ABSTRACT

Title of Dissertation: REPRESENTATION LEARNING
FOR LARGE-SCALE GRAPHS

Yu Jin
Doctor of Philosophy, 2023

Dissertation Directed by: Professor Joseph F. JaJa
Department of Electrical and Computer Engineering

Graphs are widely used to model object relationships in real world applications such as biology, neuroscience, and communication networks. Traditional graph analysis tools focus on extracting key graph patterns to characterize graphs which are further used in the downstream tasks such as prediction. Common graph characteristics include global and local graph measurements such as clustering coefficient, global efficiency, characteristic path length, diameter and so on. Many applications often involve high dimensional and large-scale graphs for which existing methods, which rely on small numbers of graph characteristics, cannot be used to efficiently perform graph tasks. A major research challenge is to learn graph representations that can be used to efficiently perform graph tasks as required by a wide range of applications.

In this thesis, we have developed a number of novel methods to tackle the challenges associated with processing or representing large-scale graphs. In the first part, we propose a general graph coarsening framework that maps large graphs into smaller ones while preserving

important structural graph properties. Based on spectral graph theory, we define a novel distance function that measures the differences between graph spectra of the original and coarse graphs. We show that the proposed spectral distance sheds light on the structural differences in the graph coarsening process. In addition, we propose graph coarsening algorithms that aim to minimize the spectral distance, with provably strong bounds. Experiments show that the proposed algorithms outperform previous graph coarsening methods in applications such as graph classification and stochastic block recovery tasks.

In the second part, we propose a new graph neural network paradigm that improves the expressiveness of the best known graph representations. Graph neural network (GNN) models have recently been introduced to solve challenging graph-related problems. Most GNN models follow the message-passing paradigm where node information is propagated through edges, and graph representations are formed by the aggregation of node representations. Despite their successes, message-passing GNN models are limited in terms of their expressive power, which fail to capture basic characteristic properties of graphs. In our work, we represent graphs as the composition of sequence representations. Through the design of sequence sampling and modeling techniques, the proposed graph representations achieve provably powerful expressiveness while maintaining permutation invariance. Empirical results show that the proposed model achieves superior results in real-world graph classification tasks.

In the third part, we develop a fast implementation of spectral clustering methods on CPU-GPU platforms. Spectral clustering is one of the most popular graph clustering algorithms which achieved state-of-the art performance in a wide range of applications. However, existing implementations in commonly used software platforms such as Matlab and Python do not scale

well for many of the emerging Big Data applications. We present a fast implementation of the spectral clustering algorithm on a CPU-GPU heterogeneous platform. Our implementation takes advantage of the computational power of the multi-core CPU and the massive multithreading capabilities of GPUs. We show that the new implementation achieved significantly accelerated computation speeds compared with previous implementations on a wide range of tasks.

In the fourth part, we study structural brain networks derived from Diffusion Tensor Imaging (DTI) data. The processing of DTI data coupled with the use of modern tractographic methods reveal white matter fiber connectivity at a relatively high resolution; this allows us to model the brain as a structural network which encodes pairwise connectivity strengths between brain voxels. We have developed an iterative method to delineate the brain cortex into fine-grained connectivity-based brain parcellations. This allows to map the initial large-scale brain network into a relatively small weighted graph that preserves the essential structural connectivity information. We show that graph representations based on the brain networks from new brain parcellations are more powerful in discriminating between different populations groups, compared with existing brain parcellations.

REPRESENTATION LEARNING FOR LARGE-SCALE GRAPHS

by

Yu Jin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Joseph F. JaJa, Chair/Advisor
Professor Jonathan Simon
Professor Behtash Babadi
Professor Rajeev Barua
Professor Luiz Pessoa, Dean's Representative

Acknowledgments

I would like to express my most sincere gratitude to Professor Joseph JaJa for his mentorship and guidance over the long journey. His wisdom and depth of knowledge has influenced me as a better researcher and a better person. Under his trust and encouragement, I am able to freely pursue a wide range of challenging research topics. I am grateful for his patience and flexibility that supported me through the most puzzling years.

I want to thank Dr. Andreas Loukas who supported me to extend the work on graph coarsening. His enthusiasm and persistence in graph research has influenced me to persistently tackle challenging and fundamental research problems.

I would like to thank the committee members Professor Jonathan Simon, Professor Behtash Babadi, Professor Rajeev Barua and Professor Luiz Pessoa for providing valuable feedback on the thesis and presentation.

I want to thank the department of ECE for fostering a diverse environment to learn about cutting-edge technologies from different domains.

Lastly, I want to thank my wife, Yun Zhou and my parents Wu Jin and Liumei Zhou for their love and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BLAS	Basic Linear Algebra Subprograms
BSR	Block Compressed Sparse Row Format
CIN	Cell Isomorphism Network
CNN	Convolutional Neural Network
COO	Coordinate Format
CPU	Central Processing Unit
CSC	Compressed Sparse Column Format
CSR	Compressed Sparse Row Format
DGCNN	Deep Graph Convolutional Neural Network
dMRI	Diffusion Magnetic Resonance Imaging
DTI	Diffusion Tensor Imaging
EM	Edge Matching
FC	Functional Connectivity
fMRI	Functional Magnetic Resonance Imaging
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
IPCC	Human Inferior Parietal Cortex Complex
IRAM	Implicitly Restarted Arnoldi Method
LAPACK	Linear Algebra PACKage
LSTM	Long Short-term Memory Networks
LV	Local Variation
MDL	Minimum Description Length
MGC	Multilevel Graph Coarsening
MKL	Math Kernel Library
MNI	Montreal Neurological Institute
MPI	Message Passing Interface
MPNN	Message Passing Graph Neural Network
MRI	Magnetic Resonance Imaging
NC	Normal Control
Ncut	Normalized Cut
NetLSD	Network Laplacian Spectral Descriptor

NMI	Normalized Mutual Information
PCIe	Peripheral Component Interconnect Express
PNA	Principal Neighbourhood Aggregation
PPGN	Provably Powerful Graph Network
ROI	Region of Interest
RNN	Recurrent Neural Network
SC	Spectral Clustering
SD	Spectral Distance
SeqGNN	Graph neural network with sequence modeling
SGC	Spectral Graph Coarsening
SGD	Stochastic Gradient Descent
SIMD	Single Instruction, Multiple Data
SIN	Simplicial Isomorphism Network
SM	Streaming Multiprocessors
SP	Streaming Processors
SZ	Schizophrenia
TPJ	Temporoparietal Junction Area
WL	Weisfeiler Lehman Isomorphism Test

Chapter 1: Introduction

1.1 Motivation

Graphs are widely used to model object relationships in real-world applications. There have been extensive studies that propose the use of various graph features to help analyze and understand the complex structures in graphs over a number of domains. Most of the proposed graph features are based on the statistics of elementary graph structures such as paths, walks, trees, and cuts, which are then used for subsequent machine learning tasks. However, the pre-defined, hand-crafted graph features are not flexible enough to handle graph prediction tasks for a wide range applications. The computations of many graph features do not scale well with the graph size, which causes high computational costs especially for the analysis of large-scale graphs. In this introduction, we describe the main problems addressed in this thesis.

Graph Coarsening. To analyze large size graphs, a common technique is to coarsen the graph into a smaller graph that reduces the computational load while attempting to maintain the important structural graph properties of the original graph. The coarsening process significantly reduces the dimensionality of the original large graphs and accelerates subsequent graph processing tasks. Graph statistics such as *characteristic path length*, *global efficiency*, *clustering coefficient* can be efficiently computed on the coarsened graphs. Although various graph coarsening methods have been proposed, there have been no overall consensus on the criteria for the graph coarsening,

which directly determines the statistical effectiveness of subsequent network analysis.

Spectral Methods. Spectral methods are emerging as powerful tools in machine learning and network science. Graph spectra are represented by the eigenvalues and eigenvectors of matrices associated with the graph such as the adjacency and Laplacian matrices. The field of spectral graph theory has established a number of interesting results linking the fundamental relationship between graph structural properties and graph spectra. In particular related work shows that the graph spectra contains rich information of important graph characteristics such as connectivity, bipartiteness, the number of connected components, and diameter and average path length [3]. We introduce a novel framework in this thesis that links tightly graph coarsening and spectral graph theory.

Spectral Clustering Algorithm. There are a number of algorithms which include the computation of the graph spectrum as one of the major computational components to solve problems in a wide variety of applications. An example is the spectral clustering algorithm, which is one of the most popular clustering algorithms aiming to find densely-connected clusters within graphs. Note that graph spectral features, in addition to the common network statistics, are used in graph representations, which can achieve good prediction capability.

A significant hurdle for using graph spectra in applications with large-scale graphs is the computational cost to compute the whole set of eigenvalues and eigenvectors. Graphics Processing Units (GPUs) are well-known in their parallel processing capabilities, which can accelerate the computation of large-scale matrix computations. We provide in this thesis a GPU accelerated implementation to accelerate the computation of the spectra, which significantly outperforms previous implementations.

Graph Neural Networks. Deep learning models have achieved great successes in a wide

range of applications including computer vision, natural language processing, audio processing, and many others. Recently, deep learning models have been introduced to solve graph-related tasks such as node classification and graph classification. A notable model is the *graph neural network* (GNN) model which learns the node and graph representations to solve important graph tasks. Most of GNN models follow a message-passing paradigm where node representations are formed by iteratively aggregating information from neighboring nodes, followed by a graph pooling function to generate graph representations [4, 5, 6, 7]. Despite the success of message passing GNN models in many graph-related tasks, recent studies have shown that GNN models have limited expressive power which fails to capture even simple graph characteristics. For example, GNN graph representations cannot distinguish any pair of regular graphs. In this thesis, we develop a new approach that achieves more powerful expressiveness while maintaining permutation invariance.

