i) the median of $\mathbf{x}_2$, (ii) the value that gives the best ratio cut, (iii) 0, and (iv) the value that is equal the largest gap in the sorted components of $\mathbf{x}_2$ [47].

A graph can also be decomposed into more than two subgraphs by using information of higher order eigenvalues. Throughout this thesis, the standard spectral partitioning approached is referred to as *baseline spectral*. While baseline spectral performs well in practice, it provides a bad separator for several bounded-degree graph families as shown by Guattery and Miller [19].

Spectral Clustering as a Semi-supervised Method Spectral clustering is generally viewed as an **unsupervised** method. However it has been suggested that it can also work as a **semi-supervised** algorithm via computing generalized eigenvectors [10]. Corollary 4.4.1 shows that the theoretical performance of this kind of semi-supervised spectral clustering has the potential to be much more accurate than predicted by the generalized inequality of [10] if the input graph is spectrally close to the maximizer of its cut structure.

Corollary 4.4.1 also has consequences for classical algorithmic problems. For instance, the isoperimetric number of a graph G is often defined as

$$h = \min_{S \subseteq V} cut_G(S)/(|S| \cdot |V - S|).$$

If B is the complete unweighted graph then $h = \phi(G, B)$. The isoperimetric number has a weaker Cheeger inequality, namely $\lambda_2(L_G) \geq h^2/(2d_{\max})$, where $d_{\max}$ is the

maximum degree of the graph. Then inequality 4.1 applies directly and gives a different and usually stronger estimate. A similarly interesting inequality follows for the minimum s-t cut problem, if B is set to be the graph consisting only of the (s, t) edge.

One-vs-Rest Classifier for Multi-label Classification Task Most classification predictive algorithms such as Logistic Regression were designed for binary classification and do not support classification tasks with more than two classes natively. The approach chosen for performance checking and reporting follows the universally accepted approach One-vs-Rest which splits the multi-class classification datasets into multiple binary classification datasets with one binary classification problem per class. The returned estimates are marginal probability values from 0 to 100% that a given sample belongs to a class. It is entirely possible that multiple classes have the same probability for a given sample. During micro score calculation, the number of classes is determined per sample and the returned probabilities are sorted which gives the indices of the same number of expected classes with highest probabilities.

Binary classification with Logistic Regression Logistic Regression is a linear statistical model estimating the parameters of a logistic function with a common S-shaped curve

$$f(x) = \frac{M}{1 + e^{-k(x-x_0)}}$$

where x is the sample value, x_0 is the x value of the sigmoid's midpoint. M is the curve's maximum value and k is the logistic growth rate, also known as the steepness of the curve. The solver `LIBLINEAR` uses a coordinate descent algorithm to train separate binary classifiers for all classes. It can accommodate $l1$ regularization by

solving the optimization problem:

$$min_{m,c}||w||_1 + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

$l2$ regularization can also be applied to penalize logistic regression by minimizing the cost function

$$min_{m,c}\frac{1}{2}w^T w + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

No Regularization While NetMF uses regularization by default for its embeddings, spectral modification embeddings do not require regularization due to the radial projection step that places the embedding onto the unit hypersphere [32] with equal distance from its center.

Geometric Clustering Methods Graph partitioning could also be performed via geometric methods. Geometric methods slice vertices based on their corresponding spatial coordinates. Spatial coordinates can be generated via the use of network embedding algorithms discussed in Section 3.2.4 or through feature engineering efforts.

Geometric clustering could be viewed as a generalization of spectral clustering. Section 3.2.4 discusses the geometric properties of spectral eigenvectors when projected onto a unit sphere [32], the result of which can be instrumental in both semi-supervised and unsupervised machine learning applications. A combination of geometric methods and combinatorial algorithm of baseline spectral yields state-of-the-art result as extended in Chapter 7.

3.2 Relevant Development In Spectral Graph Theory

3.2.1 A Generalized Cheeger Inequality

The problem of graph partitioning with domain knowledge were commonly solved as extensions and modifications of the basic spectral method [45][36]. It was not

until [10] where Cucuringu et al. generalized it as an optimization problem of two weighted graphs: the ML constraint graph G, and the CL constraint graph H. H does not have to be connected. This generalization is *organic* as the basic spectral method of clustering is shown to be a special case of constrained clustering with *implicit* soft ML and CL constraint graphs. Indeed, graph edge data can be viewed as 'soft' declaration that two adjacent vertices should be connected rather than disconnected. The *demand graph* K derived from G as $K_{ij} = \mathbf{d}_i\mathbf{d}_j/vol(V)$ contains implicit 'soft' CL constraints. It is easy to see that $vol(S)vol(\bar{S})/vol(V) = cut_K(S, \bar{S})$. Therefore, the optimization problem in the standard method equates to the objective to minimize the ratio between weight of cut severing implicit ML constraints and weight of cut complying with implicit CL constraints as

$$min_{S\subseteq V}\frac{cut_G(S,\bar{S})}{vol(S)vol(\bar{S})/vol(V)} = min_{S\subseteq V}\frac{cut_G(S,\bar{S})}{cut_K(S,\bar{S})}$$

Furthermore, if the goal is to partition the vertices into k disjoint clusters C_i where a partitioned group i shows a cost of

$$\phi_i(G,H) = \frac{cut_G(C_i,\bar{C}_i)}{cut_H(C_i,\bar{C}_i)} = \frac{\mathbf{x}_{C_i}^T L_G \mathbf{x}_{C_i}}{\mathbf{x}_{C_i}^T L_H \mathbf{x}_{C_i}} \quad \text{(Equation (3.1))} \tag{3.7}$$

it is sensible to make the objective to solve for the optimization problem of minimizing individual cost ϕ_i

$$\Phi = \mathbf{min}\ \underset{i=1,\cdots,k}{max}\ \phi_i$$

In other words, this minimization problem is solved by finding k vectors in $\{0,1\}^n$ with disjoint support. To relax this NP-complete problem, the k vectors, which minimize the maximum among the k Rayleigh quotients $(\mathbf{x}_{C_i}^T L_G \mathbf{x}_{C_i})/(\mathbf{x}_{C_i}^T L_H \mathbf{x}_{C_i})$, are the generalized eigenvectors corresponding to the k smallest eigenvalues of the problem $L_G\mathbf{x} = \lambda L_H\mathbf{x}$. This approach provides a concrete theoretical guarantee for the two-way constrained partitioning via a generalized Cheeger inequality.

Theorem 3.2.1 (Generalized Cheeger Inequality [10])**.** Let G and H be any two weighted graphs and $\mathbf{d}$ be the vector containing the degrees of the vertices in G. Let also K be the demand graph and $\phi(G,H) = min_{S\subset V}\, cut_G(S,\bar{S})/cut_H(S,\bar{S})$. For all $\mathbf{x} \in \mathbb{R}^n$ s.t. $\mathbf{x}^T\mathbf{d} = 0$,

$$\frac{\mathbf{x}^T L_G \mathbf{x}}{\mathbf{x}^T L_H \mathbf{x}} \geq \phi(G,K)\phi(G,H)/4$$

3.2.2 Higher-order Cheeger Inequalities

As the generalized Cheeger inequality states, if the goal is to partition the vertices into k disjoint clusters C_i where each partitioned group shows a cost ϕ_i, the partitioning algorithm optimization needs to solve for minimizing the maximum ϕ_i [10]. [32] calls Φ_k (see Equation (3.7)) as the *k-way expansion constant* $\rho_G(k)$ for all collections of k non-empty, disjoint sub sets $\{S_i \subseteq V\}$ for $i = 1, \cdots, k$.

Theorem 3.2.2. [Higher-order Cheegeer Inequalities[32]] For every graph G and every $k \in \mathbb{N}$,

$$\lambda_k/2 \leq \Phi_k \leq O(k^2)\sqrt{\lambda_k}$$

Theorem 3.2.2 makes it feasible to find a partition P_i of V into k non empty sets such that each set has an expansion $O(k^3)\sqrt{\lambda_k}$. This leads to a theoretical justification for clustering using the bottom k eigenvector to embed the vertices. See Section 7.1 for the geometric insight about the segregation into k directions.

3.2.3 Eigenvectors Approximation

Equation (3.2) can be written in the form

$$(A - \lambda I)\mathbf{x} = 0 \tag{3.8}$$

Thus, λ is an eigenvalue of A if and only if Equation (3.8) has a nontrivial solution and the set of solutions to Equation (3.8) is the null space $N(A-\lambda I) \neq \{\mathbf{0}\}$. Equivalently, the nontrivial solution exists if and only if $A - \lambda I$ is singular or that

$$det(A - \lambda I) = 0 \tag{3.9}$$

If the determinant in Equation (3.9) is expanded, a nth-polynomial in the variable λ can be obtained,

$$p(\lambda) = det(A - \lambda I)$$

The roots of this polynomial are the eigenvalues of A. In the case when the exact value of an eigenvalue λ is not known but a close approximation λ' is available, the matrix $\lambda' I - A$ is nonsingular.

Approximate the eigenvectors for the generalized problem $L_G\mathbf{x} = \lambda L_H\mathbf{x}$ requires the approximation of eigenvalues such that the quotients $\mathbf{x}^T L_G\mathbf{x}/\mathbf{x}^T L_H\mathbf{x}$ are close to their exact eigenvalues [8][32]. The fastest known algorithm runs in $O(km\log^2 m)$ time by combining the fast Laplacian linear system solver [28] and a standard power method [16]. In practice, the combinatorial multigrid solver (CMG) [30] is preferred for its empirical $O(m)$ running time. CMG combines the structure and operators of multigrid algorithms with powerful and algebraically sound combinatorial preconditioners based on novel tools from support graph theory. CMG provides an approximate inverse for L_G to be used with the preconditioned eigenvalue solver `LOBPCG`[26].

3.2.4 Network Embedding

Network embedding can be understood as the result of learning the graph representations in some low dimensional vector space while preserving the relationships among vertices such as their induced similarity (or distance) function. By itself,

the adjacency matrix of a network is a version of network embedding. If a row of the adjacency matrix represents a vertex, then the similarity with other vertices is captured by the dot product between itself and other vertex row representations. However, such embeddings are in very high dimensional space and are subject to the curse of dimensionality which provides little to no utility.

The result of network embedding is a set of fixed-length vectors $\ll n$, each of which represent a vertex in the graph. These vectors can then be used in various downstream applications such as link prediction, classification, or other applications of machine learning techniques designed for vectorial data. In semi-supervision, network embeddings are the input to produce separable partitions in the vector space of choice aiming to bring members in the same class close together.

Spectral clustering is a linear method and the only network embedding method offering both a provably fast computation and an advanced theoretical understanding. Recently, spectral clustering has been surpassed by state-of-the-art non-linear network embedding methods whose performance is backed by little to no theoretical understanding. The solutions to non-linear network embedding (beside spectral clustering) can be roughly divided into two categories: random walk based [18][38][52][51][42], and deep neural network based [55].

Spectral Embedding Spectral clustering is often used as a baseline in comparing network embedding methods. As the spectral partitioning algorithm is broadly understood as a relaxation of the discrete clustering problem [10], a number of variants have been proposed [54]. Among those, a variant that leverages the theoretical understanding of eigenvectors geometric properties [32] is summarized here as the prime for non-linear embedding method created via implementing the baseline spectral method boosted by neural network discussed in Chapter 7.

- *Eigenvector Embedding.* Compute the eigenvectors $\mathbf{x}_j$ belonging to the k smallest non-zero eigenvalues of the generalized problem $L_G\mathbf{x} = \lambda D_G\mathbf{x}$, under the constraint that $\mathbf{x}^T\mathbf{d} = 0$. The eigenvectors are also normalized so that $\mathbf{x}_j^T D\mathbf{x}_j = 1$. A graph vertex j is represented by a tuple of the jth values in all k eigenvectors.

- *Radial Projection Embedding.* Divide each embedding dimension by its Euclidean norm to normalize the embedding values onto a unit sphere in $\mathbb{R}^k$.

- *Geometric Partitioning.* Use some unsupervised geometric clustering algorithm on the normalized embeddings. (e.g., k-means, or the provable algorithms presented in [32]) as discussed in Section 3.1.4.

Algorithm 1: Spectral Embedding Computation via Spectral Relaxation of the Generalized Baseline Spectral Problem [10].

Output: $U \in \mathbb{R}^{n\times k}$.
Input: L_G, k, L_H (optional)

(Step 1) *Prepare CL Matrix*
1: **if** $\neg L_H$ **then**
2: $\quad L_H = D - \mathbf{d}\mathbf{d}^T/(\mathbf{d}^T\mathbf{1})$
3: **end if**

(Step 2) *Estimate k smallest eigenvectors*
4: $X \in \mathbb{R}^{n\times k} \leftarrow L_G\mathbf{x} = \lambda L_H\mathbf{x}$

(Step 3) *Compute eigenvector embedding*
5: $\mathbf{u} \leftarrow 1^n$
6: **for** $i = 1 : k$ **do**
7: $\quad \mathbf{x} = X_{:,i}$
8: $\quad \mathbf{x} = \mathbf{x} - (\mathbf{x}^T\mathbf{d}/\mathbf{u}^T\mathbf{d})\mathbf{u}$
9: $\quad \mathbf{x} = \mathbf{x}/\sqrt{\mathbf{x}^T L_H\mathbf{x}}$
10: $\quad U_{:,i} = \mathbf{x}$
11: **end for**

(Step 4) *Compute radial projection embedding*
12: **for** $i = 1 : n$ **do**
13: $\quad l_j = ||U_{j,:}||_2$
14: $\quad U_{j,:} = U_{j,:}/l_j$
15: **end for**

Algorithm 1 details the spectral embedding approach that produces clusters with approximation guarantees for a two-way cut. Step 3.8 ensures $\mathbf{x}^T\mathbf{d} = 0$ following the $\mathbf{d}$-orthogonality requirement derived from the proof of Theorem 3.2.1.

Step 3.9 normalized $\mathbf{x}$ by L_H to make the rows of U concentrate in k different directions following [32]. The radial projection normalizes these row vectors onto the k-dimensional hypersphere, which can be the input for geometric partitioning methods.

3.2.5 Spectral Sparsification

The need to design efficient approximation algorithms drives development in graph sparsification. Its main goal is to approximate a graph by some sparse graph. An early example of graph sparsification is *graph spanners* to solve proximity problems in computational geometry [7]. Graph spanners are sparse graphs which have approximately a (thresholded) same distance between every pair of vertices as the original graph such as the shortest-path distance or within some constant factor obtained systematically.

Motivated by optimization problems in linear algebra and spectral graph theory, Spielman and Teng introduced *spectral sparsification* [48]. A spectral sparsifier is a subgraph of the original whose Laplacian quadratic form is approximately the same as one of the original graph on all vector inputs in $\mathbb{R}^{|V|}$.

It is proven that every graph has some spectral sparsifiers with a nearly-linear number of edges which can be found in nearly-linear time. By viewing the graph as an electrical resistive network, and defining the effective resistance R of a edge as the potential difference that must be applied between the end points of an edge to send one unit of electrical flow through, a graph sparsifier can be computed by sampling edges with probabilities proportional to their effective resistances.

Given two graphs G, H, and L_G denotes the Laplacian matrix of G, the spectral ordering of these two graphs is defined by the relation $\prec$ as

$$G \prec H \iff \mathbf{x}^T L_G \mathbf{x} \leq \mathbf{x}^T L_H \mathbf{x}, \forall \mathbf{x} \in \mathbb{R}^{|V|}$$

Let G associated with an electrical network with link e having conductance w_e, and the effective resistance R_e being the potential difference induced across an edge e when a unit current is injected at one of its end and extracted at the other.