Structural Brain Networks. An important investigation conducted in this thesis is the study of large-scale brain networks derived from Diffusion Tensor Imaging (DTI) data. The objective is to learn characteristic graph representations which can achieve high prediction performance in applications involving for example brain disorders. Diffusion Magnetic Resonance Imaging (MRI) exploits the anisotropic diffusion of water molecules in the brain to enable the estimation of the brain's anatomical fiber tracts at a relatively high resolution. Tractographic methods can be used to generate whole-brain anatomical connectivity matrix where each matrix element provides an estimate of the connectivity strength between the corresponding voxels. The resulting brain connectivity graphs are quite large involving hundreds of thousands of nodes and billions of edges. A common method to address this challenge is to build structural brain networks using a predefined whole brain parcellations (defining Regions of Interest - ROIs), where the nodes of

the network represent the brain regions and the edge weights capture the connectivity strengths between the corresponding brain regions. Traditional methods use the existing anatomical brain atlases such as Automated Anatomical Labelling (AAL) [8] to build structural brain networks, which do not consider the underlying connectivity information. There is potential to develop more effective brain parcellations using structural information derived from the DTI data.

1.2 Contributions of this Thesis

In this thesis, we make significant, original contributions to all the problems mentioned above.

In the first part, we propose an effective graph coarsening method which coarsens large graphs while provably preserving some important graph structural properties. Graph coarsening is a common technique used to simplify complex graphs by reducing the number of nodes and edges while trying to preserve some of the main structural properties of the original graph. A key challenge is to determine what specific graph properties are to be preserved by coarse graphs. We propose a new graph coarsening method that attempts to preserve the graph spectral properties. We start by relating the spectrum of the original graph to the coarsened graph. Based on this insight, we define a novel distance function that measures the differences between graph spectra of the original and coarsened graphs. We show that the proposed spectral distance captures in a significant way the structural differences during the graph coarsening process. We also propose graph coarsening algorithms that aim to minimize the spectral distance. Experiments show that the proposed algorithms can outperform previous graph coarsening methods in applications such as graph classification and stochastic block recovery tasks.

In the second part, we propose a new graph neural network paradigm that improves the expressiveness of graph representations. We represent graphs as the weighted combination of sequence representations. We show that, through the design of sequence sampling and modeling techniques, the proposed graph representations achieve provably powerful expressiveness while maintaining the permutation invariance property. Empirical result show that the proposed graph representations achieve superior performance in real-world graph classification tasks.

The third part focuses on the acceleration of the spectral clustering methods leveraging on the computational advantages of CPUs and GPUs. Spectral clustering is one of the most effective graph clustering algorithms which are widely used in a number of applications. However, existing implementations in commonly used software platforms such as Matlab and Python do not scale well for many of the emerging Big Data applications. We present a fast implementation of the spectral clustering algorithm on a CPU-GPU heterogeneous platform. Our implementation takes advantage of the computational power of the multi-core CPU and the massive multithreading capabilities of GPUs. We show that the new implementation achieved significantly accelerated computation speed compared with previous implementations in a number of important applications.

The fourth part focuses on the study of structural brain networks with the objective to determine graph representations based on DTI data. We develop an iterative method to delineate the brain cortex into fine-grained connectivity-based brain parcellations. This allows the mapping of the initial large-scale brain network into a relatively small weighted graph, which preserves the essential structural connectivity information. We show that graph representation based on the brain networks from the new brain parcellations is more powerful in discriminating between a number of population groups.

1.3 Outline of Thesis

The rest of the thesis is organized as follows. In Chapter 2, we give the background of the research work conducted in this thesis. In Chapter 3, we propose a new graph coarsening method which coarsens large graphs into smaller ones with preserved graph spectral properties. In Chapter 4, we propose a new graph neural network model that improves the expressiveness of graph representations. In Chapter 5, we present a fast implementation of spectral clustering for large-scale graphs. In Chapter 6, we study the graph representations of structural brain network while we conclude in Chapter 7 by discussing several potential directions for future research.

Chapter 2: Background

2.1 Preliminaries

We denote the graph as $\mathcal{G} = (\mathcal{V}, \boldsymbol{W}, \boldsymbol{H})$, with $\mathcal{V}$ as the set of graph nodes with $n = |\mathcal{V}|$, $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ as the (possibly weighted) adjacency matrix and $\boldsymbol{H} \in \mathbb{R}^{n \times d}$ as the matrix holding the node features. We denote by $v_i \in \mathcal{V}$ as the node indexed by i, $\boldsymbol{w}(i) \in \mathbb{R}^n$ as the vector of all possible edge weights associated with v_i and $d(i) = \sum_{j=1}^{n} \boldsymbol{W}(i, j)$ as the node degree.

In addition, we use $\boldsymbol{h}_i$ as the node representation of node v_i (corresponding to the ith row of $\boldsymbol{H}$) and $\boldsymbol{h}_{\mathcal{G}}$ as the graph-level representation.

2.2 Graph Coarsening

2.2.1 Overview

Graph coarsening is a common graph processing technique widely used in a number of applications involving large scale graphs. Graph coarsening simplifies large graphs by reducing the number of nodes and edges while trying to preserve some of the main properties of the original graph. The goal is to accelerate graph processing while maintaining a similar performance on graph processing tasks.

One of the key challenges is to define the coarsening criteria based on which the graph

nodes and edges are reduced. The coarsening criteria directly determine the coarsening process and the quality of the resulting coarsened graphs. Researchers have proposed a number of different coarsening criteria, resulting in the optimization to preserve certain graph properties. For example, Loukas et al. proposed to preserve the action of the graph Laplacian with respect to an (eigen)-space of fixed dimension, arguing that this suffices to capture the global properties of graph relevant to partitioning and spectral clustering [9]. Durfee et al. proposed to preserve the all-pairs effective resistance [10] while Garg and Jaakkola defined a cost based on the theory of optimal transport [11]. Saket et al. suggested a Minimum Description Length (MDL) principle relevant to unweighted graphs [12]. Most of previous graph coarsening methods are specific to particular applications; the question of how to define an application-independent graph coarsening framework remains a challenge.

2.2.2 Graph Coarsening Methods

There are a number of graph coarsening methods which are designed to optimize for different graph properties some of which are summarized below.

Heavy Edge Matching In this method, the graph nodes are collapsed based on the *heaviest edge* (i.e., the edge with the largest weight) connecting them. The process is repeated iteratively until a certain graph size is reached [13].

Algebraic Multigrid Coarsening This method is based on the concept of the multigrid method used in numerical linear algebra. It involves constructing a hierarchy of coarser graphs from the original graph using matrix operations, which makes it possible to solve linear systems efficiently [14].

Community Detection-based Coarsening The method first clusters graph nodes forming densely-connected communities and merges graph nodes belonging to the same community into one single node. The method is particularly useful to find community structures within graphs [15].

2.2.3 Applications

The following are examples of applications where graph coarsening methods have been used.

Graph partitioning Graph coarsening are often used as a preprocessing step in graph partitioning, which aims to divide a graph into smaller subgraphs while minimizing some cost function related to the number of edges cut. The coarsened graph can be partitioned more efficiently, and the resulting partition can then be projected back onto the original graph.

Community Detection Graph coarsening can be used to find communities or clusters within the graph. By merging closely connected nodes, the coarser graph can help reveal the underlying community structures more efficiently [16, 17].

Multigrid Methods Graph coarsening plays a central role in algebraic multigrid methods as well as the related class of multilevel incomplete LU factorizations. The idea is to project the original problem to an ”equivalent” problem on the coarse mesh and the solution is interpolated back to the fine level grid [14]. Graph coarsening significantly accelerates the processing speed for multigrid methods.

Machine Learning Graph-based machine learning methods, such as Graph Convolutional Networks (GCN), have included graph coarsening as one of the core components in building the end-to-end machine learning models for specific graph tasks. One benefit of graph coarsening is to reduce

computational time and memory requirements for large graphs [7]. Moreover, graph coarsening can often improve the convergence speed and performance for specific tasks [18].

2.3 Spectral Graph Theory

2.3.1 Overview

We introduce some basic concepts from spectral graph theory, which we believe offers a more robust foundation to understand our work on graph coarsening. In essence, the spectral graph theory explores the fundamental relationship between the structural properties of graphs and the spectra of matrices associated with these graphs, typically either the adjacency matrix, or the Laplacian matrix. There are many tutorials and several textbooks that have been written about spectral graph theory. See for example [3].

In this thesis, we focus on the normalized Laplacian matrix whose eigenvalues are closely related to a number of topological invariants of the graph. The normalized Laplacian of a graph is defined using the following equation:

$$I - D^{-1}W \tag{2.1}$$

where W is the weight matrix of the graph and D is the diagonal matrix such that each diagonal entry is defined by $D_{i,i} = \sum_j W_{i,j}$.

for our purposes, the spectrum of the graph is defined as the sequence of the eigenvalues of the normalized graph Laplacian ordered as follows: $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n \leq 2$.

2.3.2 Relationship with Graph Structures

Extensive studies have shown that the spectra are closely related to the graph structures. For example, the multiplicity of the eigenvalue 0 corresponds to the number of connected components in the graph. The largest eigenvalue λ_n is equal to 2 if and only if the graph is bipartite [3]. Moreover, the second smallest eigenvalue λ_2 is of special importance and has been shown to be closely related to the convergence rate of random walks on the graph, "chemical conductance," Cheeger's constant, graph diameter, and graph quasi-randomness [3]. Other eigenvalues also play an important role in capturing the graph structural properties, which has not been fully explored yet.

2.3.3 Applications

While we focus in this thesis on the application of spectral graph theory on graph coarsening, other major applications include:

- **Spectral Clustering.** Spectral clsutering is a widely-used technique that groups similar data points by analyzing the eigenvectors of the graph's Laplacian matrix [19].

- **Image Segmentation.** In computer vision, spectral graph theory can be applied to segment images into meaningful regions, by representing the image as a graph and using eigenvectors to identify boundaries between different regions [20, 21].

- **Dimensionality Reduction.** Spectral methods for dimensionality reduction are techniques that use the spectral (eigenvalue) decomposition of matrices derived from the data in order to transform the data into a lower-dimensional space. These methods are primarily based

on constructing a similarity graph from the data, where each data point is represented as a node and the edge weights represent the similarity between data points. From this graph, a matrix representation is derived, often the graph Laplacian, but sometimes the adjacency or degree matrix. The eigenvectors corresponding to the smallest eigenvalues of this matrix are then used to represent the data in a lower-dimensional space. Notable examples are Laplacian Eigenmaps, Locally Linear Embedding (LLE) and Isomap [22, 23, 24, 25].

2.4 Graph Neural Network

2.4.1 Overview

Graph neural network (GNN) models have emerged as a powerful tool to solve graph related problems such as node classification, link prediction and graph classification [4, 5, 7, 26, 27, 28]. Most of GNN models follow a message-passing paradigm where node representations are formed by iteratively aggregating information from neighboring nodes, followed by a graph pooling function to generate graph representations [4, 5, 6, 7].

Starting with the initial node features $\boldsymbol{H}^{(0)} = \boldsymbol{H}$, the message passing graph neural network iteratively updates node representations by aggregating information from neighboring nodes. The following functions characterize the graph convolutional process [4, 6, 9],

$$\begin{aligned} \boldsymbol{h}_v^{(i)} &= \textbf{Aggregate}^{(i)}(\boldsymbol{h}_v^{(i-1)}, \boldsymbol{m}_v^{(i)}), \\ \boldsymbol{m}_v^{(i)} &= \textbf{Msg}^{(i)}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}(v)\}) \end{aligned} \tag{2.2}$$

where $\boldsymbol{h}_u^i$ is the representation of node u at iteration i and **Msg** is the message function that aggregates neighborhood information. The graph representation $\boldsymbol{h}_{\mathcal{G}}$ is obtained by applying the

pooling function **Pool** on the corresponding node representations as,

$$\boldsymbol{h}_{\mathcal{G}} = \textbf{Readout}(\boldsymbol{h}_v^{(k)}, v \in \mathcal{V}) \tag{2.3}$$

2.4.2 Permutation Invariance

Most GNN models need to satisfy the property of **Permutation Invariance**, that is, the graph representations remain invariant under any permutations applied on the inputs. We denote $\boldsymbol{\pi} = \{v_1, v_2, ..., v_n\}$ as a node permutation and $\mathcal{G}_{\boldsymbol{\pi}} = (\mathcal{V}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}, \boldsymbol{H}_{\boldsymbol{\pi}})$ is the graph where the node indices, weight matrix and feature matrix are permutated over $\boldsymbol{\pi}$. The permutation invariance is expressed as follows,

$$\textbf{GNN}(\boldsymbol{H}, \boldsymbol{W}) = \textbf{GNN}(\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}), \forall \boldsymbol{\pi} \in \boldsymbol{\Pi}. \tag{2.4}$$

2.4.3 Examples of GNN Models

There are many variants of GNN models with different message, aggregation and readout functions. Some of the notable examples are as follows,

Graph Convolutional Network (GCN). GCN is one of the most popular GNN models which borrow the idea of Convolutional Neural Network (CNN) in the graph domain. The GCN model leverages the graph convolutional function to aggregate information from neighboring nodes and apply a pooling function to form graph representations from node representations. GCN models achieve high performance on node and graph classification tasks[28, 29].

Graph Sample and Aggregation (GraphSAGE). GraphSAGE models extends the GCN framework

by introducing a more flexible, scalable, and inductive learning approach. GraphSAGE learns to generate embeddings for nodes by sampling and aggregating information from their local neighborhood, which is particularly useful for learning on large graphs or graphs with unseen nodes during training [30].

Graph Attention Network (GAT). GAT models introduce the attention mechanisms into GNNs, which enables them to weigh the importance of neighboring nodes differently when aggregating information. Through the attention mechanism, the message function can focus on the most relevant neighbors and capture more complex graph structures. GAT models have been applied to various tasks, including node classification, link prediction, and graph classification [31].

2.4.4 Expressiveness of GNN

The expressiveness of graph representation models is the ability to capture the intricate structure of graph data and encode this information into node or graph representations. For GNN models, the expressiveness is determined by the model architecture and the specific aggregation, messaging and readout functions. The number of message propagation steps can have a significant influence on the expressiveness[9, 32]. GNN models with a large number of propagation steps can capture long range of dependencies and interactions [33, 34, 35].

Test of Isomorphism. The expressiveness of GNN models is often evaluated on the ability to distinguish non-isomorphic graphs. Isomorphic graphs are pairs of graphs with the same structure but but may differ in the labels of their vertices and edges up to node permutations. Two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \boldsymbol{W}_1, \boldsymbol{H}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \boldsymbol{W}_2, \boldsymbol{H}_2)$ are said to be isomorphic when there exist a permutation π such that the edges and node features are identical after applying the node

permutations $\boldsymbol{W}_{1,\pi} = \boldsymbol{W}_2$ and $\boldsymbol{H}_{1,\pi} = \boldsymbol{H}_2$. The ability to distinguish isomorphic graphs is an important component in understanding the expressiveness of graph representation models.

Weisfeiler-Lehman Tests of Isomorphism. One of the most powerful and widely used tests of graph isomorphism is the Weisfeiler-Lehman Test of Isomorphism (WL) [36, 37]. The WL test is an refinement-based method that iteratively updates the node labels with local neighborhood until the stopping conditions are reached. The 1-dimensional WL test (1-WL) updates the node labels based on the immediate neighbors during the labeling process. The detailed steps of 1-WL are as follows,

- **Initialization**: Assign an initial label to each node in each graph.
- **Iterative Labelling**: For each iteration, a new label is computed for each node based on its current label and the multiset of labels of its neighbors.
- **Stopping Condition**: the process continues until no new labels are generated or until a pre-determined number of iterations are reached.
- **Decision**: After the last iteration, nodes of the two graphs are sorted according to their final labels. The two graphs are considered isomorphic if and only if the sequence of node labels are the same for both graphs.

Higher dimensional Weisfeiler-Lehman Isomorphism Tests (k-WL) consider the k-hop neighbors in the color relabeling step, which results stronger capability to distinguish non-isomorphic graphs. Cai et al. show that $(k+1)$-WL has strictly stronger expressive power than k-WL, i.e. there are non-isomorphic graphs which can be distinguished by $(k+1)$-WL but not by k-WL [38].

Recent studies have established key results on the relationship between the expressiveness of GNN and the series of Weisfeiler-Lehman (WL) Tests of Isomorphism (WL). They showed that the expressiveness of the **most powerful** message-passing GNN models does not exceed 1-WL [1, 39]. As 1-WL cannot distinguish between certain graphs that are not isomorphic, such as regular graphs, GNN models suffer from the limitation of expressiveness. How to design graph representations with higher-order of expressiveness exceeding the WL tests remains a challenge.

2.4.5 Limitations

Despite the success GNN models have achieved over a number of applications, GNN models suffer from limitations which affect their effectiveness. Some limitations are listed next.

- **Scalability.** The complexity of graph convolutions and the memory requirements can grow significantly with the size of the graph and the number of iterations. This makes it challenging to apply GNNs to real-world problems that involve large-scale graphs.

- **Expressiveness.** As mentioned previously, the expressiveness of GNN models is limited by 1-WL, which cannot distinguish non-isomorphic graphs, such as pairs of regular graphs [6].

- **Over-smoothing.** As the number of convolutional layers increase, the information incorporated become indistinguishable, resulting in a loss of node-specific information and reduced discriminative power [40].

A number of methods have been proposed to address the limitations which improve the GNN expressiveness and its applicability to a wider range of problems. For example, approximate

methods and graph sampling methods are introduced to solve the scalability problem [**?**]. To address the limitation of expressiveness, higher-order graph neural network models are proposed with provably higher expressiveness than 1-WL [41, 42]. To tackle the *over-smoothing* problem, various methods have been proposed to improve the effectiveness, such as DropEdge with the idea of dropping edges in the input graph during training, which reduces the information flowing between nodes thus preventing the over-smoothing [43].

2.5 Structural Brain Networks

2.5.1 Overview

Diffusion Magnetic Resonance Imaging (MRI) exploits the anisotropic diffusion of water molecules in the brain to enable the estimation of the brain's anatomical fiber tracts at a relatively high resolution. In particular, tractographic methods can be used to generate whole-brain anatomical connectivity matrix where each element provides an estimate of the connectivity strength between the corresponding voxels. Structural brain networks are built using the connectivity information and a predefined brain parcellation, where the nodes of the network represent the brain regions and the edge weights capture the connectivity strengths between the corresponding brain regions.