Theorem 3.2.3 (Spectral Sparsification [46])**.** Let H be obtained by sampling edges of G independently with probability $p_e = \Theta(w_e R_e \log n/\epsilon^2)$ for some $\epsilon > 1/\sqrt{n}$ and giving each sampled edges weight $1/p_e$. Then, with high probability

$$(1-\epsilon)G \prec H \prec (1+\epsilon)G$$

Lemma 3.2.1 (Oversampling [29])**.** Let H be obtained by sampling edges of G with probability $p_e \geq c w_e R_e \log n/\epsilon^2$ for some $\epsilon > 1/\sqrt{n}$ and a sufficiently large constant $c > 0$. When giving each sampled edge weight $1/p_e$, with high probability

$$(1-\epsilon)G \prec H \prec (1+\epsilon)G$$

Theorem 3.2.4 (Spectral Sparsification via Random Spanners [24])**.** Let $\{G_{i,j} : 1 \leq i \leq O(\log_{\frac{1}{1-\epsilon}}(\frac{n^4 w_{max}}{w_{min}}), 1 \leq j \leq O(\log^3 n/\epsilon^3)\}$ be a collection of random subgraphs of G, where $G_{i,j}$ is an independent copy of G_p for $p = \frac{1}{w_{min}}(1-\epsilon)^i$. Then there is a weighting of the edges of the subgraph $H = \cup_{i,j} S(G_{i,j})$ such that it is a $(1 \mp \epsilon)$-sparsifier of G. Moreover, such a weighting can be constructed in $\tilde{O}(m)$ time where m is the number of edges in G.

Recent efforts aim for simpler algorithms to achieve spectral sparsifiers of similar results. Inspired by theorem 3.2.4, Koutis [28] showed a closer connection between spanners and spectral sparsification. In order to reduce the number of edges by a **sparsification factor** ρ while being able to preserve the graph spectrum within a $(1+\epsilon/(4log\rho))$ factor, $O(log^2 nlog^2\rho/\epsilon^2)$ edge-disjoint spanners of the graph can be computed in $O(mlog^2 nlog^3\rho/\epsilon^2)$ time. Such computation certifies upper bounds for the effective resistances of the rest of the edges. In comparison to earlier approaches, the development of [28] allows improved efficient construction of spectral sparsifiers

with high approximation quality, the ability to be implemented as a stand-alone sparsification routine, along with the freedom to choose ρ, and a reduction in the dependency of ϵ ($1/\epsilon^2$ vs $1/\epsilon^4$).

Graph Decomposition into High Conductance Clusters Graph sparsification can be understood as the problem of simplifying linear systems. [27] reduces this problem to the same one but in a sparser, tree-like, spanning subgraph via the construction of subgraph preconditioners known as *Steiner* preconditioners [17][30]. *Preconditioned* iterative methods allows a faster convergence when solving linear systems as the ratio of the maximum eigenvalue over the minimum eigenvalue of a matrix is large. With the creation of a new vertex I_i that connects all vertices in a disjoint set S_i, the $[\phi, \rho]$-decomposition gives the sets $\{I_i \cup S_i\}$ where $i = 1, \cdots, m$ conductance values bounded below by ϕ and the vertex reduction factor of P being at least $\rho = n/m$. Since the clusters can be found independently, such decomposition can be computed in $O(\log n)$ time with linear work in planar graphs.

CHAPTER 4

CHEEGER INEQUALITIES FOR SPECTRAL MAXIMIZERS

This chapter explains the anatomy of the spectral maximizers based on multiple modified trees whose external nodes are the set of vertices V of the original graph, and whose internal nodes are add to define the level of the external nodes.

4.1 Graph Decomposition and Cut Approximators

Definition 4.1.1 (Hierarchical Cut Decomposition)**.** Denote I as the set of internal vertices and V as the set of external vertices of a tree $\mathcal{T}$. A hierarchical cut decomposition for a graph $G = (V, E, w)$ is represented as a rooted tree $\mathcal{T} = (V \cup I, E', w')$, with the following properties:

(i) Every vertex u of $\mathcal{T}$ identifies a set $S_u \subseteq V$.

(ii) If r is the root of $\mathcal{T}$ then $S_r = V$.

(iii) If u has children $v_1, \ldots, v_t$ in $\mathcal{T}$, then $S_{v_i} \cap S_{v_j} = \emptyset$ for all (i, j).

(iv) If u is the parent of v in $\mathcal{T}$ then $w'(u, v) = cut_G(v)$.

Definition 4.1.2 (α-Cut Approximator)**.** A hierarchical decomposition $\mathcal{T} = (V \cup I, E', w')$ for G is an α-cut approximator for G if for all $S \subseteq V$ there exists a set $I_S \subseteq I$ such that

$$cut_G(S) \leq cut_{\mathcal{T}}(S \cup I_S) \leq \alpha \cdot cut_G(S).$$

Definition 4.1.3 (Stretch [31])**.** Let p be a path joining the two endpoints of an edge $e \in E$ via traversing an ordered set of e'. The stretch of e over a graph H is the minimum of all stretches $st_p(e)$ of e

$$st_H(e) = \min_{p \in H} st_p(e) \text{ where } st_p(e) = w_e \sum_{e' \in p} st_p(e')$$

Definition 4.1.4 (log$n$-spanner [31])**.** A log$n$-spanner of a graph G is a subgraph H of G such that for all edges $e \in E$,

$$st_H(e) \leq 2\log n$$

Definition 4.1.5 (Spectral Maximizers)**.** Let $\mathcal{T} = (V \cup I, E')$ be a cut approximator for a graph $G = (V, E, w)$ and let

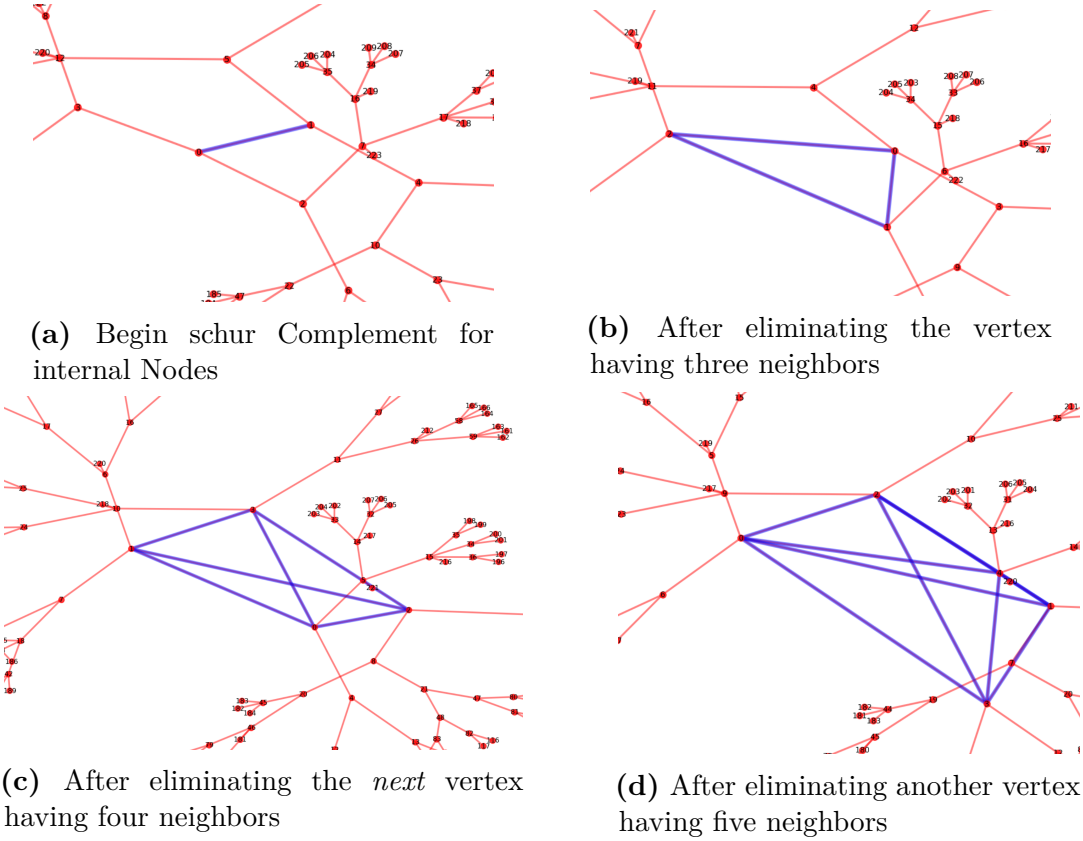
$$L_{\mathcal{T}} = \begin{pmatrix} L_I & V \\ V^T & D \end{pmatrix}$$

ordered so that its first $|I|$ rows are indexed by I in an arbitrary order, and its last $|V^T|$ rows are indexed by V in the given order. H is defined as the graph with Laplacian matrix $L_H = D - V^T L_I^{-1} V$. H is the spectral maximizer of G given $\mathcal{T}$.

The matrix $D - V^T L_I^{-1} V$ in definition 4.1.5 is known as the **Schur complement** with respect to the elimination of the vertices in I. The Schur complement summarizes the interactions between sub components of the graphs. In general, the Schur complement is dense and computational intensive to be solved directly. Given the fact that $L_{\mathcal{T}}$ is Laplacian, D is well known to be a Laplacian matrix (e.g., see [12]). Theoretical graph shows that the elimination of a vertex v from a graph introduces a weighted clique on the neighbors of v. The elimination of the entire set of vertices I can be performed as a sequence of vertex eliminations in any arbitrary order. Figure 4.1 illustrates the gradual elimination of vertices, each step of which form the weighted clique on the neighbors of vertex eliminated.

4.2 Properties of Spanners

One of the key assumptions of this thesis relies on the approximation of a **spectral sparsifier** of $G + H$ where H is the spectral maximizer of G as mentioned in Section 2.1.



(a) Begin schur Complement for internal Nodes

(b) After eliminating the vertex having three neighbors

(c) After eliminating the *next* vertex having four neighbors

(d) After eliminating another vertex having five neighbors

Figure 4.1 A visualization of the elimination process of the internal vertices of I one vertex at a time following Schur complement. An arbitrary root of the internal set of I will be the first to be eliminated. This creates a weighted clique among its direct neighbors. From here, among the vertices of the internal nodes, another vertex is removed and so on.

Spectral sparsifiers are sparse graphs that preserve within an $1+\epsilon$ factor the quadratic form $\mathbf{x}^T L_G \mathbf{x}$ where ϵ is a parameter of choice. Kapralov et al. [24] introduced spanners in the context of spectral graph sparsification, proving the existence of tight upper bounds of the approximate effective resistances on average via Lemma 3.2.1. The extra sampling can compensate for the lack of accuracy in the estimates for the effective resistances. These spanners inspires [31] to compute a small number of edge-disjoint spanners that allow the certification of upper bounds, which in turn enables uniformly sampling-away about half of the remaining edges while spectrally preserving the graph within a $(1+\epsilon/(4\log p))$ factor.

4.3 Properties of Spectral Maximizers

A triple $(G, \mathcal{T}(\alpha), H)$ defines the relationship between a graph G with one of its associated α-cut approximator $\mathcal{T}(\alpha)$, and the spectral maximizer H corresponding to $\mathcal{T}$. $diam(\mathcal{T})$ is used to denote the diameter of the tree $\mathcal{T}$ (i.e., the number of edges on the longest path in $\mathcal{T}$).

Let G and H be two graphs. G is required to be connected. G and H may share 0, some or all vertices between them. G is said to **spectrally dominate** H, if for all vectors $\mathbf{x}$, $\mathcal{R}(L_G, \mathbf{x}) \geq \mathcal{R}(L_H, \mathbf{x})$. The spectral domination of G over H is denoted by $G \succeq H$. $\alpha \cdot G$ denotes the graph G with its edges' weights multiplied by α individually. The $\tilde{O}$ notation is used hide factors logarithmic in n.

Theorem 4.3.1 (Spectral Domination of Cut Structure)**.** Given a triple $(G, \mathcal{T}(\alpha), H)$, let $\tilde{G}$ be an arbitrary graph which is ρ-cut similar to G. Then, $diam(\mathcal{T}) \cdot \rho \cdot H \succeq \tilde{G}$.

Theorem 4.3.2 (Cut similarity of spectral maximizer)**.** Given a triple $(G, \mathcal{T}(\alpha), H)$, the maximizer H is $\alpha \cdot diam(\mathcal{T})$-cut similar with G. In particular, $cut_H(S)/\alpha \leq cut_G(S) \leq diam(\mathcal{T}) cut_H(S)$.

If parameters $diam(\mathcal{T})$ and α are of size $\tilde{O}(1)$, theorem 4.3.1 shows that up to a $\tilde{O}(1)$ factor, H spectrally dominates **every** graph that is $\tilde{O}(1)$-cut similar with G. This directly implies that up to the same $\tilde{O}(1)$ factor, the i^{th} eigenvalue of L_H is greater than that of $L_{\tilde{G}}$, for every graph $\tilde{G}$ which is cut-similar to G. Combined with Theorem 4.3.2, L_H has nearly the maximum possible eigenvalues that any graph with similar cuts can have. In the particular case of λ_2, it is actually within $\tilde{O}(1)$ from the graph conductance. This extends to a generalized notion of conductance with algorithmic implications for **semi-supervised clustering**.

4.4 Cheeger Inequalities for Spectral maximizers

Definition 4.4.1 (Generalized Conductance)**.** Let G_1 and G_2 be two graphs on the same set of vertices V. The generalized conductance $\phi(G_1, G_2)$ of the pair is defined

as:

$$\phi(G_1, G_2) = \min_{S \subseteq V} \frac{cut_{G_1}(S)}{cut_{G_2}(S)}$$

Definition 4.4.2 (Second Generalized Eigenvalue)**.** The smallest generalized eigenvalue of a pair of graphs (G_1, G_2) is given by

$$\lambda_2(G_1, G_2) = \min_{x} \frac{x^T L_{G_1} x}{x^T L_{G_2} x}$$

Let K be a complete weighted graph, where the weight of edge (u, v) is set to be the product of the degrees of u and v in G_1

$$w_K(u, v) = vol_{G_1}(u) vol_{G_1}(v)$$

Also let λ_2 denote the second eigenvalue of the normalized Laplacian of G_1, denoted by $\mathcal{N}$. Degree normalized Laplacian

$$\mathcal{N} = D^{-\frac{1}{2}} L_{G_1} D^{-\frac{1}{2}}$$

where D is the diagonal matrix of the vertex degrees in G_1. It is easy to see that

$$\phi(G_1, K) = \phi(G_1) = min_{S \subseteq V} \frac{cut_{G_1}(S)}{vol_K(S) vol_K(V - S)}$$

and

$$\lambda_2(G_1, K) = \lambda_2$$

Theorem [8] states that $\lambda_2 \geq \phi^2/2$. The generalized conductance is known to have the following relationship following Theorem 3.2.1

$$\lambda_2(G_1, G_2) \geq \phi(G_1, G_2)\phi(G_1)/8$$

The following Theorem is to be proven.

Theorem 4.4.1 (Extended Cheeger Inequality for Cut Structure)**.**
For any graph G, there exists a graph H such that (i) H is $\tilde{O}(1)$-cut similar with G, and (ii) H satisfies the following inequality for all graphs B:

$$\lambda_2(H, B) \leq \phi(H, B) \leq \tilde{O}(1)\lambda_2(H, B).$$

Theorem 4.4.1 implies that the actual accuracy performance of spectral clustering on a given graph G ultimately depends on its 'spectral distance' from its maximizer H. This implication is captured in the following corollary.