Table 2.1: Demographics of NKI Sample

Age Group	**4-9**	**10-19**	**20-29**	**30-39**	**40-49**	**50-59**	**60-69**	**70-85**	**Total**
Female	4	18	19	8	5	6	9	10	79
Male	5	16	24	9	26	8	5	4	97
Total	9	34	43	17	31	14	14	14	176

2.5.2 Diffusion Tensor Imaging Processing

2.5.2.1 Data Collection

The diffusion MRI dataset used in this research is taken from the publicly available Nathan Kline Institute (NKI)/ Rockland dataset[1], which consists of data for individuals whose ages range from 4 to 85. The diffusion MRI was performed using a SIEMENS MAGNETOM TrioTim syngo MR B15 system. The high-angular resolution diffusion imaging protocol was used to assess white matter integrity as measured by fractional anisotropy. Diffusion tensor data were collected using a single- shot, echo-planar, single refocusing spin-echo, T2-weighted sequence with a spatial resolution of $2.0\times2.0\times2.0$mm. The sequence parameters were:
TE/TR=91/10000ms, FOV=256mm, axial slice orientation with 58 slices, 64 isotropically distributed diffusion weighted directions, two diffusion weighting values (b=0 and 1000s/mm^2) and six b=0 images. These parameters were calculated using an optimization technique that maximizes the contrast to noise ratio for FA measurements. For each subject, the image data consists of 76 volumes of 3D images of dimensions $128\times128\times53$, each voxel representing $2.0\text{mm}\times2.0\text{mm}\times2.0$mm brain volume.

[1] http://fcon_1000.projects.nitrc.org/indi/pro/nki.html

Figure 2.1: Left: Seed region which is JHU white matter atlas. Middle: target region which is the AAL mask. Right: the result of probabilistic tractography which models the distribution of neuron fiber bundles where the intensity of each voxel represents the number of streamlines passing through that voxel. All figures are imaged in the axial plane

2.5.2.2 Nonlinear Registration

The diffusion images of all subjects are registered to Montreal Neurological Institute (MNI) standard space using the nonlinear registration package *FNIRT* in FSL software [44]. The nonlinear registration process generates the warping coefficients that balance the similarity between the diffusion image and the standard MNI152 image, and the smoothness of the warping coefficients.

2.5.2.3 Probabilistic Tractographical Methods

After a set of standard preprocessing steps, probabilistic tractography is used to model cross fiber distributions for each voxel through the BEDPOSTX package in FSL [45]. Probabilistic tractography is processed through the diffusion toolbox in FSL [46]. The standard white matter atlas is specified as a seed region. The brain atlas AAL mask is specified as the target region, which covers the whole brain cortex region. We generate 50 streamlines from every voxel in the seed region. These streamlines are propagated following the cross fiber distribution computed during a preprocessing step. Curvature threshold is enforced to eliminate unqualified streamlines.

The distance correction option is set to correct for the fact that the distribution drops as the travel distance increases. The tractography output is a structural connectivity network modeled as a weighted graph where each node is a voxel in the target region space and each edge weight corresponds to the relative connectivity strength in terms of the number of streamlines connecting the corresponding pair of voxels. Using this connectivity information and a set of predefined regions of interests (ROIs) or whole brain parcellations, structural brain networks can be built and analyzed. Figure 2.1 shows the seed and target regions as well as the tractography result of a subject.

2.5.3 Brain Parcellation Scheme

The brain parcellation scheme takes as input the probabilistic tractography results represented as a large connectivity matrix, which is a large sparse matrix with millions of nodes and billions of edges.

The k-region brain parcellation is brain segmentations that partition the brain's grey matter into k spatially contiguous regions, such that the connectivity profiles of the voxels in each region are as similar as possible. The parcellations are expected to be consistent among members of a structurally homogeneous population sample. Notable brain parcellations include the following,

- **Brodmann's Areas**. Proposed by Korbinian Brodmann in 1909, Brodmann's areas divide the brain based on cytoarchitectonic properties, or the distribution and arrangement of neurons in the cortex [47].

- **Automated Anatomical Labeling (AAL).** The AAL atlas is a widely used atlas introduced by Tzourio-Mazoyer et al. in 2002. It divides the brain based on anatomical landmarks into

116 regions (90 cortical and subcortical, and 26 cerebellar) [8]. It is commonly used for voxel-based morphometry and functional fMRI studies[48, 49].

- **Harvard-Oxford Atlas**. The Harvard-Oxford Atlas is a probabilistic atlas that provides a standard space for neuroimaging analyses. It's based on the segmentation of structural images of the brain from 48 subjects [50].

2.5.4 Construction of Structural Brain Networks

Given the brain parcellations and the connectivity information revealed by probabilistic tractography, the structural brain network are built where the nodes represent the regions in the parcellation and the edges reflect the connectivity strength between the corresponding regions. The edge weights are defined as follows,

$$W(R_i, R_j) = \sum_{v_a \in R_i, v_b \in R_j} \boldsymbol{W}(v_a, v_b) \tag{2.5}$$

where $W(v_a, v_b)$ represents the number of streamlines connecting the two voxels as generated by the tractographic results.

2.5.5 Graph Theoretical Analysis

Graph theoretical analysis is applied on the structural brain networks to extract graph patterns [51]. Many global and local graph-theoretic measurements have been introduced to characterize structural brain networks [52, 53, 54]. The following are some of the mostly common used graph measurements,

- **Characteristic path length (CPL)**. The characteristic path length tries to capture the

network integration and is computed as the average of the shortest paths between all pairs of vertices.

$$CPL = \frac{1}{N(N-1)} \sum_{i,j,i \neq j} d(i,j) \tag{2.6}$$

where $d(i,j)$ is the shortest path between node i and j.

- **Global efficiency** (E_{global}). The global efficiency of a graph is the average of the inverse of the shortest paths between all pairs of vertices and hence tries to capture how well pairs of nodes are connected. [55].

$$E_{global} = \frac{1}{N(N-1)} \sum_{i,j,i \neq j} \frac{1}{d(i,j)} \tag{2.7}$$

- **Clustering coefficient**. The clustering coefficient tries to capture graph separation and is defined as the average of the local clustering coefficient at each node.

$$C = \frac{1}{N} \sum C_i \tag{2.8}$$

where the local clustering coefficient at node i is defined as

$$C_i = \frac{2\Gamma_i}{deg_i(deg_i - 1)} \tag{2.9}$$

Γ_i is the number of triangles around node i,

$$\Gamma_i = \frac{1}{2} \sum_{j,h} u_{i,j} u_{i,h} u_{j,h} \tag{2.10}$$

Note that $deg_i > 0$ in our case since isolated nodes are removed from the network.

- **Sparsity ratio**. The sparsity ratio is defined as the ratio of the number of edges over the number of possible edges between nodes, that is,

$$\text{Sparsity} = \frac{\sum_{i,j} u_{i,j}}{k(k-1)} \tag{2.11}$$

A more detailed description of graph theoretic measures commonly used to analyze structural brain networks can be found in [53]. These traditional graph-theoretical measurements summarize the complex graph structures of structural brain networks [56]. However, when the graph size increases, the graph measurements cannot fully capture the complex network patterns.

Chapter 3: Graph Coarsening with Preserved Spectral Properties

3.1 Introduction

Graphs are widely used to represent object relationships in real-world applications. As many applications involve large-scale graphs with complex structures, it is generally hard to explore and analyze the key properties directly from large graphs. Hence graph coarsening techniques have been commonly used to facilitate this process [57, 58].

Generally speaking, the aim of any graph reduction scheme is to reduce the number of nodes and edges of a graph, while also ensuring that the "essential properties" of the original graph are preserved. The question of what properties should be preserved remains inconclusive, but there is significant evidence that they should relate to the spectrum of a graph operator, such as the adjacency or normalized Laplacian matrices [59, 60]. A long list of theorems in spectral graph theory shows that the combinatorial properties of a graph are aptly captured by its spectrum. As such, graphs with similar spectra are generally regarded to share similar global and local structure [61, 62]. Based on this realization, modern graph sparsification techniques [63, 64, 65] have moved on from previously considered objectives, such as cut and shortest-path distance preservation, and now aim to find sparse spectrally similar graphs.

In contrast to graph sparsification, there has been little progress towards attaining spectrum preservation guarantees in coarsening. The foremost roadblock seems to lie in defining what

spectral similarity should entail for graphs of different sizes. The original and coarse graphs have different numbers of eigenvalues and eigenvectors, which prohibit a direct comparison. To circumvent this issue, recent works have considered restricting the guarantees to a subset of the spectrum [9, 66]. However, focusing only on a subset of eigenvalues and eigenvectors also means that important information of the graph spectrum is ignored.

In this chapter, we start by reconsidering the fundamental spectral distance metric [64, 67, 68, 69], which compares two graphs by means of a norm of their eigenvalue differences. This metric is seemingly inappropriate as it necessitates that two graphs have the same number of eigenvalues. However, we find that in the context of coarsening, this difficulty can be circumvented by substituting the coarse graph with its lifted counterpart: the latter contains the same information as the former while also having the correct number of eigenvalues.

Our analysis shows that the proposed distance naturally captures the graph changes in the graph coarsening process. In particular, when the graph coarsening merges nodes that have similar connections to the rest of the graph, the spectral distance is provably small. By merging similarly connected nodes, nodes and edges in coarse graphs are able to represent the connectivity patterns of the original graphs, thus preserving structural and connectivity information.

Our contributions in this chapter are summarized as follows:

- We show how the spectral distance [64, 67, 68, 69], though originally restricted to graphs of the same size, can be utilized to measure how similar a graph is with its coarsened counterpart.
- We examine how the new spectral distance captures graph structural changes occurring during the graph coarsening process.

- We present two coarsening algorithms that provably minimize the spectral distance.
- We experimentally show that the proposed methods outperform other graph coarsening algorithms on two graph related tasks.

All the proofs can be found in the Appendix.

3.2 Related Work

Recent works have proposed to coarsen graphs by preserving the spectral properties of the matrix representations of graphs [9, 10, 60, 66, 70]. For example, Loukas et al. proposed to preserve the action of the graph Laplacian with respect to an (eigen)-space of fixed dimension, arguing that this suffices to capture the global properties of graphs relevant to partitioning and spectral clustering [9]. Durfee et al. proposed to preserve the all-pairs effective resistance [10]. Garg and Jaakkola defined a cost based on the theory of optimal transport [11]. Saket et al. suggested a Minimum Description Length (MDL) principle relevant to unweighted graphs [12]. Most of these distance functions are specific to particular applications; the question of how to define an application-independent graph coarsening framework remains a challenge.

There is a sizable literature dealing with the characterization of graphs in terms of their spectral properties [64, 71, 72]. Previous work defined distance functions based on Laplacian eigenvalues which measure differences between graphs [64, 67]. Spielman and Teng introduced a notion of spectral similarity for two graphs in their graph sparsification framework [59, 65]. Recently, Tsitsulin et al. proposed an efficient graph feature extractor, based on Laplacian spectrum, for comparisons of large graphs [71]. Dong uses spectral densities to visualize and estimate meaningful information about graph structures [72]. Nevertheless, despite the popularity

of spectral methods, the graph spectrum remains little explored in the context of graph coarsening.

3.3 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ be a graph, with $\mathcal{V}$ a set of $N = |\mathcal{V}|$ nodes, $\mathcal{E}$ a set of $M = |\mathcal{E}|$ edges, and $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ the weighted adjacency matrix. We denote the node v_i the node by $\boldsymbol{w}(i) \in \mathbb{R}^N$ representing the vector of the weights of the edges incident on v_i and by $d(i) = \sum_{j=1}^{N} \boldsymbol{W}(i, j)$ the node degree of v_i. The graphs considered in this work are weighted, undirected, and possess no isolated nodes (i.e. $d(i) > 0$ for all v_i).

The combinatorial and normalized Laplacians of $\mathcal{G}$ are defined as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \quad \text{and} \quad \boldsymbol{\mathcal{L}} = \boldsymbol{I}_N - \boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}, \tag{3.1}$$

respectively, where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix and $\boldsymbol{D}$ is the diagonal degree matrix with $\boldsymbol{D}(i, i) = d(i)$.

3.3.1 Graph Coarsening

The coarse graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c, \boldsymbol{W}_c)$ with $n = |\mathcal{V}_c|$ is obtained from the original graph $\mathcal{G}$ by first selecting a set of non-overlapping graph partitions $\mathcal{P} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\} \subset \mathcal{V}$, which cover all the nodes in $\mathcal{V}$. Each partition $\mathcal{S}_p$ corresponds to a "super-node" denoted by s_p and the "super-edge" connecting the super-nodes $\boldsymbol{W}_c(p, q)$ has weight equal to the accumulative edge

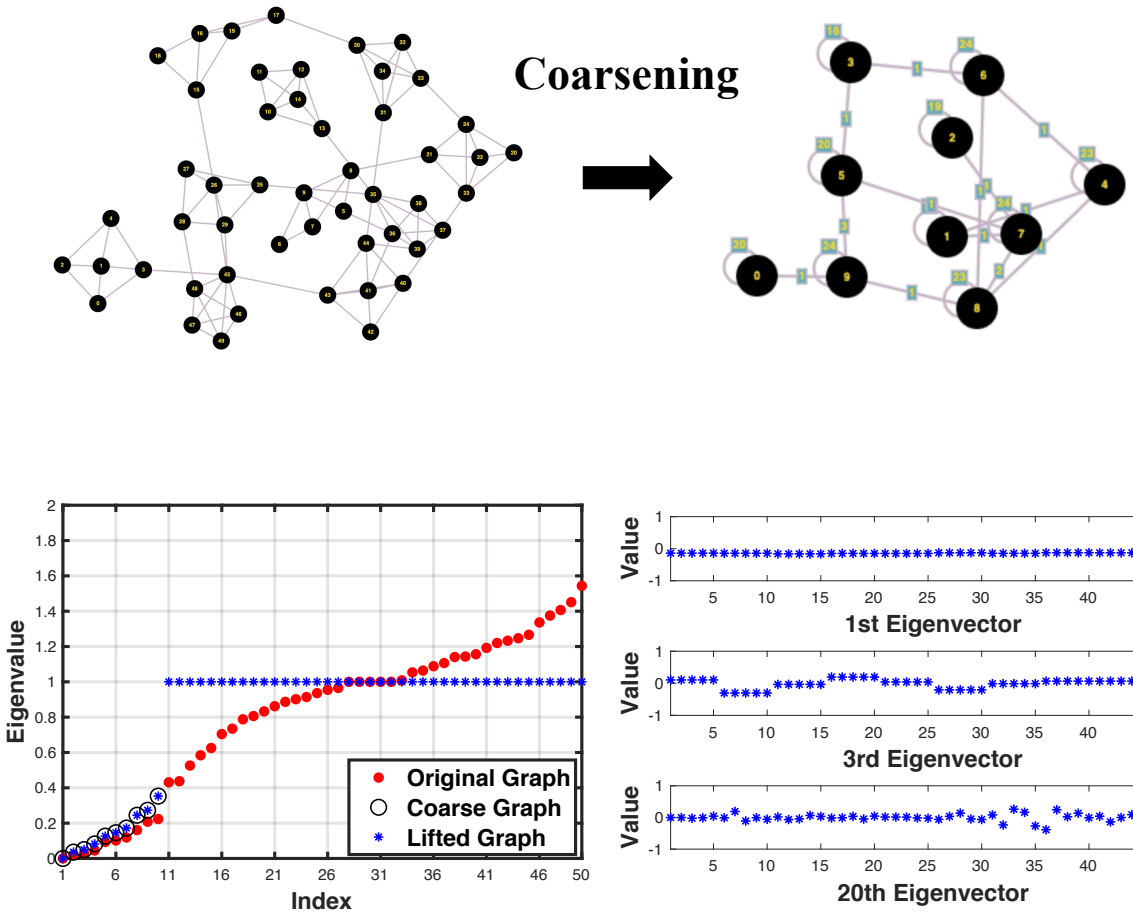


Figure 3.1: **Left**: an example illustrating the graph coarsening process. The original graph is a random graph sampled from the stochastic block model with 50-node and 10 predefined blocks. The coarse graph is obtained from the predefined partitions. **Right**: Eigenvalues and eigenvectors of normalized Laplacian matrices of original, coarse and lifted graphs. The eigenvalues of coarsened graphs align with the eigenvalues of original graphs and the eigenvectors indicate the block membership information.

weights between nodes in the corresponding graph partitions $\mathcal{S}_p$ and $\mathcal{S}_q$:

$$\boldsymbol{W}_c(p,q) = w(\mathcal{S}_p, \mathcal{S}_q) := \sum_{v_i \in \mathcal{S}_p, v_j \in \mathcal{S}_q} \boldsymbol{W}(i,j) \tag{3.2}$$

Let $\boldsymbol{P} \in \mathbb{R}^{n \times N}$ be the matrix whose columns are partition indicator vectors:

$$\boldsymbol{P}(p,i) = \begin{cases} 1, & \text{if } v_i \in \mathcal{S}_p \\ 0, & \text{othewise.} \end{cases}$$

It is then well known that the weight matrix $\boldsymbol{W}_c$ of the coarse graph $\mathcal{G}_c$ satisfies

$$\boldsymbol{W}_c = \boldsymbol{P}\boldsymbol{W}\boldsymbol{P}^\top.$$

The definition of the coarsened Laplacian matrices follows directly:

$$\boldsymbol{L}_c = \boldsymbol{D}_c - \boldsymbol{W}_c \quad \text{and} \quad \boldsymbol{\mathcal{L}}_c = \boldsymbol{I}_n - \boldsymbol{D}_c^{-1/2}\boldsymbol{W}_c\boldsymbol{D}_c^{-1/2}.$$

Similarly to the adjacency matrix, the combinatorial Laplacian of the coarse graph can be obtained by the formula $\boldsymbol{L}_c = \boldsymbol{P}\boldsymbol{L}\boldsymbol{P}^\top$. The same however doesn't hold for the normalized Laplacian, as in general $\boldsymbol{P}\boldsymbol{\mathcal{L}}\boldsymbol{P}^\top \neq \boldsymbol{\mathcal{L}}_c$.