Corollary 4.4.1 (Actual Cheeger Inequality)**.** Let G be a graph and H be the graph whose existence is guaranteed by Theorem 4.4.1. Further, suppose that G and H are δ-spectral similar. Then, for all graphs B, G satisfies the following inequality:

$$\lambda_2(G, B) \leq \phi(G, B) \leq \tilde{O}(\delta)\lambda_2(G, B). \tag{4.1}$$

4.5 Proofs

Depending on the context, the notion of any graph G in this section could be understood as the graph itself or its corresponding Laplacian L_G.

Lemma 4.5.1. (Edge-Path Support [6]) Let P be an unweighted path graph on k vertices, with endpoints u_1, u_k. Also let $E_{u_1 u_k}$ be the graph consisting only of the edge (u_1, u_k). Then $kP \succeq E_{u_1 u_k}$.

Lemma 4.5.2 (Quadratic form of Schur complement)**.** Let H and $\mathcal{T}$ be the graphs matrices appearing in the definition of spectral maximizers (definition 4.1.5, this implies

$$\mathcal{R}(H, \mathbf{x}) = \min_{\mathbf{y} \in \mathbb{R}^{|I|}, \mathbf{x} \in \mathbb{R}^{|V|}} \mathcal{R}(\mathcal{T}, \begin{pmatrix} \mathbf{y} \\ \mathbf{x} \end{pmatrix}).$$

The following (adjusted) Lemma from [43, 5] is needed to complete the proof:

Lemma 4.5.3. Every graph G has an $\tilde{O}(1)$ cut-approximator $\mathcal{R}$. The diameter of $\mathcal{T}$ is $O(\log n)$, where n is the number of vertices in G.

The following is the proofs of spectral modification:

Proof. (of Theorem 4.3.1) Below is the proof for the intermediate claim that the product of diameter of $\mathcal{T}$ and $\mathcal{T}$ spectrally dominates G:

$$diam(\mathcal{T}) \cdot \mathcal{T} \succeq G$$

The technique uses elements from support theory [6]. Let E_{uv} be an arbitrary edge of G of weight w_{uv}. Let P_{uv} be the unique path between u and v in $\mathcal{R}$; notice that by definition, the path has length at most $diam(\mathcal{T})$. By construction of $\mathcal{R}$,

$$\mathcal{T} = \sum_{(u,v)\in G} w_{uv} P_{uv}$$

Let $\mathbf{y}$, $\mathbf{x}$ be arbitrary vectors of appropriate dimensions where $\mathbf{y} \in \mathbb{R}^{|I|}$, $\mathbf{x} \in \mathbb{R}^{|V|}$. Set $\mathbf{z} = [\mathbf{y}, \mathbf{x}]^T$.

$$\frac{\mathcal{R}(\mathcal{T}, \mathbf{z})}{\mathcal{R}(G, \mathbf{z})} = \frac{\sum_{(u,v)\in G} w_{uv}\mathcal{R}(P_{uv}, \mathbf{z})}{\sum_{(u,v)\in G} w_{uv}\mathcal{R}(E_{uv}, \mathbf{z})} \geq \min_{(u,v)\in G} \frac{\mathcal{R}(P_{uv}, \mathbf{z})}{\mathcal{R}(E_{uv}, \mathbf{w}A\mathbf{z})} \geq 1/diam(\mathcal{T}) \qquad (4.2)$$

The first inequality of Equation (4.2) is standard for a ratio of sums of positive numbers, and the second inequality is an application of lemma 4.5.1. This proves the intermediate claim. Notice now that since the claim holds for all vectors $\mathbf{z} = [\mathbf{y}, \mathbf{x}]^T$ for arbitrary $\mathbf{y}$, it also holds for vectors where $\mathbf{y}$ is defined as in Lemma 4.5.2. That implies $diam(\mathcal{T}) \cdot H \succeq G$

$$\mathcal{T}(H, \mathbf{x}) \geq \mathcal{T}(G, \mathbf{x})/diam(\mathcal{T})$$

To prove the claim for a G' which is ρ-cut similar to G, the above proof can be repeated if $\mathcal{T}$ is replaced with $\mathcal{T}'$

$$\mathcal{T}' = \sum_{(u,v)\in G} w'_{uv} P_{uv}$$

$\mathcal{T}'$ has the same edges as $\mathcal{T}$ but with different weights. Thus,

$$diam(\mathcal{T}) \cdot \mathcal{T}' \succeq G' \tag{4.3}$$

Notice that $\mathcal{T}'$ keeps the same edges of $\mathcal{T}$ but with different weights. Observe now that if v is a vertex in $\mathcal{T}'$ then the edge to its parent has weight equal to $cut_{G'}(S_v)$, where S_v is the set identified by v according to the definition of the cut approximator. However, by the cut similarity of G and G', it is known that

$$cut_{G'}(S_v) \geq cut_G S_v/\rho$$

It follows that the edges of $\mathcal{T}'$ have weight at most ρ times smaller than their weights in $\mathcal{T}$, which directly implies that $\mathcal{T} \preceq \rho\mathcal{T}'$. Substituting into inequality 4.3 above yields

$$\rho \cdot diam(\mathcal{T}) \cdot \mathcal{T} \succeq G'$$

Then applying lemma 4.5.2 one more time gives the claim. □

Proof. (of Theorem 4.3.2) From Theorem 4.3.1, for all $S \subseteq V$

$$cut_S(G) \leq diam(\mathcal{T}) cut_S(H) \tag{4.4}$$

x_S is defined to be the indicator vector of S with $x_S(v) = 1$ if $v \in S$ and $x_S(v) = 0$ otherwise. For any graph G,

$$\mathcal{R}(G, x_S) = \sum_{u \in S, v \notin S} w_{uv}(x_u - x_v)^2 = cut_G(S)$$

Definition 4.1.2 identifies a set $I_S \subset I$ of internal nodes of T. Construct vector $\mathbf{z} = [\mathbf{y}_{I_S}, \mathbf{x}_S]^T$ where $\mathbf{y}_{I_S}$ is the indicator vector for I_S. This results in,

$$cut_H(S) = \mathcal{R}(H, x_S) \leq \mathcal{R}(\mathcal{T}, z) = cut_{\mathcal{T}}(S \cup I_S) \leq \alpha \cdot cut_G(S) \tag{4.5}$$

where the first inequality comes from lemma 4.5.2 and the second one comes from definition 4.1.2. Combining Equations 4.4 and 4.5, $cut_H(S)/\alpha \leq cut_G(S) \leq diam(\mathcal{T})cut_H(S)$. □

Proof. (of Theorem 4.4.1) Let $(G, \mathcal{T}(\alpha), H)$ be the given triple. Also, let $B = (V, E, w)$ be an arbitrary graph. The first part of the inequality is trivial. Let x be the eigenvector corresponding to the smallest non-zero eigenvalue of the generalized problem $L_H\mathbf{x} = \lambda L_B\mathbf{x}$. Using the standard Courant-Fischer characterization of eigenvalues,

$$\lambda_2(H, B) = \frac{\mathcal{R}(L_H, \mathbf{x})}{\mathcal{R}(L_B, \mathbf{x})} = \frac{\mathcal{R}(L_\mathcal{T}, \mathbf{z})}{\mathcal{R}(L_B, \mathbf{x})}, \tag{4.6}$$

where $\mathbf{z}$ is the extension of $\mathbf{x}$ described in lemma 4.5.2.

For an edge (u, v), let P_{uv} denote the (unique) path connecting u and v in $\mathcal{T}$. Following lemma 4.5.1:

$$\mathcal{R}(L_B, \mathbf{x}) = \sum_{(u,v)\in B} w_{uv}(x_u - x_v)^2 \leq \sum_{(u,v)\in B} w_{uv}\mathcal{R}(L_{P_{uv}}, \mathbf{z}) = \sum_{(u,v)\in B} \mathcal{R}(w_{uv}L_{P_{uv}}, \mathbf{z})$$

Note that the current result is the quadratic form of the graph $\mathcal{T}' = \sum_{(u,v)\in B} w_{uv}P_{uv}$.

Because $\mathcal{T}'$ is a sum of paths on $\mathcal{T}$, it has the same edges with T. Denote by $w_\mathcal{T}(q, q')$ the weight of the edge (q, q') on $\mathcal{T}$, where q' is the parent of q. The inequality in 4.6 can be continued as follows:

$$\lambda_2(H, B) \geq \frac{\mathcal{R}(L_\mathcal{T}, z)}{\mathcal{R}(L_{\mathcal{T}'}, z)} = \frac{\sum_{(q,q')\in\mathcal{T}} w_\mathcal{T}(q, q')(z_q - z_{q'})^2}{\sum_{(q,q')\in\mathcal{T}} w_{\mathcal{T}'}(q, q')(z_q - z_{q'})^2} \geq \min_{q\in\mathcal{T}} \frac{w_\mathcal{T}(q, q')}{w_{\mathcal{T}'}(q, q')} \tag{4.7}$$

If $S_q \subseteq V$ is the set identified by q,

$$w_\mathcal{T}(q, q) = cut_G(S_q) \geq cut_H(S_q)/\alpha,$$

where the inequality comes from Theorem 4.3.2. Observe now that (q, q') appears on $\mathcal{T}'$ exactly on the paths P_{uv} such $u \in S_q$ and $v \in S'_q$. It follows that the edge (q, q')

receives in $\mathcal{T}'$ a total weight equal to the total weight of the edges leaving S_q on B, i.e., $w_{\mathcal{T}'}(q, q') = cut_B(S_q)$.

Further continuing on inequality 4.7 yields

$$\lambda_2(H, B) \geq \min_{q \in \mathcal{T}} \frac{w_{\mathcal{T}}(q, q')}{w_{\mathcal{T}'}(q, q')} \geq \min_{q} \frac{cut_H(S_q)}{\alpha \cdot cut_B(S_q)} \geq \min_{S} \frac{cut_H(S)}{\alpha \cdot cut_B(S)} = \phi(H, B)/\alpha.$$

The Theorem then follows by invoking Lemma 4.5.3 and Theorem 4.3.2. □

CHAPTER 5

A SPECTRAL MODIFICATION ALGORITHM FRAMEWORK

5.1 The Tree Decomposition Spectral Modification Framework

5.1.1 Intuition

Given a path graph P, additional edges via long-range connections can be added to eliminate 'elongated features' without changing its cuts. In addition, the path analysis can be replicated on every graph [44].

Figure 5.1 'Elongated features' of the path graph are eliminated via long-range connections without changing its cuts.

Suppose the 2^{nd} eigenvector $\mathbf{x}_2$ corresponds to a low eigenvalue λ_2 where $\lambda_2 = (\mathbf{x}_2^T L \mathbf{x}_2)/(\mathbf{x}_2^T D \mathbf{x}_2)$. Empirically speaking, trees $\mathcal{T}_j$ that capture 'elongated features' of the graph capture a significant fraction of λ_2. A combination of these trees is cut-similar to G can maximize $\mathbf{x}_2$ in the spectrum.

The fact that $\mathcal{T}$ approximates the cuts of G while it has a logarithmic diameter is a desirable property: λ_2 is within $diameter(\mathcal{T})$ from ϕ. This leads to $O(logn)$ Cheeger gap.

5.1.2 The Tree Decomposition Framework

Given a graph $G = (V, E, w)$, the framework will generate a small set of weighted trees T_j on V systematically. For each T_j, a cut approximator $\mathcal{M}_j$ on V is computed. Each $\mathcal{M}_j$ is spectrally close to M which is close to the maximizer H of G. Each $\mathcal{M}_j$ will share the same set of vertices of V as its external leaves and have its own set of vertices I_j related in a hierarchical way. All $\mathcal{M}_j$ will then be combined with G to

form $\mathcal{M}$. Figure 5.2 shows the increases in the number of internal nodes by fusing the cut estimators to the original vertices. The aim is to approximate modifier M that is the Schur complement of the adjacency matrix of $\mathcal{M}$ by gradually eliminating vertices in I. By doing this, it is conclusive that M is cut-similar to G and closer to maximizer H of G, relative to G itself.

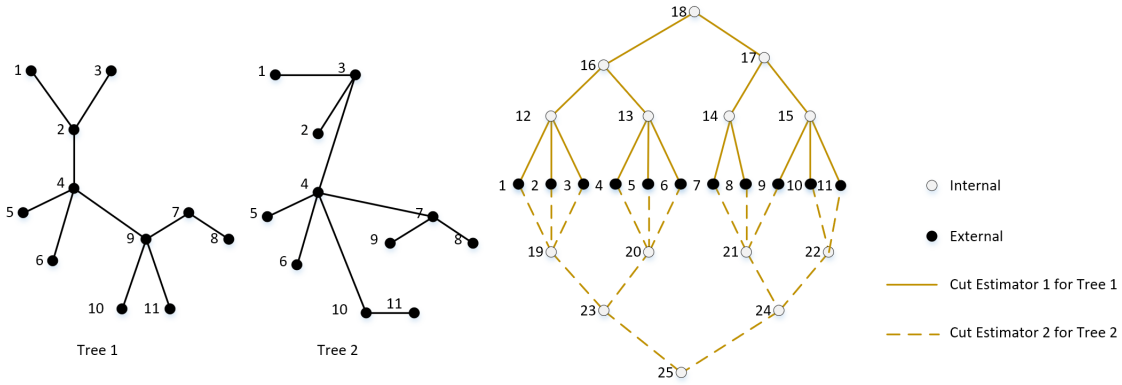


Figure 5.2 Two weighted trees are generated from a graph G. Each tree will be used to derive its own cut estimator. Both cut estimators are 'fused' together via the sharing of the original vertices of G. This graph will be added back to G to form $\mathcal{M}$.

5.2 The First Spectral Modification Algorithm

The goal of the first spectral modification algorithm is not to construct the actual spectral maximizer H of the input graph G but rather, a graph $\mathcal{M}$ that is close to M which is δ-**spectral similar** to H. By directly modifying G while maintaining its spectral structure to arrive at M, this algorithm guarantees the modifier graph $\mathcal{M}$ is cut-similar with the input graph G. Corollary 4.4.1 shows that improved Cheeger inequalities holds (up to δ) for M. The algorithms are heuristic but guarantee to construct M with $\delta = \tilde{O}(1)$.

Algorithm 2 summarizes the four steps involved in performing spectral clustering on an input graph G via spectral modification. Initially, a number of trees is generated from working with the original graph G. These trees are then used to generate their corresponding cut estimators. These cut estimators are then combined with G to

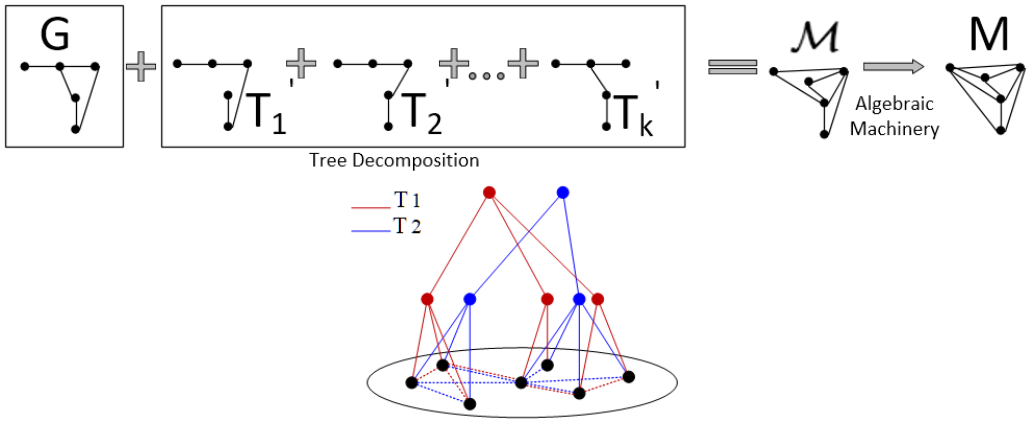


Figure 5.3 **Top**: A small set of weighted trees composed based on the original graph G, each of which is used to compute a cut approximator. The spectral maximizer M is approximated. **Bottom**: A detailed view of the cut approximator for two different trees.

form $\mathcal{M}$. Upon obtaining $\mathcal{M}$, a fast algorithm by solving a linear systems [48, 33] is utilized to obtain M. Finally, spectral clustering on M is performed.

Algorithm 2: Overview of the modified spectral clustering algorithm

---

Output: The approximate maximizer M.
Input: Graph $G(V, E, w)$ where $|E| = m$ and $|V| = n$.

(Step 1) *Compute a small set of spanning trees*

1: Choose (small) $k \leq 2\lceil m/n \rceil$
2: Compute weighted trees $T_1, T_2, ..., T_k$ on V

(Step 2) *Compute cut approximators*

3: **for** $j = 1 : k$ **do**
4: Compute a cut approximator $\mathcal{M}_j$ of T_j
5: **end for**

(Step 3) *Approximate graph spectral maximizer*

6: $\mathcal{M} = \mathcal{G} + \mathcal{M}_1 + ... + \mathcal{M}_k$
7: $M \leftarrow L_{\mathcal{M}}\mathbf{x} = \mathbf{b}$

(Step 4) *Perform baseline spectral clustering on M*

In step 1, the second eigenvector of $L_{G_0}x = \lambda Dx$ is approximated where D is the diagonal of L_{G_0}. $G' = (V, E, w')$ is derived where $w'_{uv} = w_{uv}(\mathbf{z}(u) - \mathbf{z}(v))^2$. From G', a predetermined modified spanning tree structure (MST) denoted R_1. The first tree T_1 is obtained by setting each edge of T_1 (i.e., a copy of R_1) to be equal to the

weight of the corresponding edge in the original graph G. The edge weights of edges corresponding to R_1 is discounted by a discount factor of 2 on G', and continue to find another MST based on this new G'. This process can be iterated up to $2\lceil m/n \rceil$ times to obtain k trees.

In step 2, the cut approximator for each tree T_i is computed. Essentially, two algorithms to achieve these cut approximators are identified. The first algorithm following [43] is a recursive top down analysis of the cut structure of a tree T in $O(nlogn)$ time. The key to this algorithm is the linear runtime for computing the sparsest cut on a **tree**. The costs of cutting a tree at the sparsest cuts are equal to the costs of removing a segment out of the tree. Simply put, this cost when normalized is equal to the total weight of edges that are removed to disjoint the tree into separate connected components over the total weight of the edges within the segment. The other algorithm uses the decompositions from [27], following a bottom-up approach with a faster runtime of O(n).

In step 3, all of the cut approximators are combined into the original graph. This combination forms $\mathcal{M}$, which will then be used in the same manner as in [48, 33] to approximate M being δ-spectral similar to the spectral maximizer H. M is understood to be the Schur complement of $\mathcal{M}$ with respect to the elimination of all extra internal vertices outside V. M does not need to be computed explicitly but by following [48] where linear systems of the form $L_{\mathcal{M}}x = b$ are solved using a fast implementation of the required eigenvector computation.

In step 4, the standard spectral clustering algorithm is run against M.

5.3 Algorithm Justification and Running Time

Step 1 of algorithm 2 is an algebra-based heuristic algorithm that computes k MSTs on a version of G which is transformed based on the MST found in previous iteration. These MST trees used the same weights of the edges in the original G. Assume

that the graph G is spectrally away from its maximizer H, the second eigenvector $\mathbf{z}$ is expected to be 'bad'. Step 3 to seven in algorithm 3 find k trees in G that keep the most"energy" that contributes to the Rayleigh quotient $\mathcal{R}(G, \mathbf{z})$. Adding the maximizers of these trees attempts to increase this Rayleigh quotient higher in the spectrum of the modified graph M. At the same time, because the trees T_i are subtrees of G, their maximizers have similar cuts, the modifier M has cuts similar to those in G.

Algorithm 3: Tree decomposition algorithm

Output: A small set of weighted trees $T_1, T_2, ..., T_k$ on V.
Input: Graph $G(V, E, w)$ and k

(Step 1) *Approximate the* 2^{nd} *smallest eigenvector*

1: $\mathbf{z} \leftarrow L_G x = \lambda D_G x$

(Step 2) *Generate the energy graph* G'

2: $w'_{uv} = w_{uv}(\mathbf{z}(u) - \mathbf{z}(v))^2$
3: $G' = (V, E, w')$

(Step 3) *Energy tree decomposition*

4: $0 < df < 1$ as a discount factor
5: **for** $i = 1 : k$ **do**
6: Generate modified spanning tree $R_i = (V, E_i, w'_{E_i})$
7: $T_i = (V, E_i, w_{E_i})$
8: $w'[E_i] = w'[E_i] \times df$
9: **end for**

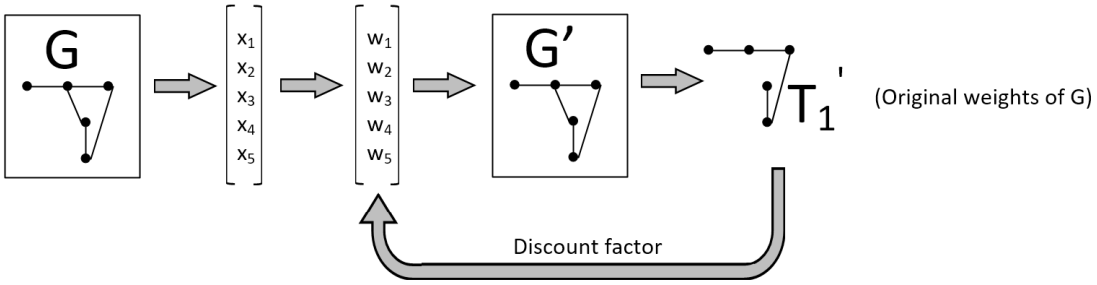


Figure 5.4 Trees that capture a significant fraction Of the 'energy' λ_2.
Note: These trees empirically capture 'elongated features' of G. Maximizing the trees is intended to push the second eigenvector higher in the spectrum.

The modifier M is a dense graph, and only $\mathcal{M}$ which is the implementation following the observation by Spielman and Teng [48] is effectively used, a nearly-

linear time implementation of the required eigenvector computation via inverse power methods to solve near systems of the form $L_M\mathbf{x} = \mathbf{b}$ by solving linear systems of the form $L_{\mathcal{M}}\mathbf{x}' = \mathbf{b}'$ as M is simply the product of Gaussian elimination on $\mathcal{M}$[33]. Specifically, all components of $\mathbf{b}'$ that are mapped to V will have the corresponding values of components of $\mathbf{b}$ on V and be equal to 0 otherwise. Once $\mathbf{x}'$ is finalized, $\mathbf{x}$ is known based on $\mathbf{x}'$. Solving linear systems on $L_{\mathcal{M}}$ can be done in $O(mlogm)$, where m is the number of edges in $\mathcal{M}$ using a fast Laplacian solver [28, 29]. The fastest way as shown in actual experiments is to use the Combinatorial Multigrid (CMG) solver [30]. The worse-case time required for computing the k vectors used in the embedding is at most $O(km\log^2 m)$ employing a standard inversed power method [16], and assuming that the running time of the linear system solver is $O(m\log m)$. In practice the code is much faster due to the faster than worst-case performance of the linear system solver, and to the used preconditioned eigensolver `LOBPCG`[26].

CHAPTER 6

IMPLEMENTATION AND EXPERIMENTS

6.1 Implementation with Different Cut Approximators

Initially, G needs to be re-weighted to $G_{rw} = (V, E_{rw})$ so that the majority of its 'energy' become available to construct a set of trees $T_1, \cdots, T_k$ each of which represents G well. In the first iteration, each node in G_{rw} is embedded by a value of the corresponding component of the 2^{nd} smallest eigenvector $\mathbf{x}$ of G. The squared differences between $\mathbf{x}_u$ and $\mathbf{x}_v$ where u, and v are the end of an edge in G are the energy consumption on the edges in G. Thus, the weight of an edge $(u, v) \in E_{rw}$ is determined by scaling the squared result of the difference between the embedding values of the two end nodes $\mathbf{x}_u$, and $\mathbf{x}_v$ by a factor equals to the original weight of edge $(u, v) \in E$. This yields the first tree structure T_1 whose edges are equal to the original weight as provided in G. All edges of T_1 as appearing in graph G_{rw} will be scaled down by a pre-determined factor < 1 (defaulted to $1/2$). This means G_{rw} has been re-weighted one more time at the edges chosen for T_1 and will be used in the next iteration to find a different tree structure. This process ends after k iterations to yield k trees.

Two choices to structure T_k is chosen to claim its 2^{nd} smallest eigenvalue captures the majority of the 'energy' of the 2^{nd} eigenvalue of G. The current choices of T are the max spanning tree and the low-stretch spanning tree. Two heuristics used to generate the low-stretch spanning trees including a variant of the algorithm of Alon, Karp, Peleg, and West [1] and a variant of the Kruskal's spanning tree algorithm to add tree edges at random with probability proportional to their weights. Overall, the difference of the final results between using the max spanning tree and low-stretch spanning trees are negligible. Below is the detailed implementation to find modified

trees $\mathcal{T}_k$ acting as G cut approximators based on T_k. The goal is to produce balanced cuts, which enables the modified tree $\mathcal{T}$ to have a diameter of logarithmic height.

6.1.1 Cut Approximator using The Top-Down Approach

This method follows cut-based hierarchical decompositions in almost linear time [44] which Räcke developed for oblivious routing schemes but has subsequently been used for a variety of cut-related problems. The hierarchical decomposition tree $\mathcal{T}$ closely approximates the cut-structure of a graph, which is T in our case. The linear tree partition is performed on T to find the cheapest normalized cut(s). These cuts result in a set of clusters of small subtrees. The sum of the weights of the edges required to separate the clusters is then calculated and assigned to the newly created edge that connect such cluster with an internal node. The loop continues until all vertices in G have become the leaf nodes of the tree $\mathcal{T}$.

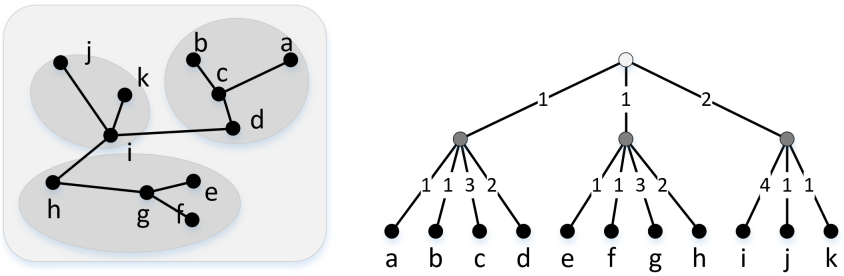


Figure 6.1 Top-down tree decomposition. **Left**: A tree T with each edge of weight 1 exposes the clusters by the top-down tree decomposition algorithm Using the cheapest normalized cut. **Right**: A cut approximator tree $\mathcal{T} = (V \cap I, E_{\mathcal{T}})$ of three levels. The leaf nodes have a one-to-one relation to V. The edge weights correspond to the $cut(S)$ required to separate the set of lower Level leaf nodes from T.

6.1.2 Cut Approximator using the Bottom-up Approach

This decomposition follows the approach of [27] which draws techniques from the construction of preconditioners that use extra *Steiner* vertices. The clustering can be found completely in parallel. From T, form another graph $\hat{T}$ by independently perturbing each edge in T by a random constant. For each vertex in T, keep its

heaviest incident edge as appeared in $\hat{T}$. This turns the tree T into a forest of trees. These trees are known as Steiner trees with asymptotically better condition numbers relative to subgraph preconditioners. Each tree in this forest will then be split independently into clusters of size at most α for some constant α.

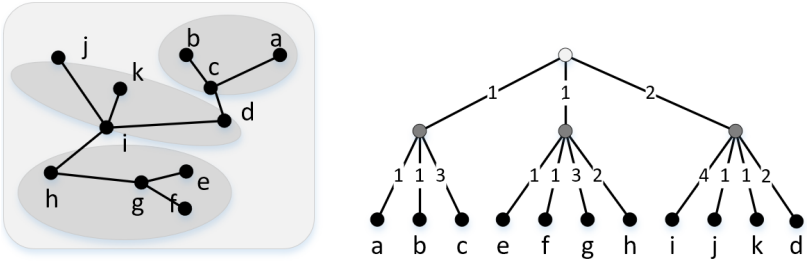


Figure 6.2 Bottom-up tree decomposition. **Left**: A tree T with each edge of weight 1 exposes the clusters by bottom-up tree decomposition algorithm. **Right**: Steiner preconditioner of T in the lowest Level (i.e., star graphs). A cut approximator tree $\mathcal{T} = (V \cap I, E_{\mathcal{T}})$ of three levels. The leaf nodes have an one-to-one relation to V. The sub-trees of this spanning tree is constructed based on the split of a forest of trees formed by perturbing the edges of T.

6.1.3 Cut Approximator Generation Performance

The majority of the running time to modify graph spectrally is spent on the computation of the eigenvector used to construct the trees T in energy decomposition. In addition, the total time to construct $\mathcal{T}$ drastically improves the overall running time. The running times are $O(n)$ for the bottom-up approach and $O(n \log n)$ for the top-down one.

Table 6.1 shows that the bottom-up approach outperforms the top-down approach to generate $\mathcal{T}$ using synthetic data with a growth rate of approximately three times the previous number of vertices and edges. The time difference ratio grows at a faster rate than that of edge growth. This result in the default choice to use the bottom-up approach for finding cut approximators although the top-down approach is a strong contender.

Table 6.1 Running Time to Generate One Cut Approximator Tree $\mathcal{T}$ from an 'Energy' Tree T by Both Top-down and Bottom-up Approaches

$\lvert E_T \rvert$	53	195	599	1,735	5,039	14,223	40,799	114,463	325,313
Bottom-up	0.001	0.002	0.014	0.002	0.006	0.008	0.020	0.049	0.143
Top-down	0.029	0.128	0.453	1.290	4.386	12.733	52.825	216.566	1,545.314
Time Difference Ratio	23.3	76.8	32.9	568.6	700.5	1,538.7	2,603.5	4,443.0	10,806.5

Note: Reported in seconds. The huge running time differences are mostly due to implementation.

6.1.4 Graph Bi-partition in Linear Time Using Spanning Tree

The implementation is to support sweep bi-partition and top-down cheapest normalized cut to form cut estimator in linear time. Algorithm ?? is used to determine the normalized cost ϕ of every bi-partition when dissecting a graph G into two sets of vertices S and $\bar{S}$ based on its corresponding spanning tree T where $S \cup \bar{S} = V$ and $S \cap \bar{S} = \emptyset$.