3.3.2 Graph Lifting

We define $\mathcal{G}_l = (\mathcal{V}, \mathcal{E}_l, \boldsymbol{W}_l)$ to be the graph lifted from the coarse graph $\mathcal{G}_c$ with respect to a set of non-overlapping partitions $\mathcal{P}$. In graph lifting, each node s_p of the coarse graph is lifted to $|\mathcal{S}_p|$ nodes and nodes in the lifted graph are connected by edges whose weight is equal to the coarse edge weight normalized by the sizes of partitions. Specifically, for any $v_i \in \mathcal{S}_p$ and

$v_j \in \mathcal{S}_q$ we have:

$$\begin{aligned} \boldsymbol{W}_l(i,j) = \frac{w(\mathcal{S}_p, \mathcal{S}_q)}{|\mathcal{S}_p||\mathcal{S}_q|} &= \frac{\sum_{v_i' \in \mathcal{S}_p, v_j' \in S_q} \boldsymbol{W}(i',j')}{|S_p||S_q|} \\ &= \frac{\boldsymbol{W}_c(p,q)}{|S_p||S_q|}. \end{aligned} \tag{3.3}$$

When $\mathcal{S}_p = \mathcal{S}_q = \mathcal{S}$, the weight $\boldsymbol{W}_l(i,j)$ can be seen to be equal to the weight of all edges in the subgraph induced by $\mathcal{S}$, after normalization by $|\mathcal{S}|^2$. It easily follows that if $\boldsymbol{W}(i,j)$ is the same for every $v_i, v_j \in \mathcal{S}$, then also $\boldsymbol{W}_l(i,j) = \boldsymbol{W}(i,j)$, i.e., in-partition weights are exactly preserved by successive coarsening and lifting in this case.

The above combinatorial definition can be expressed in an algebraic form in terms of the the pseudo-inverse $\boldsymbol{P}^+$ of $\boldsymbol{P}$, (i.e., $\boldsymbol{P}\boldsymbol{P}^+ = \boldsymbol{I}$), whose elements are:

$$\boldsymbol{P}^+(j,p) = \begin{cases} \frac{1}{|\mathcal{S}_p|} & \text{if } v_j \in \mathcal{S}_p \\ 0 & \text{otherwise.} \end{cases}$$

With this in place, the adjacency matrices of the lifted and coarse graphs are connected by the following relations:

$$\boldsymbol{W}_l = \boldsymbol{P}^+ \boldsymbol{W}_c \boldsymbol{P}^{\mp} \quad \text{and} \quad \boldsymbol{W}_c = \boldsymbol{P}\boldsymbol{W}_l\boldsymbol{P}^\top.$$

The following equation reveals that lifting preserves the connectivity up to a projection onto the

partitions:

$$\begin{aligned} \boldsymbol{W}_l = \boldsymbol{P}^{+}\boldsymbol{W}_c\boldsymbol{P}^{\mp} &= \boldsymbol{P}^{+}\boldsymbol{P}\boldsymbol{W}\boldsymbol{P}^{\top}\boldsymbol{P}^{\mp} \\ &= \boldsymbol{\Pi}\boldsymbol{W}\boldsymbol{\Pi}^{\top} = \boldsymbol{\Pi}\boldsymbol{W}\boldsymbol{\Pi}, \end{aligned} \tag{3.4}$$

where $\boldsymbol{\Pi} = \boldsymbol{P}^{+}\boldsymbol{P}$ is a projection matrix, with $\boldsymbol{\Pi}\boldsymbol{\Pi} = \boldsymbol{P}^{+}\boldsymbol{P}\boldsymbol{P}^{+}\boldsymbol{P} = \boldsymbol{P}^{+}\boldsymbol{P} = \boldsymbol{\Pi}$.

The lifted Laplacian matrices are given by

$$\boldsymbol{L}_l = \boldsymbol{P}^{+}\boldsymbol{L}_c\boldsymbol{P}^{\mp} \quad \text{and} \quad \boldsymbol{\mathcal{L}}_l = \boldsymbol{C}^{\top}\boldsymbol{\mathcal{L}}_c\boldsymbol{C}, \tag{3.5}$$

where $\boldsymbol{C} \in \mathbb{R}^{n\times N}$ is the *normalized coarsening matrix* whose entries are given by:

$$\boldsymbol{C}(p,i) = \begin{cases} \frac{1}{\sqrt{|\mathcal{S}_p|}} & \text{if } v_i \in \mathcal{S}_p \\ 0 & \text{otherwise,} \end{cases}$$

such that $\boldsymbol{C}^{\top} = \boldsymbol{C}^{+}$ and $\boldsymbol{C}^{\top}\boldsymbol{C} = \boldsymbol{P}^{\mp}\boldsymbol{P} = \boldsymbol{\Pi}$. In this manner, we have

$$\boldsymbol{L}_c = \boldsymbol{P}\boldsymbol{L}_l\boldsymbol{P}^{\top} \quad \text{and} \quad \boldsymbol{\mathcal{L}}_c = \boldsymbol{C}\boldsymbol{\mathcal{L}}_l\boldsymbol{C}^{\top}. \tag{3.6}$$

For a more in-depth discussion of the mathematics of graph coarsening and graph lifting, we refer the interested reader to [9].

3.4 Spectral Distance

We start by briefly reviewing some basic facts about the spectrum associated with the Laplacian matrix of a coarse graph. We then demonstrate how to exploit these properties in order to render the classical spectral distance metric amenable to (coarse) graphs of different sizes.

3.4.1 Properties of the Coarse Laplacian Spectrum

Denote the eigenvalues and eigenvectors of the normalized Laplacian matrices as $\boldsymbol{\lambda}$ and $\boldsymbol{u}$, respectively, with $\boldsymbol{\mathcal{L}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ where the i-th column of $\boldsymbol{U}$ corresponds to $\boldsymbol{u}_i$ and $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda})$. The eigenvalues are ordered in non-decreasing order.

Property 3.4.1 (**Interlacing**. Section 5.3 in [73])**.** *The normalized Laplacian eigenvalues of the original and coarsened graphs satisfy*

$$\boldsymbol{\lambda}(i) \leq \boldsymbol{\lambda}_c(i) \leq \boldsymbol{\lambda}(i + N - n) \quad \textit{for all} \quad i = 1, \ldots, n.$$

Property 3.4.1 is a general interlacing inequality that captures pairwise difference between the eigenvalues of the original and coarse graph Laplacians [3, 74]. Since it holds for any graph and coarsening, the inequality will, in some cases, be loose.

Property 3.4.2 (**Eigenvalue Preservation**)**.** *The normalized Laplacian eigenvalues of the lifted graph contain all eigenvalues of the coarse graph and additional eigenvalues* 1 *with* $(N - n)$ *multiplicity.*

Property 3.4.3 (**Eigenvector Preservation**). *The eigenvectors of the coarse graph lifted by* $\boldsymbol{C}$*, i.e.* $\boldsymbol{u}_l = \boldsymbol{C}\boldsymbol{u}_c$ *are the eigenvectors of* $\boldsymbol{\mathcal{L}}_l$*.*

Proof. We start by noticing that the projection matrix $\boldsymbol{\Pi}$ acts as an identity matrix w.r.t. the lifted normalized Laplacian $\boldsymbol{\mathcal{L}}_l = \boldsymbol{\Pi}\boldsymbol{\mathcal{L}}_l\boldsymbol{\Pi}$, since $\boldsymbol{\mathcal{L}}_l = \boldsymbol{C}^\top\boldsymbol{\mathcal{L}}_c\boldsymbol{C} = \boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{\mathcal{L}}_l\boldsymbol{C}^\top\boldsymbol{C} = \boldsymbol{\Pi}\boldsymbol{\mathcal{L}}_l\boldsymbol{\Pi}$. Now, consider the following eigenvalue equation:

$$\begin{aligned}
\boldsymbol{\mathcal{L}}_c\boldsymbol{u}_c &= \lambda_c\boldsymbol{u}_c \\
\boldsymbol{C}\boldsymbol{\mathcal{L}}_l\boldsymbol{C}^\top\boldsymbol{u}_c &= \lambda_c\boldsymbol{u}_c \\
\boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{\mathcal{L}}_l\boldsymbol{C}^\top\boldsymbol{u}_c &= \lambda_c\boldsymbol{C}^\top\boldsymbol{u}_c \\
\boldsymbol{\Pi}\boldsymbol{\mathcal{L}}_l\boldsymbol{\Pi}\boldsymbol{C}^\top\boldsymbol{u}_c &= \lambda_c\boldsymbol{C}^\top\boldsymbol{u}_c \\
\boldsymbol{\mathcal{L}}_l\boldsymbol{C}^\top\boldsymbol{u}_c &= \lambda_c\boldsymbol{C}^\top\boldsymbol{u}_c
\end{aligned}$$

Note that in the fourth step, we used the relation $\boldsymbol{C}^\top = \boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{C}^\top = \boldsymbol{\Pi}\boldsymbol{C}^\top$, which holds due to the properties of the Moore-Penrose pseudo-inverse. Thus, $\boldsymbol{C}^\top\boldsymbol{u}_c$ are eigenvectors of $\boldsymbol{\mathcal{L}}_l$ with the corresponding eigenvalues of the coarse graph.

To show there are $N - n$ additional eigenvalues 1, one can observe that $\boldsymbol{I}_N - \boldsymbol{\mathcal{L}} = \boldsymbol{D}_l^{-1/2}\boldsymbol{W}_l\boldsymbol{D}_l^{-1/2}$ is a rank-n matrix because nodes within the same partition have exactly the same edge weights. Hence $\boldsymbol{I}_N - \boldsymbol{\mathcal{L}}$ contains $N - 1$ eigenvalue 0 and correspondingly $\boldsymbol{\mathcal{L}}$ contains eigenvalue 1 with $N - n$ multiplicity. □

Property 3.4.2 and 3.4.3 state that the action of lifting preserves most spectral properties of the coarse graph. Thus, we may use the lifted graph as a proxy to define the distance function [75]. Figure 3.1 shows an example illustrating the graph coarsening process as well as the effect

on the graph spectrum.

3.4.2 Spectral Distance

In the following, we propose two notions of the spectral distance to quantify the difference between original and coarse graphs. We first use the lifted graph as the "proxy" of the coarse graph and define the *full spectral distance*:

Definition 3.4.4. *The full spectral distance between graph $\mathcal{G}$ and $\mathcal{G}_c$ is defined as follows:*

$$SD_{full}(\mathcal{G}, \mathcal{G}_c) = \|\boldsymbol{\lambda} - \boldsymbol{\lambda}_l\|_1 = \sum_{i=1}^{N} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)|,$$

where vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}_l$ contain the eigenvalues of the original and lifted graphs.

As the original and lifted graphs have the same number of nodes, we may directly use a norm to measure the pairwise differences between eigenvalues.

On the flip side, the definition requires computing all eigenvalues of original graphs regardless of the coarse graph size, which is computationally expensive, especially for large graphs. The limitation motivates us to define the *partial spectral distance* by selecting part of the terms in the full spectral distance definition.

Let k_1 and k_2 be defined as $k_1 = \arg\max_i\{i : \boldsymbol{\lambda}_c(i) < 1\}, k_2 = N - n + k_1$. We expand

the full spectral distance into three terms as follows:

$$\begin{aligned} SD_{\text{full}}(\mathcal{G}, \mathcal{G}_c) &= \sum_{i=1}^{N} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| \\ &= \sum_{i=1}^{k_1} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| + \sum_{i=k_1+1}^{k_2} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| \\ &\quad + \sum_{i=k_2+1}^{N} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| \qquad (3.7) \\ &= \sum_{i=1}^{k_1} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_c(i)| + \sum_{i=k_1+1}^{k_2} |\boldsymbol{\lambda}(i) - 1| \\ &\quad + \sum_{i=k_2+1}^{N} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_c(i - N + n)| \qquad (3.8) \end{aligned}$$

The last equation is from the Property 3.4.2 where $\boldsymbol{\lambda}_l$ contains eigenvalues of the coarse graph as well as eigenvalue 1 with $N - n$ multiplicity. Eigenvalue $\boldsymbol{\lambda}_l$ satisfies:

$$\boldsymbol{\lambda}_l(i) = \begin{cases} \boldsymbol{\lambda}_c(i) & i \leq k_1 \\ 1 & k_1 + 1 \leq i \leq k_2 \\ \boldsymbol{\lambda}_c(i - N + n) & i > k_2 \end{cases}$$

With this in place, we define the *partial spectral distance* to be equal to the full spectral distance minus the $N - n$ terms for which $\boldsymbol{\lambda}_l = 1$:

Definition 3.4.5. *The partial spectral distance between graph $\mathcal{G}$ and $\mathcal{G}_c$ is defined as*

$$\begin{aligned} SD_{part}(\mathcal{G}, \mathcal{G}_c) = &\sum_{i=1}^{k} |\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_c(i)| + \\ &\sum_{i=k+1}^{n} |\boldsymbol{\lambda}_c(i) - \boldsymbol{\lambda}(i + N - n)|, \end{aligned}$$

where $k = \arg\max_i \{i : \boldsymbol{\lambda}_c(i) < 1\}$.

For the partial spectral distance, we only need to compute n rather than N eigenvalues of the normalized Laplacian of the original graph, which significantly reduces the computational cost when $n \ll N$.

The full and partial spectral distances are related by,

$$SD_{\text{full}}(\mathcal{G}, \mathcal{G}_c) = SD_{\text{part}}(\mathcal{G}, \mathcal{G}_c) + \sum_{i=k_1+1}^{k_2} |\boldsymbol{\lambda}(i) - 1|$$

The excluded terms $\sum_{i=k_1+1}^{k_2} |\boldsymbol{\lambda}(i) - 1|$ measure the closeness of the original Laplacian eigenvalues and eigenvalue 1. The two definitions are equivalent when the normalized Laplacian of the original graph $\boldsymbol{\mathcal{L}}_N$ contains eigenvalue 1 with $N - n$ multiplicity. The condition is equivalent to asserting that the adjacency matrix W is *singular* with $N - n$ algebraic multiplicity of the eigenvalue 0 [76, 77]. We have observed empirically that, when coarsening nodes that have similar connections, the adjacency matrix has eigenvalues close to 0. In such situations, the terms of the full spectral distance that are excluded by the partial spectral distance are almost 0 and the partial distance closely approximates the full one.

Note that both definitions of spectral distance are proper distance metrics over the space of graph Laplacian eigenvalues. However, the spectral distance is not able to distinguish graphs with the same sets of Laplacian eigenvalues (referred to as cospectral graphs [61]). Thus, there could exist multiple coarse graphs corresponding to the same spectral distance.

3.4.3 Relation to Graph Coarsening

To illustrate the connections between spectral distance and graph coarsening, we first consider the ideal case when merged nodes within the same partitions have the same normalized edge weights:

Proposition 3.4.1. *Let the graph $\mathcal{G}_c$ be obtained by coarsening $\mathcal{G}$ with respect to a set of partitions $\mathcal{P} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$. If $\mathcal{P}$ is selected such that every node in a partition has the same normalized edge weights,*

$$\frac{\boldsymbol{w}(i)}{d(i)} = \frac{\boldsymbol{w}(j)}{d(j)} \quad \textit{for all } v_i, v_j \in \mathcal{S} \quad \textit{and} \quad \mathcal{S} \in \mathcal{P} \tag{3.9}$$

then

$$SD_{full}(\mathcal{G}, \mathcal{G}_c) = 0 \quad \textit{and} \quad SD_{part}(\mathcal{G}, \mathcal{G}_c) = 0.$$

Therefore, the ideal graph coarsening attains a minimal (full and partial) spectral distance.

We next provide a more general result on how the spectral distance can capture the structural changes in the graph coarsening framework. Consider the basic coarsening where the coarse graph is formed by merging one pair of nodes (i.e. $n = N - 1$). In this setting, we prove the following:

Proposition 3.4.2. *Suppose the graph $\mathcal{G}_c$ is obtained from $\mathcal{G}$ by merging a pair of nodes $v(a)$ and $v(b)$. If the normalized edge weights of merged nodes satisfy*

$$\left\| \frac{\boldsymbol{w}(a)}{d(a)} - \frac{\boldsymbol{w}(b)}{d(b)} \right\|_1 \leq \epsilon,$$

then the spectral distance between the original and coarse graphs is bounded by

$$SD_{full}(\mathcal{G}, \mathcal{G}_c) \leq N\epsilon \text{ and } SD_{part}(\mathcal{G}, \mathcal{G}_c) \leq n\epsilon.$$

The above proposition states that the spectral distance is bounded by the discrepancy of normalized edge weights of merged nodes. The bound implies that minimizing the nodes' edge weights within the same partitions results in bounded spectral perturbations.

3.5 Algorithms

We propose two graph coarsening algorithms to produce coarse graphs with minimal small spectral distance. The first follows from Proposition 3.4.2 in that the coarse graphs are formed by iteratively merging graph nodes with similar normalized edge weights. The second algorithm is inspired by spectral clustering: we leverage on the combinations of normalized Laplacian eigenvectors combined with k-means clustering to find the graph partitions and the corresponding coarse graphs. Though different, both algorithms are shown to generate coarse graphs of bounded spectral distance.

3.5.1 Multilevel Graph Coarsening

The Multilevel Graph Coarsening (**MGC**) algorithm iteratively merges pairs of nodes which share similar connections. During each iteration, **MGC** searches for the pair of nodes with the most similar normalized edge weights and merges them into super-nodes. To reduce the

Algorithm 1 Multilevel Graph Coarsening (**MGC**)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ and target size of the coarse graph n.
$s \leftarrow N$
while $s > n$ **do**
  for $v_i \in \mathcal{V}_s$ **do**
    for $v_j \in \mathcal{N}_i$ **do**
      $d_s(i,j) = \left\| \frac{\boldsymbol{w}(i)}{d(i)} - \frac{\boldsymbol{w}(j)}{d(j)} \right\|_1$
    end for
  end for
  $i_{\text{min}}, j_{\text{min}} = \arg\min_{i,j} d_s(i,j)$
  $s \leftarrow s - 1$
  Merge nodes $v_{i_{\text{min}}}$ and $v_{j_{\text{min}}}$ to form the coarse graph $\mathcal{G}_s$.
end while
return $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n, \boldsymbol{W}_n)$

computational cost, we constraint the candidate pairs of graph nodes to be within 2-hop distance. We denote $\mathcal{N}_i$ as the set of nodes that are within 2-hops distance from node v_i. The pseudo-code of **MGC** is presented in Algorithm 1.