When a branch of the tree is disconnected, the tree T will form two partitions whose vertices are not overlapped. The algorithm works on every branch of the tree in one scan throughout and return the normalized cut in $O(n)$. As for the input for the graph bisection, an arbitrary root is picked to treat the tree as a directed graph and the degree of every vertex is calculated on the original graph G. Based on depth first search starting from the arbitrary root, a post order traversal is formed and reversed such that the the last vertex to be visited during processing is a vertex before the root in T. The next step is to establish a dictionary to lookup successors of the tree. An appropriate data structure is used to collect children of a vertex since each node may have more than one children. In addition, the predecessors of the vertices are also captured. The raw cost of separating a vertex from the tree is equal to the sum of its degrees in G. The sum of the degrees of all vertices of the total cluster if this edge is to be broken in the tree is set to this raw cost initially. The raw cost of each child of the successors of V_i will be added to the total degree of the cluster. The degrees of

the child and its children are also aggregated as the total degree of the cluster. The normalized cost is the ratio of the raw costs over the total degree of the cluster.

Algorithm 4: Normalized Cut Costs of All Possible Bi-partitions of G in $O(n)$ Based on A Spanning Tree T.

Output: $\mathbf{x} \in \mathbb{R}^{\binom{|V|}{|S|}}$ where $\mathbf{x}_i = \phi(S_i, \bar{S}_i)$
Input: A spanning tree T of G, $\mathbf{d}_G$, $dict_s$, $dict_a$

(Step 1) *Accumulate cut cost*

```
1: for i = 1 : |V| - 1 do
2:     if dict_s[V_i] = 0
3:         x_i = d_i
4:     else
5:         u = dict_s[V_i]
5:         for j = 1 : |u|
6:             x_i = x_i + x_{u_j}
7:             x_i = x_i - 2 Σ_{k∈u} x_k
7:         end for
8:     end if
9: end for
```

(Step 2) *Compute* $min(vol(S), vol(\bar{S}))$

```
10: vol ← d
for i = 1 : |V| - 1 do
11:     u = dict_s[V_i]
12:     for j = 1 : |u|
13:         vol_i = vol_i + Σ_{k∈u} vol_k
14:     end for
15: end for
```

(Step 3) *Compute normalized cost (Element wise)*

```
16: x = x./vol
```

Note: Step 1 and 2 can be combined. The outer loop of both steps processed vertices in post order traversal. $dict_s$: List of successors, $dict_a$: List of ancestor (as in T)

6.2 Establish Performance Benchmark

6.2.1 Baseline Spectral Method

While all other existing clustering algorithms are complicated and far from practical, spectral algorithms possess significant strengths, which are its **speed** and **quality**.

Provably fast linear system solvers for graph Laplacians [28][29] empowers the baseline spectral clustering with even faster speed. A theoretical upper bound for the computation of k eigenvectors is $O(km \log^2 m)$. In practice, for a graph with millions of edges, one eigenvector can be computed in mere seconds on standard hardware. This time would potentially be improved down to milliseconds under parallel programming. The baseline spectral method is implemented based on [10]. By solving the eigenvalue problem $L_G\mathbf{x} = \lambda D\mathbf{x}$, $\mathbf{x}$ is retrieved as the standard embedding. A deviation from the approach of [10] is the further processing of the embedding by projecting the points onto the unit hypersphere as analyzed in [32]. There is a significant improvement in performance in the baseline spectral clustering utilizing this method.

6.2.2 NetMF and Network Embedding

The authors of NetMF algorithm in [42] claimed to discover a theoretical connection and mechanism between recent research in using **deep learning**, **skip-gram**[1] powered networks for network embedding (DeepWalk [38], LINE [52], PTE [51] , node2Vec [18]). Specifically, they made three claims: First, all aforementioned deep learning methods are in theory performing implicit matrix factorizations and have their respective closed forms; Second, some of them are special cases of others [42]; Third, there is a theoretical connection between DeepWalk's implicit matrix and graph Laplacians. Based on the third claim, they proposed NetMF to approximate the closed form of DeepWalk's implicit matrix. Then, they will explicitly factorize this matrix using SVD to obtain a network embedding with (relatively up to 50%) improved prediction performance results compared to ones by DeepWalk and LINE via semi-supervised training.

[1] In Natural Language Processing (NPL), skip-gram model aims to learn continuous feature representations of words by optimizing a neighborhood preserving likelihood objective based on the distributional hypothesis which states that words in similar contexts tend to have similar meanings [20]. The deep learning methods to perform network embedding mentioned here utilize an alike reasoning in term of vertex relations in a graph/network and neighborhood preserving.

Connection between DeepWalk Matrix and Normalized Graph Laplacian Initially, DeepWalk is proven to implicitly approximate and factorize the following closed form matrix

$$\log(\text{vol}(G)(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r)D^{-1}) - \log b \tag{6.1}$$

where $A \in \mathbb{R}_{+}^{|V|\times|V|}$ is the adjacency matrix of a weighted graph G, $D = \text{diag}(d_1, \cdots, d_{|V|})$ where d_i represents generalized degree of vertex i, $\text{vol}_G = \sum_i \sum_j A_{i,j} = \sum_i d_i$ is the volume of G, T & b are the context window size and the number of negative sampling in skip-gram respectively. The connection between Equation (6.1) and the normalized graph Laplacian of G is established by utilizing eigen-decomposition of G to characterize the spectrum of the matrix $\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r)D^{-1}$ as followed:

1. Its eigenvalues is always bounded by the magnitude of $U(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r)U^T$ where $\mathcal{L} = U\Lambda U^T$ such that U orthonormal and $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_{|V|})$

2. Its smallest eigenvalue is bounded by the smallest eigenvalue of $U(\frac{1}{T}\sum_{r=1}^{T}\Lambda^r)U^T$

NetMF Algorithm For large window size, NetMF approximates matrix M by first performing the eigen-decomposition[2] on $D^{-1/2}AD^{-1/2}$ with its top-h eigenpairs $U_h\Lambda_h U_h^T$ then performs a series of matrix multiplications and summations. Once M is approximated, NetMF explicitly aproximates the log of M. This is followed by using SVD to approximate rank-d matrices where $\log(M) = U_d \sum_d V_d^T$ and d is the number of desirable dimensions. Finally, the network embeddings are returned being equal to $U_d\sqrt{\sum_d}$.

[2]NetMF for small window size does not require the approximation based on eigen-decomposition but a direct computation of matrix multiplications.

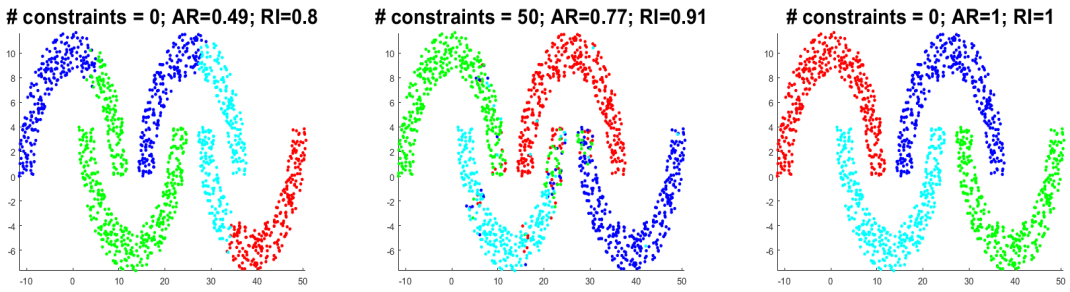
NetMF Weakness In general cases, NetMF makes long range connections by powering up $P = D^{-1}A$. Naturally, there is a connection between the graph Laplacian and the implicit matrix M constructed using P. Even when a good low-rank approximation of M is achieved, the error for approximating $\log(M)$ is bounded by the error bound for the approximation of M. **log** is the only non-linear operator in NetMF. Last but not least, in spite of performing all calculations via approximation, NetMF is still empirically computation expensive, requiring specialized hardware for larger networks such as one for the dataset Flickr of over 80 thousand vertices classified into 195 labels and nearly 5.9 million edges. More detail is discussed in Section 6.3.2.

6.3 Empirical Demonstration of Spectral Modification Gains

6.3.1 Synthetic DataSets

Figure 2.1 demonstrates the ability of spectral modification algorithm that can provide the correct cut that baseline spectral clustering was not able to do. In contrast, NetMF embeddings of 128 dimensions do not perform well when coupled with `k-means`. In binary classification task for the double binary tree, NetMF returns 100% accuracy at around 3% of training labels for embeddings of eight dimensions.

Figure 6.3 shows a synthetic example taken from [10], where spectral modification clearly outperforms even a semi-supervised method.



(a) Baseline spectral **(b)** Semi-supervised [10] **(c)** Spectral modification

Figure 6.3 Comparison of different methods of spectral clustering. The '4-moons' example from [10]. (A)RI is the adjusted rand index.

6.3.2 Social Network Data - Multi-label Classification Task

Four labeled datasets that have been widely used as benchmarks [38] shows the performance of spectral modification. Table 6.2 summarizes their features.

Table 6.2 Dataset Features Including the Second Eigenvalue λ_2

Dataset	BlogCatalog	PPI	Wikipedia	Flickr
$\|V\|$	10,312	3,890	4,777	80,513
$\|E\|$	333,983	76,584	184,812	5,899,882
# Labels	39	50	40	195
λ_2	0.4961	0.4316	0.2001	0.0589

The second eigenvalue λ_2 of these networks is quite high. The theory developed is insensitive to $\tilde{O}(1)$ factors. Nevertheless, the experiments with the implemented version of spectral modification follow exactly the methodology of [42]: First the embeddings are computed and then performed a 10x cross-validation using `LIBLINEAR` [13], at various levels of supervision, for the standard Micro-F1 and Macro-F1 metrics.

The comparison results show:

- The baseline spectral clustering method, implemented with the radial projection step proposed and analyzed in [32][10] (See 1). The step projects the points onto the unit hypersphere, then the points are computed by the standard embedding.
- The NetMF network embedding method [42] which has been shown to perform better than other recent network embedding methods (e.g., DeepWalk [38], LINE [52]).

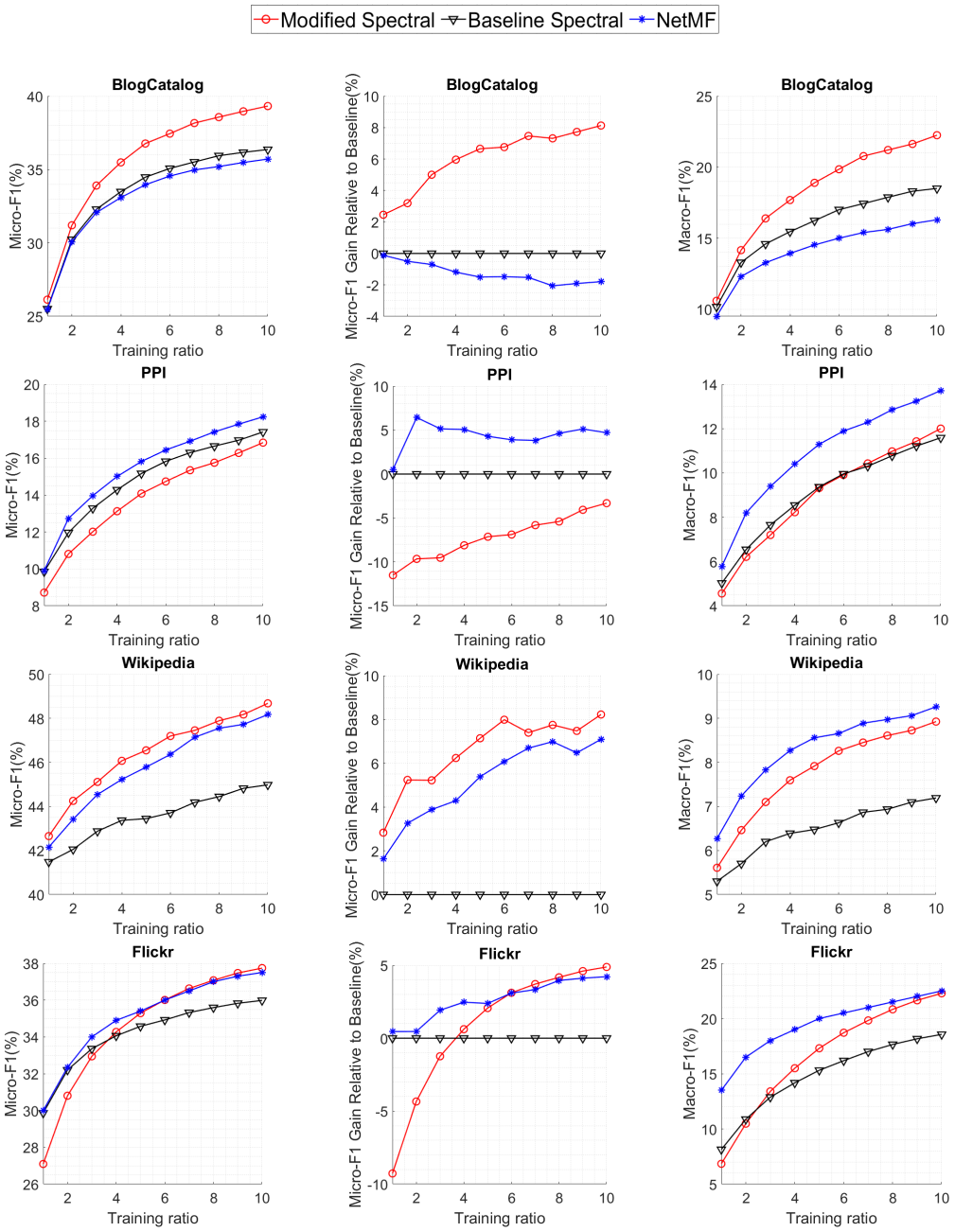


Figure 6.4 Micro-F1 and macro-F1 performance for multi-label classification using `LIBLINEAR` [13] solver (within logistic regression linear Model combined with one-vs-rest classifier)

For each dataset, the dimension of the embeddings is set to be the number of clusters. Hardware of 64GB DDR4 2666MHz RAM encountered *out of memory* error and was not able to complete the NetMF calculation for the embeddings for Flickr dataset at any given dimension. Thus, the Flickr result at 128 dimensional

embeddings were manually approximated as it was reported in [42]. Figure 7.5 summarizes the experiments. Baseline spectral clustering as discussed in Section 6.2.1 performs much better than the version reported in [42].

6.3.3 Social Network Data - Single-label Classification Task

The results produced by spectral modification are from six labeled datasets that have been widely used as benchmarks for the single-label classification task. These datasets have been narrowed down to only include the largest connected component. Data pre-processing includes the removal of self-loop edges, multiple edges, vertices and their edges which are not part of the largest connected component. The number of classes do not change for all datasets after data pre-processing. Table 6.3 summarizes the number of vertices, edges and classes of these largest connected components.

Table 6.3 Number of Vertices and Edges in the Largest Connected Component (*) in Comparison to the Original Data by Dataset

Dataset	CITESEER	CORA	WIKICS	PUBMED	ARXIV	PRODUCTS
$\|V\|$	3,327	2,708	11,701	19,717	169,343	2,449,029
$\|V^*\|$	2,120	2,485	11,311	19,717	169,343	2,385,902
$Difference$	63.72%	91.77%	96.67%	100.00%	100.00%	97.42%
$\|E\|$	4,614	6,632	215,863	44,326	2,315,598	123,718,152
$\|E^*\|$	3,679	5,069	215,554	44,324	2,315,598	123,612,606
$Difference$	79.74%	76.43%	99.86%	100.00%	100.00%	99.91%
# Labels	6	7	10	3	40	47

For each dataset, the dimension of the embeddings is set to be twice the number of unique labels. The training ratio is increased by two times from its previous value for a total of five times starting from 1% to the max of 32%. This setup

help estimate the accuracy performance every time the size of the training data is doubled. To compare to existing benchmark for single-label classification task, the accuracy score calculation is used instead of using micro-F1 and macro-F1 as seen in multi-label classification experiments. The major difference between the accuracy score calculation and the micro-F1 one is that there is only one comparison between expected class and predicted class per sample while micro-F1 allows multiple comparisons for a single sample which expects to belong to more than one class. Since all datasets only have one class per sample, there is no difference between the accuracy score and micro-F1 score.

Table 6.4 reports 5-fold cross-validation results for all datasets. For the CITESEER dataset, spectral network embeddings via spectral modification when coupled with geometric methods produced result not as good as one produced by baseline spectral.

Table 6.4 Single-label Classification with `LIBLINEAR` [13] Solver (Logistic Regression, One-vs-Rest)

	Citeseer						Cora						WikiCS					
Classifier	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Spectral Modified	49.1	51.8	62.8	66.7	67.6	68.9	**64.8**	64.6	70.3	75.0	77.8	**78.8**	67.3	69.7	73.9	75.9	**77.0**	77.3
NetMF	39.4	49.8	53.3	54.5	57.1	60.8	61.2	66.1	69.5	71.6	74.8	76.8	70.3	68.5	68.6	69.5	73.6	**80.7**
Baseline Spectral	**55.3**	**64.4**	**67.8**	**68.7**	69.8	70.9	63.6	**68.8**	**71.5**	74.1	75.0	76.0	**72.8**	**73.0**	74.9	75.4	76.1	76.3
Baseline Spectral (No Regularization)	54.5	56.7	65.0	68.6	**70.4**	**71.1**	64.4	65.1	71.0	**75.7**	**78.2**	78.3	67.5	71.0	**75.0**	**76.2**	76.9	77.0

	Pubmed						Arxiv						Products					
Classifier	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Spectral Modified	**75.7**	**76.4**	**76.6**	**76.7**	**76.8**	**76.8**	55.2	**59.2**	**61.4**	**62.3**	**62.6**	**62.7**	**81.4**	**81.5**	**81.6**	**81.6**	**81.6**	**81.6**
NetMF	**75.7**	76.1	76.5	76.6	76.7	76.7	-	-	-	-	-	-	-	-	-	-	-	-
Baseline Spectral	60.3	60.6	61.7	64.3	67.1	68.8	43.8	45.4	47.3	48.9	50.2	51.4	67.1	69.4	73.4	75.4	76.5	77.3
Baseline Spectral (No Regularization)	73.3	73.4	72.7	72.1	71.6	71.5	**56.9**	58.7	59.2	59.1	59.2	59.2	79.9	79.9	80.0	80.1	80.1	80.1

Note: By training ratio: Highest accuracy score is in bold font. Five-fold cross validation at 20% of training data. For spectral modification, the number of dimensions is equal to only two times the number of classes by dataset. NetMF embedding keeps 128 dimensions by default. "-": Not available/Out-of-memory.

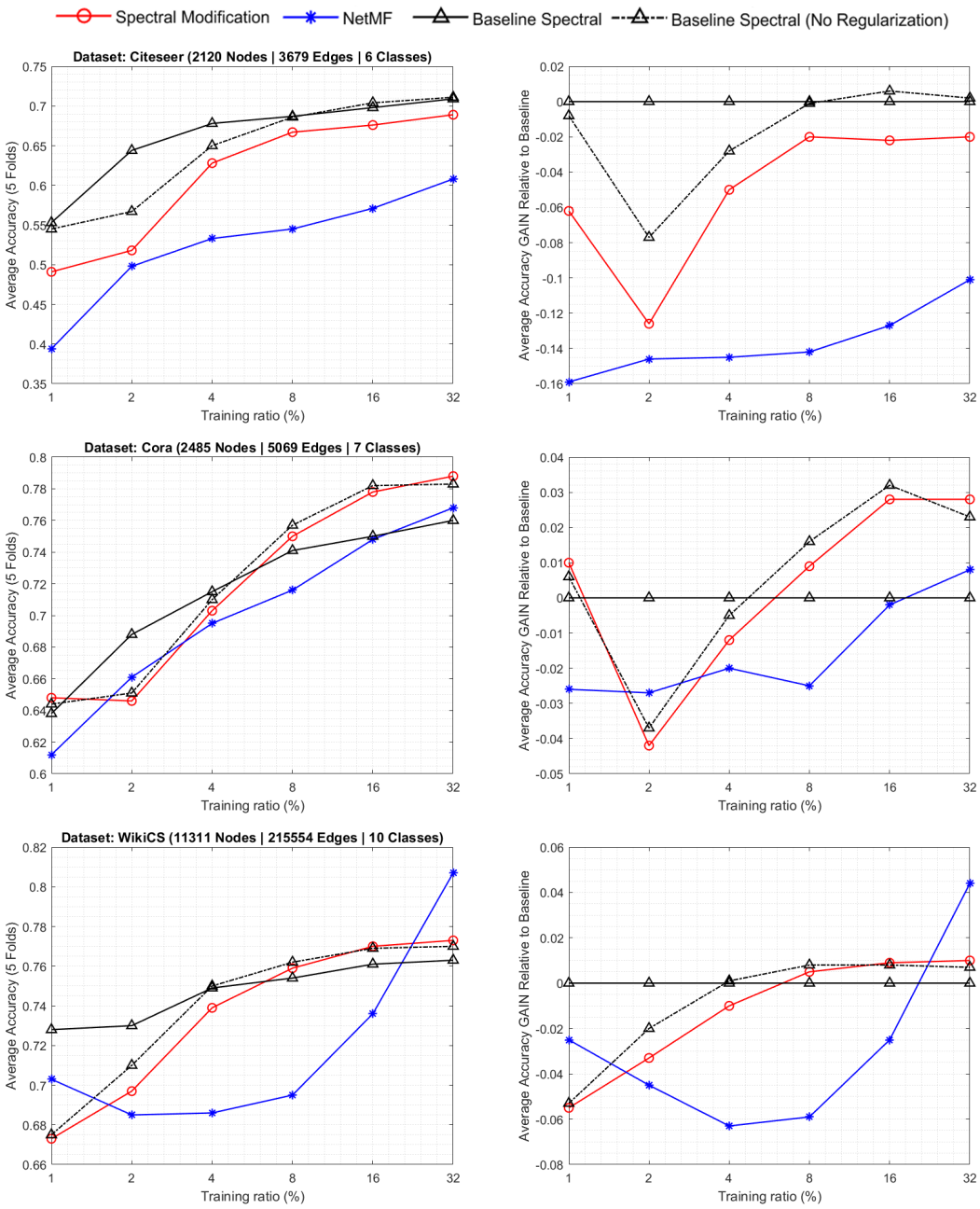


Figure 6.5 Average accuracy score for single-label classification task (to accompany table 6.4 - part one)

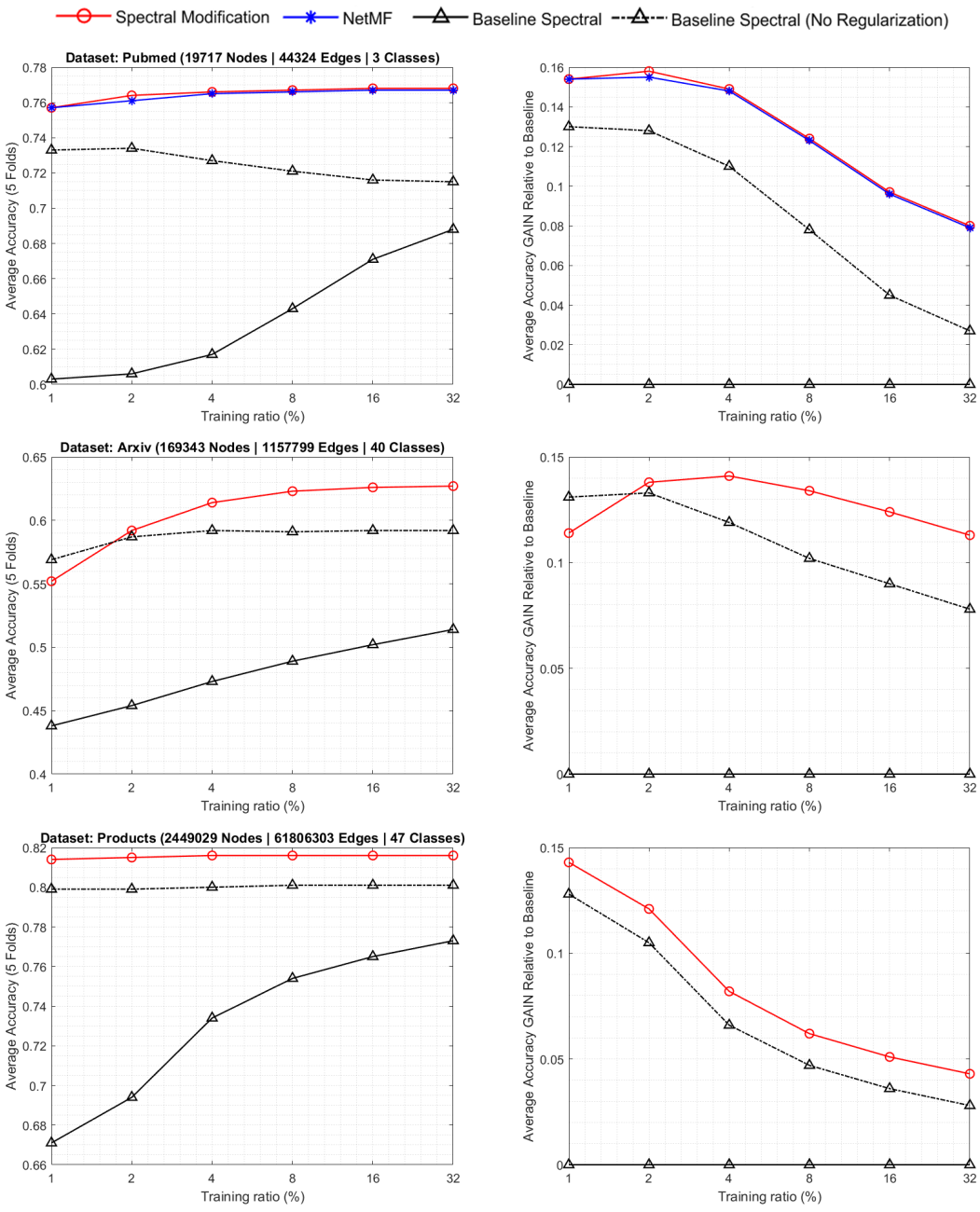


Figure 6.6 Average accuracy score for single-label classification task (to accompany table 6.4 - part two)

CHAPTER 7

SEMI-SUPERVISED SPECTRAL CLUSTERING BOOSTED BY DEEP NEURAL NETWORK

In general, the aim of graph clustering is to embed vertices in a space that distinctly group vertices of the same kind together and away from others. A recent surge in non-linear embedding methods has yielded outstanding accuracy in graph clustering for classification task. However, these come at the expense of increased computationally complexity. In addition, the main drawback of these heuristic methods stay persistent in that there is no (if not very limited) theoretical backing.

Under semi-supervised learning setting, the embeddings happen via the computation of geometric representations of the graph vertices. As aforementioned, spectral partitioning approaches are combinatorial algorithms which partition graphs and matrices based on the computation of eigenvectors of a normalized graph Laplacian. In earlier chapters, eigenvectors as spectral embeddings have been demonstrated their capacity as the strong contender in semi-supervised learning such as the result shown in Table 6.4. This chapter explores the use of spectral embeddings with a thorough understanding of their geometric interpretations to reach state-of-the-art performance in semi-supervised classification tasks. Non-linearity come from processing the baseline spectral embeddings through a procedure call 'Correction and Smooth' implemented by a neural network.

7.1 Geometric Insight for the k Bottom Eigenvectors

The drive behind the connection from eigenvectors to radial projection embedding before geometric partitioning employs the geometric properties of eigenvectors.

Let $X \in \mathbb{R}^{n \times k}$ denote a matrix whose columns are the k eigenvectors. Thus, each row of X is the eigenvector embedding of a vertex. Each column of X is then normalized by its corresponding Euclidean norm following step 4 of Algorithm 1. This projects all embeddings onto a unit sphere. All generalized eigenvectors are D-orthogonal [49], so

$$\mathbf{x}_i^T D \mathbf{x}_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

This yields, $X^T DX = I$, which means $trace(X^T DX) = k$. Furthermore, $trace(X^T DX) = trace(DXX^T)$, thus

$$\sum_{j=1}^{n} d_j ||\mathbf{x}_j||_2^2 = k \tag{7.1}$$

Let $\mathbf{z}$ be a randomly selected unit vector in $\mathbb{R}^k$. $\mathbf{z}$ is in an arbitrary direction. Let $\mathbf{w} = X\mathbf{z}$. Combining $X^T DX = I$, and $\mathbf{z}^T \mathbf{z} = 1$ gives the following equation:

$$\mathbf{z}^T X^T DX \mathbf{z} = \mathbf{w}^T D \mathbf{w} = \sum_{j=1}^{n} \mathbf{d}_j \mathbf{w}_j^2 = 1 \tag{7.2}$$

Equation (7.1) shows that the mass of n embedding points weighted by $\mathbf{d}$ is k. On the other hand, $\mathbf{w}_j^2$ is the norm of the projection of vertex j onto $\mathbf{z}$. Equation (7.2) proves that the $\mathbf{d}$-weighted mass of n embedded vertices projected on $\mathbf{z}$ is always equal to 1, for any direction of $\mathbf{z}$.

A further interpretation of these equations is that any given direction $\mathbf{z}$ captures only $1/k$ fraction of the $\mathbf{d}$-weighted sum of the norms of the n embedding vertices. Therefore, the $\mathbf{d}$-mass of the projections must concentrate in at least k different directions. While finding such directions is not straightforward, [32] proves that radially projecting the eigenvector embeddings allows their concentration in k well-separated spheres in $\mathbb{R}^k$. This specific geometric arrangement facilitates the design of

an unsupervised geometric clustering on the vertices, but it will also guide the design of semi-supervised geometric clustering model.

7.2 Conic Classifier in $\mathbb{R}^k$

The geometric insight of eigenvector embeddings motivates the design of the conic classifier to classify all vertices based on their coordinates on a *single* hypersphere. Since the normalized vertices will concentrate in k directions, neural network is utilized to identify these concentrated areas.

In two dimensional space, conic classifier can be visualized as in Figure 7.1 where all points have the same distance from the origin and each conic regions are dense and well separated. In $\mathbb{R}^k$, there would be k conics that represent how the vertices would concentrate within.

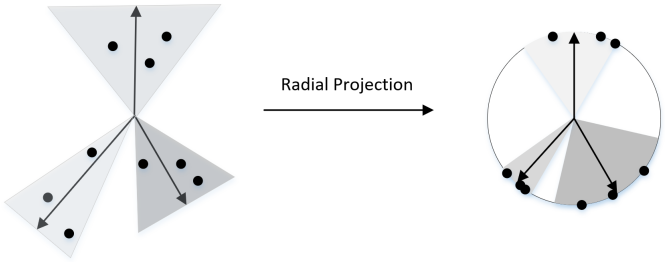


Figure 7.1 **Left:** Spectral embeddings concentrate in three different directions in two dimensional space. **Right:** After radial projection, three unit vectors point in three directions where the normalized eigenvector embeddings concentrate. Each cone is expected to cover the vertices of the same class. The sizes of the angles of the cones at the origin can vary.

Figure 7.2 depicts the main structure of the conic classifier neural network. Network input is the spectral embeddings $E \in \mathbb{R}^{n \times dim}$. dim signifies a large number of bottom eigenvectors such as 200. There can be variants from this setup but these core component remain:

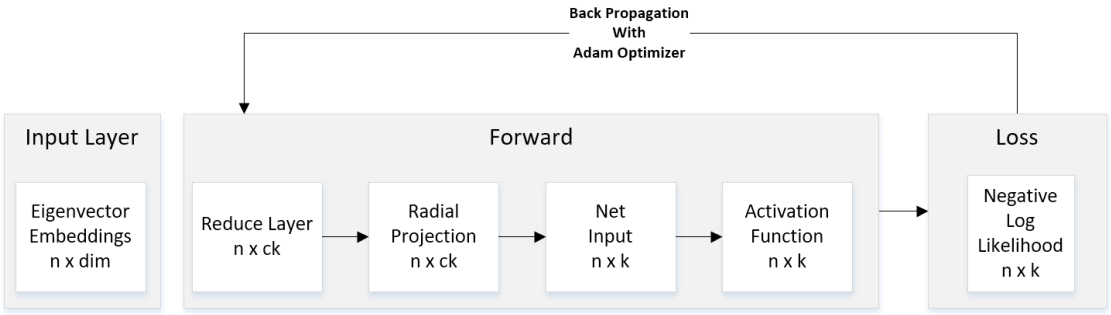


Figure 7.2 The basic anatomy of the conic classifier neural network.

- *The Reduce Layer.* A neural linear layer to reduce the dimensions to a multiple of the number of distinct classes k $\{f(E) : \mathbb{R}^{n \times dim} \rightarrow \mathbb{R}^{n \times ck}\}$.

$$\hat{E} = E \cdot (W^T) + \mathbf{b}$$

- *Radial Projection Function* to normalize all embeddings into a single sphere $\{RP(\hat{E}) : \mathbb{R}^{n \times ck} \rightarrow \mathbb{R}^{n \times ck}\}$.

- *Net Input* z to capture the total of summation between the product of normalized embeddings with W and β as trainable parameters to capture the weights W that put normalized embedded values in their (hopefully correct) territorial cone, and the bias β. While the bias can take any values, it can be the angle whose *cosine* expresses the spread of the cone.

- *Activation Function* ϕ to identify the membership probabilities of a vertex belonging to a certain cone. In reported experiments, ϕ =log$(softmax(z))$.

- *Loss Function* L to calculate the error of the training result for class prediction. In reported experiments, L is the negative log likelihood loss function.

- *Optimizer* to adjust all trainable parameters to minimize the loss. In reported experiments, Adam optimizer is used.

7.3 Correction and Smoothness Algorithms

The embedding process to bring the graph structure down to just points in lower dimensional space is bound to approximation errors, due to labels that 'resist' the metric embedding. This happens as points are getting metrically closer to a cluster that it does not belong to. Figure 7.3 gives an example of a tiny sub-cluster C (or

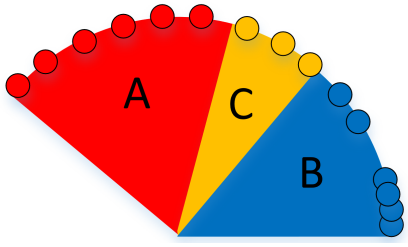


Figure 7.3 Labels resisting metric embedding.

part of it) on ARXIV dataset is a subject in the intersection of two areas A and B and is labeled as a member of B. The metric embedding alone may place C under A. The training labels for B *hopefully* contain some nodes in the sub-cluster C to help correct some neighborhood inside C.

The 'correct and smooth' procedure can be understood as a graph-based correction approach that exploits the correlation in the label structure often referred to as information regularization by label propagation in semi-supervision [50][9][58]. It is likely that some graphical correction of the baseline model prediction will always be a step that yield some improvement.

While graph-based regularization framework is normally expressed as a combination of a loss function and a regularizer, either correction or smoothness procedures are chosen to be implemented as a post-processing step after geometric clustering following [58] [21].

Following the generalized embedding case (Algorithm 1), the normal route of using spectral embeddings includes performing radial projection, then, feeding them into existing geometric algorithms to obtain final class prediction. Empirical practice shows that results by conic network classifiers benefit from the 'correction' in Algorithm 5 and 'smoothness' in Algorithm 6. In addition, other geometric methods also show benefits from adopting these procedure as demonstrated in Section 7.4.

For every vertex, the result of the network prediction is under the form of membership probabilities. Algorithm 5 is used to correct and scale this output

based on its difference from the actual labels by a scale rate α through a number of propagation $iter$. Thus, uncertainties from the training data are propagated across the graph to correct the base prediction. Both α and $iter$ are hyper-parameters.

The aim of smoothing as reflected in Algorithm 6 is to replace all prediction class by the original labels for the training data. Such result is scaled using α and $iter$ as in the case of correction.

Algorithm 5: 'Correction' Procedure to Propagate the Uncertainties of Training Data.

Output: $\hat{Y} \in \mathbb{R}^{n \times k}$

Input: Neural model $\mathcal{M}$, scale rate α, adjacency matrix A, degree vector $\mathbf{d}$, iteration *iter*, train indices *id*

(Step 1) *Normalized Adjacency Matrix*

$D = diag(\mathbf{d}^{-1/2})$
$DAD = D \cdot A \cdot D$
$DDA = D \cdot D \cdot A$
$ADD = A \cdot D \cdot D$

(Step 2) *Elect normalized matrix $\mathcal{N}$ by network type($\mathcal{M}$) and dataset among DAD, DDA, ADD*

(Step 3) *Correct output of $\mathcal{M}$*

$\hat{P} \leftarrow output(\mathcal{M})$ ▷ $P_{i,:} \in \mathbb{R}^{1 \times k}$ membership probabilities
for i = 1 : *iter* **do**
    $\hat{P} = \alpha\mathcal{N} \cdot \hat{P}$
    if *scale outcome correlation* **do**
        $\hat{P} += (1 - \alpha)\hat{P}$
    end if
end for
$\hat{P} = clamp(\hat{P}, -1, 1)$
$R \leftarrow 0^{n \times k}$ ▷ residual for training data
$R[id] = Y[id] - output(\mathcal{M})[id]$
$\mathbf{s} = \sum(|R[id]|)/n / \sum(|\hat{P}_{:,}|)$ ▷ scale factor, column wise
$\mathbf{s}[\infty|\mathbf{s}. > 1000] = 1$
$\hat{P} = output(\mathcal{M}) + \mathbf{s} \cdot \hat{P}$
$\hat{P}[\hat{P}. == nan] = output(\mathcal{M})[\hat{P}. == nan]$

(Step 4) *Assign final predicted values*

$\hat{Y} \leftarrow 0^{n \times k}$
for i = 1 : n **do**
    $\hat{Y}_{i,:}[\textbf{argmax } \hat{P}_{i,:}] = 1$
end for

Note: '.': Element-wise notation

Algorithm 6: 'Smoothness' Procedure by Using Actual Labels of Training Data.

Output: $\hat{Y} \in \mathbb{R}^{n \times k}$
Input: Neural model $\mathcal{M}$, scale rate α, adjacency matrix A, degree vector $\mathbf{d}$, iteration *iter*, train indices *id*, expected labels Y

(Step 1 & 2) See Algorithm 5

(Step 3) *Smooth output of* $\mathcal{M}$

$\hat{P} \leftarrow output(\mathcal{M})$
$\hat{P}[id] = Y[id]$ ▷ expected class for training data
for i $= 1 :$ *iter*
    $\hat{P} = \alpha\mathcal{N} \cdot \hat{P}$
    if *scale outcome correlation* **do**
        $\hat{P} += (1 - \alpha)\hat{P}$
    end if
end for
$\hat{P} = clamp(\hat{P}, 0, 1)$

(Step 4) See Algorithm 5

7.4 Social Network Single Classification Task with Conic

The results shown in Table 7.1 covers a wide range of experiment result with the same benchmark datasets used for single-label classification task under five-fold cross-validation. The training data is combined with a representative set containing one sample per class. In addition, 20% of the training data is used for validation to pick the best model. The detailed setups for all methods are as followed

- Conic. Standard anatomy. $z = RP(\hat{E}) \cdot (W^T) - (||W_{:,i}||_2)cos(\beta)$.
- Linear. $z = RP(\hat{E}) \cdot (W^T) - \beta$.
- Linear NRP. Same setup as the Linear model but without radial projection.
- Knn k choice by validation. Knn with k taking one of the following values 2, 4, 9, 14 depending on validation result.

All other variations are the combination of the main model with either correction or smoothness procedures. The overall performance shows a clear improvement versus the baseline spectral without regularization and NetMF[42].

Table 7.1 Single-label Classification Results for Conic and Other Methods

Classifier	Citeseer						Cora						WikiCS					
	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Conic	45.8	59.6	64.6	67.6	69.7	70.7	16.0	19.0	19.0	22.7	24.4	26.7	69.9	73.2	75.0	75.4	78.2	78.7
Conic +Correction	54.5	63.7	67.5	70.0	71.6	73.7	57.5	65.8	73.3	79.7	82.1	84.2	71.2	74.1	77.1	78.1	80.3	81.3
Conic +Smoothness	55.9	64.3	67.7	70.2	72.1	74.0	64.9	72.2	75.1	**80.1**	**82.7**	84.7	71.1	74.3	**77.8**	78.6	**80.5**	**81.9**
Linear	49.0	63.6	66.9	68.9	68.9	71.1	18.9	18.8	18.8	17.5	25.5	27.4	69.4	71.7	73.9	74.8	75.8	76.2
Linear +Correction	56.8	64.5	68.2	70.1	72.1	73.6	65.5	72.0	74.2	78.1	82.5	84.3	69.4	73.4	76.6	77.7	79.9	81.3
Linear +Smoothness	57.6	**64.9**	68.7	**70.6**	72.2	74.1	69.3	73.6	75.2	77.9	**82.7**	**84.8**	69.2	73.6	76.9	77.6	80.0	81.8
Linear NRP	56.4	50.3	51.0	54.3	57.4	58.4	16.7	23.9	25.7	28.6	26.8	29.0	29.7	34.3	35.0	37.7	35.8	35.9
Linear NRP +Correction	56.9	63.3	68.9	70.3	72.8	**74.2**	64.8	73.6	75.6	78.4	81.1	82.3	42.2	50.0	54.2	59.8	60.5	70.3
Linear NRP +Smoothness	**58.4**	64.5	**69.3**	70.4	**72.9**	**74.2**	69.5	74.2	75.6	78.3	81.4	83.2	52.1	52.2	54.8	57.3	60.0	71.5
Knn (K choice by validation)	57.7	62.6	64.0	68.8	69.9	70.3	61.7	68.4	71.7	73.1	76.4	79.3	71.6	72.9	74.2	76.7	77.7	79.4
Knn (K=10)	49.6	62.2	67.3	68.7	69.8	70.7	50.9	66.9	70.7	74.8	77.3	78.9	70.3	72.7	74.6	76.6	78.5	79.5

Classifier	Citeseer						Cora						WikiCS					
	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Knn (K=10+Correction)	56.6	62.2	64.9	66.8	68.3	70.3	68.3	73.7	75.9	79.4	81.2	82.7	72.4	74.1	75.9	78.0	79.8	80.6
Knn (K=10+Smoothness)	57.8	62.9	65.9	67.8	69.5	72.1	**71.7**	**75.2**	**76.3**	79.9	82.4	84.4	**73.2**	**74.6**	76.4	**78.7**	80.4	81.4
Baseline Spectral	55.3	64.4	67.8	68.7	69.8	70.9	63.6	68.8	71.5	74.1	75.0	76.0	72.8	73.6	74.9	75.4	76.1	76.3
Baseline Spectral (No Regularization)	54.5	56.7	65.0	68.6	70.4	71.1	64.4	65.1	71.0	75.7	78.2	78.3	67.5	71.0	75.0	76.2	76.9	77.0

Classifier	Pubmed						Arxiv						Products					
	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Conic	75.6	75.4	76.0	76.0	76.6	76.6	58.4	60.5	61.3	62.1	62.4	62.5	80.9	80.9	80.9	81.0	-	-
Conic +Correction	75.9	76.9	78.2	79.3	80.0	80.9	59.7	62.2	64.5	66.2	68.6	70.4	84.8	85.7	86.3	86.5	-	-
Conic +Smoothness	**76.9**	77.6	78.8	80.1	81.0	**82.3**	**61.0**	**63.3**	**65.4**	67.0	69.5	**71.3**	**85.6**	**86.2**	**86.8**	87.2	-	-
Linear	63.5	64.6	63.4	64.3	65.2	64.0	47.6	48.5	49.0	49.3	49.3	49.4	66.6	66.7	66.6	66.7	-	-
Linear +Correction	72.8	76.5	79.0	80.2	80.9	81.6	53.5	58.5	63.2	66.4	68.9	70.7	84.6	85.6	86.6	**87.5**	-	-

Classifier	Pubmed						Arxiv						Products					
	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%	1%	2%	4%	8%	16%	32%
Linear +Smoothness	76.0	77.7	**79.4**	**80.5**	**81.4**	**82.3**	56.4	61.0	64.8	**67.4**	**69.6**	**71.3**	84.8	85.7	86.5	87.4	-	-
Linear NRP	53.2	53.6	59.0	55.2	58.8	58.5	21.2	26.5	26.0	24.6	25.8	25.1	27.9	28.7	31.1	28.9	-	-
Linear NRP +Correction	68.7	75.9	78.7	79.9	80.7	81.3	48.3	57.4	62.9	65.8	68.1	69.8	84.6	85.3	86.3	**87.5**	-	-
Linear NRP +Smoothness	76.0	**77.9**	79.1	80.0	80.7	81.5	53.2	59.8	63.7	66.0	68.4	70.4	83.7	83.6	83.8	85.0	-	-
Knn (K choice by validation)	72.6	74.4	74.8	75.7	76.8	77.9	52.9	55.9	58.5	60.3	62.0	63.6	80.4	81.7	82.8	83.8	84.8	85.8
Knn (K=10)	73.5	74.7	75.4	76.3	77.3	78.3	54.7	57.3	59.2	60.9	62.6	64.3	80.7	81.9	83.0	84.1	**85.0**	**85.9**
Knn (K=10+Correction)	74.8	75.5	76.8	78.1	79.2	80.2	57.6	60.3	62.7	64.7	66.4	68.2	-	-	-	-	-	-
Knn (K=10+Smoothness)	76.0	76.6	77.9	79.4	80.7	81.9	59.5	62.4	64.8	66.8	68.7	70.6	-	-	-	-	-	-
Baseline Spectral	60.3	60.6	61.7	64.3	67.1	68.8	43.8	45.4	47.3	48.9	50.2	51.4	67.1	69.4	73.4	75.4	76.5	77.3
Baseline Spectral (No Regularization)	73.3	73.4	72.7	72.1	71.6	71.5	56.9	58.7	59.2	59.1	59.2	59.2	79.9	79.9	80.0	80.1	80.1	80.1

Note: The number of dimensions is equal to twice the number of classes by dataset. By training ratio: Highest accuracy score is in bold font, and top 2 to 3 scores are underlined. Five-fold cross validation at 20% of training data. A representative set of a single member for each class is always part of training data. "-": Not available/Out-of-memory.

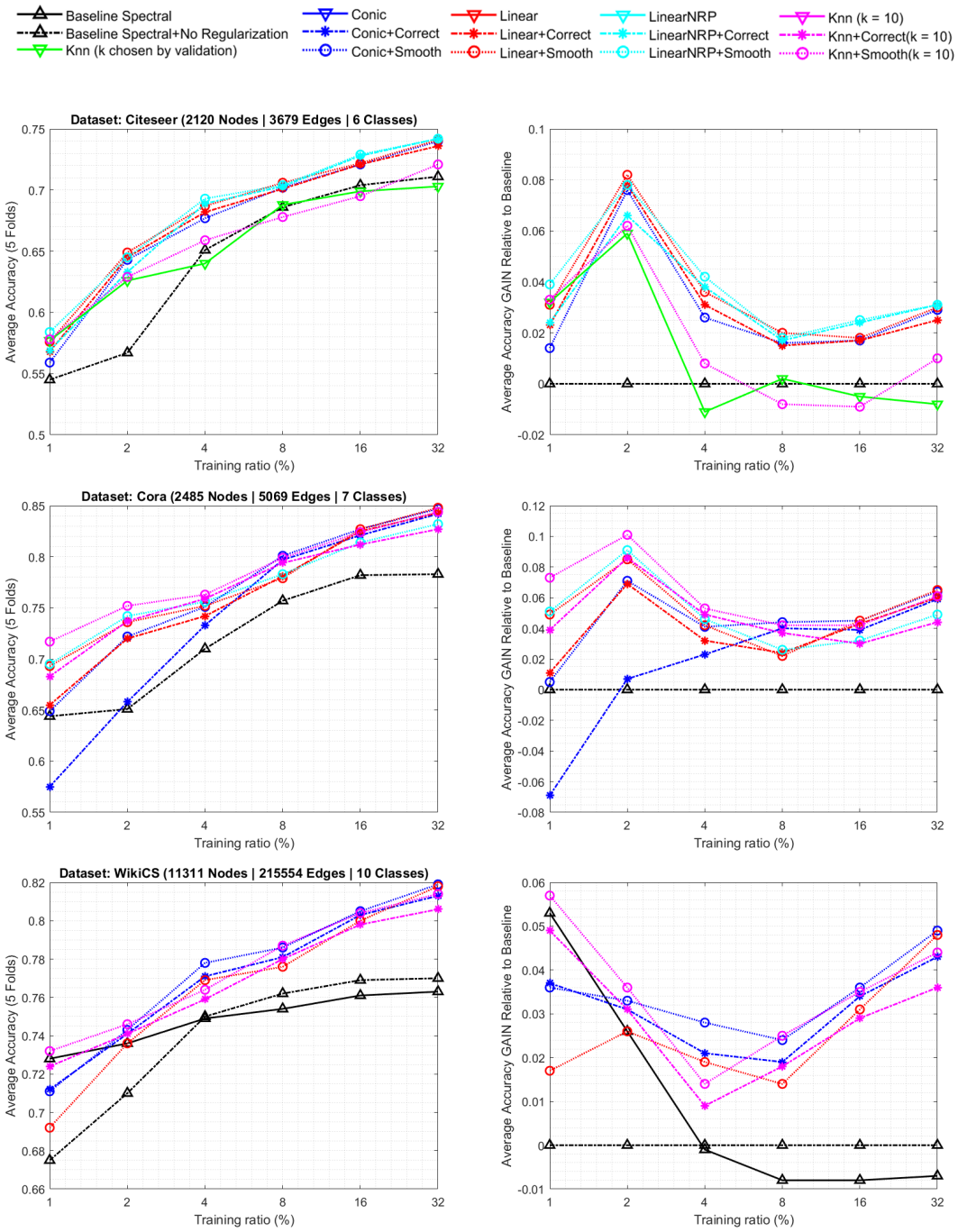


Figure 7.4 Average accuracy score for single-label classification task of the top three methods in any training ratio and baseline spectral without regularization. (to accompany table 7.1 - part one)

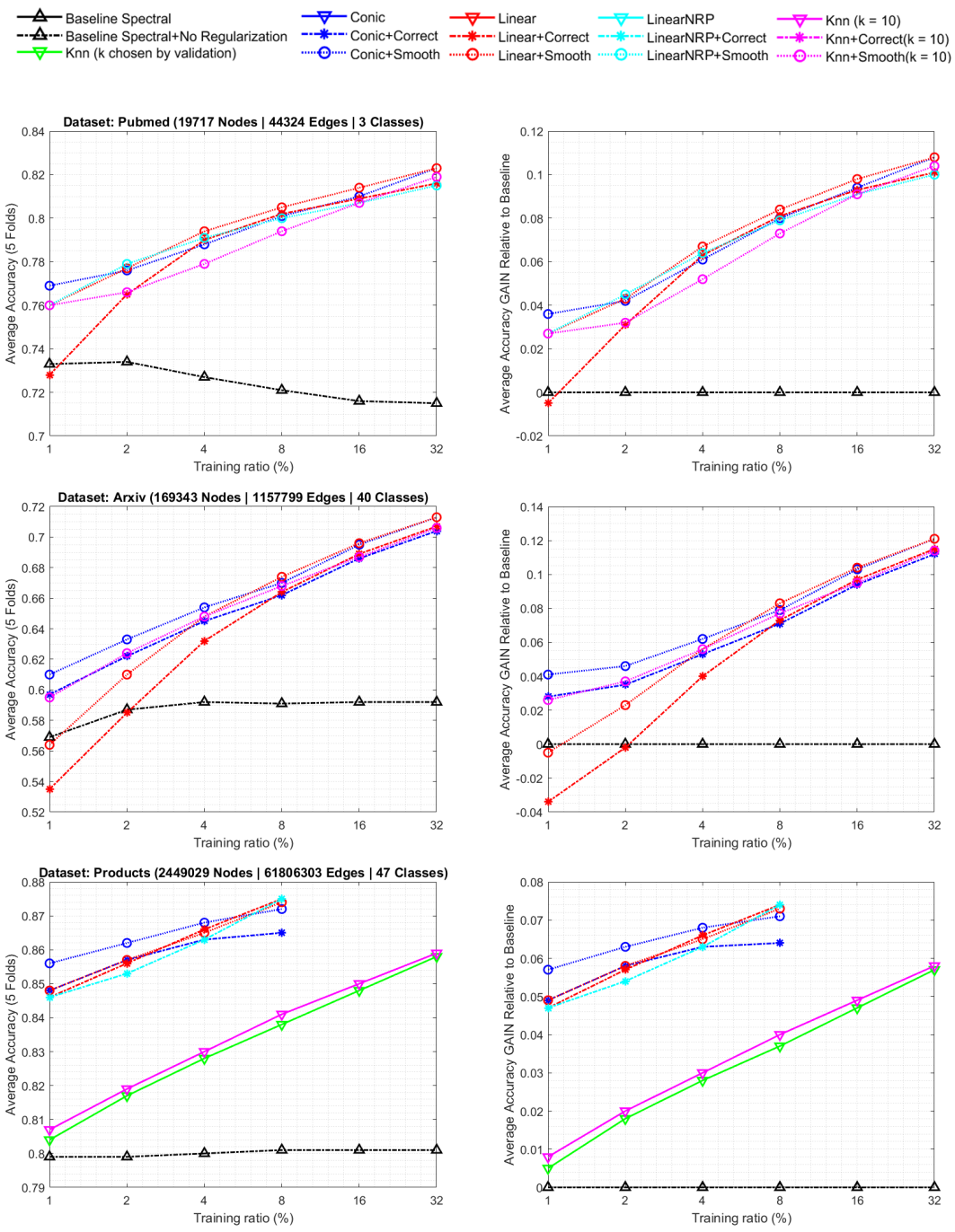


Figure 7.5 Average accuracy score for single-label classification task of the top three methods in any training ratio and baseline spectral without regularization. (to accompany table 7.1 - part two)

REFERENCES

[1] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A Graph-Theoretic Game and its Application to the k-Server Problem. *SIAM Journal on Computing*, pages 78–100, 1995.

[2] Arash A. Amini, Aiyou Chen, Peter J. Bickel, and Elizaveta Levina. Pseudo-likelihood Methods for Community Detection in Large Sparse Networks. *Ann. Statist.*, 41(4):2097–2122, 2013.

[3] Sugato. Basu, Ian Davidson, and Kiri Lou. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Boca Raton, FL: Chapman & Hall/CRC, 1 edition, 2008.

[4] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral Sparsification of Graphs: Theory and Algorithms. *Communications of the ACM*, 56(8):87–94, 8 2013.

[5] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A Practical Algorithm for Constructing Oblivious Routing Schemes. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 24–33, New York, NY, USA, 2003.

[6] Erik G Boman and Bruce Hendrickson. Support Theory for Preconditioning. *SIAM J. Matrix Anal. Appl.*, 25(3):694–717, 2003.

[7] Paul L. Chew. There Are Planar Graphs Almost as Good as the Complete Graph. *Journal of Computer and System Sciences*, (39):205–219, 1989.

[8] Fan Chung. *Spectral Graph Theory*. American Mathematical Society, cbms regio edition, 1997.

[9] Adrian Corduneanu and Tommi Jaakkola. On Information Regularization. *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 151–158, 2003.

[10] Mihai Cucuringu, Ioannis Koutis, Sanjay Chawla, Gary L. Miller, and Richard Peng. Simple and Scalable Constrained Clustering: A Generalized Spectral Method. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, pages 445–454, 2016.

[11] Wilm E. Donath and Alan J. Hoffman. Algorithms for Partitioning Graphs and Computer Logic based on Eigenvectors of Connection Matrices. *IBM Technical Disclosure Bulletin*, 15(3):938–944, 1972.

[12] David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling Random Spanning Trees Faster Than Matrix Multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 730–742, New York, NY, USA, 2017. ACM.

[13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. 9:1871–1874.

[14] Miroslav Fiedler. Algebraic Connectivity of Graphs. Technical Report 98, 1973.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[16] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 3 edition, 1996.

[17] Keith D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, 1996.

[18] Aditya Grover and Jure Leskovec. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM.

[19] Stephen Guattery and Gary L. Miller. On the Quality of Spectral Separators. *SIAM Journal on Matrix Analysis and Applications*, 19(3):701–719, 7 1998.

[20] Zellig S Harris. Distributional Structure. *Distributional Structure, WORD*, 10(3):146–162, 1954.

[21] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. *Computing Research Repository*, 2020.

[22] Antony Joseph and Bin Yu. Impact of Regularization on Spectral Clustering. *Ann. Statist.*, 44(4):1765–1791, 2016.

[23] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings: Good, Bad and Spectral. *Journal of the ACM*, 51(3):497–515, 2004.

[24] Micahel Kapralov and Rina Panigraphy. Spectral Sparsification via Random Spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, volume ITCS'12, pages 393–398, New York, New York, USA, 2012. ACM.

[25] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An Almost-linear-time Algorithm for Approximate Max Flow in Undirected Graphs, and Its Multicommodity Generalizations. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 217–226, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[26] Andrew V. Knyazev. Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[27] Ioannis Koutis and Gary L. Miller. Graph Partitioning into Isolated, High Conductance Clusters: Theory, Computation and Applications to Preconditioning. In *Symposiun on Parallel Algorithms and Architectures (SPAA)*, 2008.

[28] Ioannis Koutis, Gary L. Miller, and Richard Peng. A Nearly-m Log N Time Solver for SDD Linear Systems. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.

[29] Ioannis Koutis, Gary L. Miller, and Richard Peng. A Fast Solver for a Class of Linear Systems. *Communications of the ACM*, 55(10):99–107, 10 2012.

[30] Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial Preconditioners and Multilevel Solvers for Problems in Computer Vision and Image Processing. 115(12):1638–1646.

[31] Ioannis Koutis and Shen Chen Xu. Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification. *ACM Transactions on Parallel Computing*, 3(2):14:1–14:14, 8 2016.

[32] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway Spectral Partitioning and Higher-Order Cheeger Inequalities. *ACM*, 61:37:1–37:30, 2014.

[33] Bruce M. Maggs, Gary L. Miller, Ojas Parekh, R. Ravi, and Shan Leung Maverick Woo. Finding Effective Support-tree Preconditioners. In *Proceedings of the 17th Annual ACM Symposium on Parallel Algorithms*, pages 176–185, 2005.

[34] Hrushikesh N. Mhaskar and Tomaso Poggio. Deep vs. Shallow networks: An Approximation Theory Perspective. *Analysis and Applications*, 14(6):829–848, 2016.

[35] Guido F. Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. In Z Ghahramani, M Welling, C Cortes, N Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[36] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, {NIPS} 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 849–856.

[37] Richard Peng. Approximate Undirected Maximum Flows in O(Mpolylog(N)) Time. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1862–1867, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

[38] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[39] Philipp Petersen and Felix Voigtlaender. Optimal Approximation of Piecewise Smooth Functions using Deep ReLU Neural Networks. *Neural Networks*, 108:296–330, 2018.

[40] Tomaso Poggio, Andrzej Banburski, and Qianli Liao. Theoretical Issues in Deep Networks. *Proceedings of the National Academy of Sciences*, 117(48):30039–30045, 2020.

[41] Tai Qin and Karl Rohe. Regularized Spectral Clustering Under the Degree-Corrected Stochastic Blockmodel. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3120–3128, USA, 2013. Curran Associates Inc.

[42] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Proceedings of the Eleventh {ACM} International Conference on Web Search and Data Mining, {WSDM} 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 459–467, 2018.

[43] Harald Räcke. Minimizing Congestion in General Networks. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 43–52. IEEE, 2002.

[44] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing Cut-based Hierarchical Decompositions in Almost Linear Time. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 227–238, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[45] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 8 2000.

[46] Daniel A. Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances *. Technical report, 2009.

[47] Daniel A. Spielman and Shang-Hua Teng. Spectral Partitioning Works: Planar Graphs and Finite Element Meshes. *Linear Algebra and its Applications*, 421:284–305, 2007.

[48] Daniel A. Spielman and Shang-Hua Teng. Spectral Sparsification of Graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[49] Gilbert W. Stewart and Ji-Guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[50] Martin Szummer and Tommi Jaakkola. Information regularization with partially labeled data. In *Advances in Neural Information Processing Systems*, 2003.

[51] Jian Tang, Meng Qu, and Qiaozhu Mei. PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. *Computing Research Repository*, abs/1508.0, 2015.

[52] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. *Computing Research Repository*, abs/1503.0, 2015.

[53] Luca Trevisan. Graph Partitioning and Expanders Lecture, 2011.

[54] Ulrike von Luxburg. A Tutorial on Spectral Clustering. *Computing Research Repository*, abs/0711.0, 2007.

[55] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural Deep Network Embedding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-Augu, pages 1225–1234, 2016.

[56] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised Deep Embedding for Clustering Analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 478–487. JMLR.org, 2016.

[57] Yilin Zhang and Karl Rohe. Understanding Regularized Spectral Clustering via Graph Conductance. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10631–10640. Curran Associates, Inc., 2018.

[58] Xiaojin Zhu. Semi-Supervised Learning Literature Survey. *European Space Agency, (Special Publication) ESA SP*, 2(604):607–608, 2006.