Analysis. The following corollary bounds the spectral distance of **MGC** algorithm:

Corollary 3.5.1. *Suppose the graph $\mathcal{G}_c$ is coarsened from $\mathcal{G}$ by iteratively merging pairs of nodes $v(a_s)$ and $v(b_s)$ for s from N to $n+1$, if the normalized edge weights of merged nodes satisfy,*

$$\left\| \frac{\boldsymbol{w}(a_s)}{d(a_s)} - \frac{\boldsymbol{w}(b_s)}{d(b_s)} \right\|_1 \leq \epsilon_s,$$

then the spectral distance between the original and coarse graphs is bounded by

$$SD_{full}(\mathcal{G}, \mathcal{G}_c) \leq N \sum_{s=N}^{n+1} \epsilon_s, \quad SD_{part}(\mathcal{G}, \mathcal{G}_c) \leq n \sum_{s=N}^{n+1} \epsilon_s$$

The bound is a direct corollary of Proposition 3.4.2.

Time complexity. The time complexity of **MGC** is $O(M(N+n)(N-n))$, which is derived as follows: For each iteration, the computational cost of the 1-norm in line 6 is $O(s)$. Then the time complexity of the while loop in line 3 is $O(\sum_{s=n}^{N} s \cdot M) = O(M\frac{n+N}{2}(N-n)) = O(M(N+n)(N-n))$. When $n \approx N$, the complexity reduces to $O(MN)$. On the other hand, for $n \ll N$ the complexity becomes $O(MN^2)$.

3.5.2 Spectral Graph Coarsening

The spectral graph coarsening (**SGC**) algorithm identifies the coarsening partitions by attempting to minimize the k-means cost of rows of Laplacian eigenvectors. Different from traditional spectral clustering, we select eigenvectors with the eigenvalues corresponding to the head and tail eigenvalues as in the definition of partial spectral distance in Definition 3.4.5. The procedure is described in Algorithm 2. Notice that, since k_1 is unknown at the start, **SGC** algorithm iterates over different possible combinations of eigenvectors and selects the coarsening with minimum k-means cost.

Analysis. The following theorem relates the partial spectral distance with the k-means cost:

Theorem 3.5.2. *Let the coarse graph $\mathcal{G}_c$ be obtained from Algorithm 2 with graph partition $\mathcal{P}^*$, suppose that the graph coarsening is consistent, i.e., $\boldsymbol{\mathcal{L}}_c = \boldsymbol{C}\boldsymbol{\mathcal{L}}\boldsymbol{C}^\top$, and let the k-means cost satisfy $\mathcal{F}(\boldsymbol{U}, \mathcal{P}^*) < 1$. Then, the partial spectral distance is bounded by*

$$SD_{part}(\mathcal{G}, \mathcal{G}_c) \leq \frac{(n+2)\mathcal{F}(\boldsymbol{U}, \mathcal{P}^*) + 4\sqrt{\mathcal{F}(\boldsymbol{U}, \mathcal{P}^*)}}{1 - \mathcal{F}(\boldsymbol{U}, \mathcal{P}^*)}.$$

The theorem states that the spectral distance is bounded by the k-means clustering cost.

Algorithm 2 Spectral Graph Coarsening (**SGC**)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, eigenvectors $\boldsymbol{U}$ of the normalized Laplacian $\boldsymbol{\mathcal{L}}$, target size n.
if $\boldsymbol{\lambda}(N) \leq 1$ **then**
    Set $k_1 \leftarrow n$ ▷ Spectral Clustering
else
    Set $k_1 \leftarrow \arg\min_k \{k : \boldsymbol{\lambda}(k) \leq 1, k \leq n, \boldsymbol{\lambda}(N - n + k + 1) > 1\}$ ▷ Iterative Spectral Coarsening
end if
$k_2 \leftarrow N - n + k_1$.
while $k_1 \leq n$ **do**
    $\boldsymbol{U}_{k_1} \leftarrow [\boldsymbol{U}(1 : k_1); \boldsymbol{U}(k_2 + 1 : N)]$
    Apply k-means clustering algorithm on the rows of $\boldsymbol{U}_{k_1}$ to obtain graph partitions $\mathcal{P}^*_{k_1}$ that optimizes the following k-means cost:

$$\mathcal{F}(\boldsymbol{U}_{k_1}, \mathcal{P}^*_{k_1}) = \sum_{i=1}^{N} \left(\boldsymbol{r}(i) - \sum_{j \in \mathcal{S}_i} \frac{\boldsymbol{r}(j)}{|\mathcal{S}_i|} \right)^2$$

    where $\boldsymbol{r}(i)$ is the i^{th} row of $\boldsymbol{U}_{k_1}$.
    $k_1 \leftarrow k_1 + 1$, $k_2 = N - n + k_1$
end while
return coarse graph $\mathcal{G}_c$ generated with respect to the partitions with minimum k-means clustering cost as

$$\mathcal{P}^* = \arg\min_{k_1} \mathcal{F}(\boldsymbol{U}_{k_1}, \mathcal{P}^*_{k_1})$$

Further, when the graph eigenvectors point to well-separated clusters and the k-means cost is small, the spectral distance is smaller. The main assumption posed is that $\boldsymbol{\mathcal{L}}_c = \boldsymbol{C}\boldsymbol{\mathcal{L}}\boldsymbol{C}^\top$, which may not hold for some graphs. For situations when this assumption is not met, the claim can be readily reworked to hold for the combinatorial Laplacian matrix for which the relation $\boldsymbol{L}_c = \boldsymbol{P}\boldsymbol{L}\boldsymbol{P}^\top$ always holds.

Time complexity. Excluding the one-time partial sparse eigenvalue decomposition that takes roughly $O(R(Mn + Nn^2))$ time using Lanczos iteration with R restarts and a graph of M edges (we need the smallest and largest n eigenvalues and eigenvectors) [78], the time complexity of **SGC** is $O(KTNn^2)$, where K refers to the number of times the while loop is executed with

Table 3.1: Classification accuracy on coarse graphs that are five times smaller.

Datasets	MUTAG	ENZYMES	NCI1	NCI109	PROTEINS	PTC
EM	78.90	18.92	62.81	61.35	63.72	48.56
LV	79.01	24.68	63.59	60.49	62.72	50.24
METIS	77.62	24.79	59.74	61.64	63.70	49.34
SC	80.37	24.40	63.14	62.57	64.08	50.16
SGC	80.34	29.19	63.94	**63.69**	64.70	**52.76**
MGC	**81.53**	**30.89**	**66.07**	63.55	**65.26**	52.28
Original	86.58	37.32	66.39	64.93	66.60	53.72

$K \leq n$ and $O(TNn^2)$ is the complexity of the k-means clustering (whereas T bounds the number of k-means iterations).

3.6 Experiments

We proceed to empirically evaluate the proposed graph coarsening algorithms on tasks involving real-world and synthetic graphs. Our first experiment considers the classification of coarsened graphs, whereas the second examines how well one may recover the block structure of graphs sampled from the stochastic block model. We show that our graph coarsening algorithms, which optimize the spectral distance, yield minimal classification accuracy degradation and can recover the block structures with high accuracy. Codes for both experiments are publicly available[1].

Baseline Algorithms We compare our methods with the following graph coarsening and partitioning algorithms as,

- **Edge Matching (EM).** The coarse graphs are formed by maximum-weight matching with the weight calculated as $\boldsymbol{W}(i,j)/\max\{d(i), d(j)\}$ [79].

[1] https://github.com/yuj-umd/spectral-coarsening

- **Local Variation (LV).** Local variation methods coarsen a graph in a manner that approximately preserves a subset of its spectrum [9]. Here, we used the neighborhood-based variant and aimed to preserve the first $\max(10, n)$ eigenvectors and eigenvalues. Alternative choices for the preserved eigenspace may yield different results.
- **METIS.** This is a standard graph partitioning algorithm based on multi-level partitioning schemes that are widely used various domains, such as finite element methods and VLSI [13].
- **Spectral Clustering (SC).** Spectral clustering is a widely used graph clustering algorithm that finds densely connected graph partitions determined from the eigenvectors of the graph Laplacian [19]. For a review of recent results on the fast approximation of SC, see [78].

Note that to apply graph partitioning algorithms for coarsening purposes, we coarsen the graphs with respect to the graph partitions following the standard coarsening process as in equation 3.2.

3.6.1 Graph Classification with Coarse Graphs

Graph classification is a well studied graph machine learning problem, with a variety of applications to material design, drug discovery and computational neuroscience [6, 71, 80, 81]. However, some graph classifiers are not scalable for large graphs, such as those encountered in social network analysis and computational neuroscience [81, 82]. Graph coarsening can reduce the graph sizes in the datasets, which provides acceleration on the training and inference of graph classification models. However, if the coarsening is not carefully done, it can also result in loss of useful information and, thus, of classification accuracy. In the following, we quantify the effect of different coarsening choices to graph classification. We utilize various graph coarsening methods

Table 3.2: Recovery Accuracy of Block Structures from Random Graphs in Stochastic Block Model

p, q	Type	EM	LV	METIS	SC	MGC	SGC
$0.2, 0.01$	Associative	0.1819	0.3076	0.7792	**0.7845**	0.3664	**0.7845**
	Disassortative	0.0956	0.1071	0.0815	0.0877	**0.1093**	0.0850
	Mixed	0.1052	0.1944	0.2389	0.3335	0.6062	**0.7107**
$0.5, 0.1$	Associative	0.1015	0.1902	0.7820	**0.7930**	0.2868	**0.7930**
	Disassortative	0.0854	0.1068	0.0602	0.0788	0.1474	**0.7901**
	Mixed	0.0848	0.2241	0.2883	0.4074	**0.7343**	**0.7699**
$0.8, 0.3$	Associative	0.0823	0.1139	0.5596	**0.6532**	0.1172	**0.6532**
	Disassortative	0.0836	0.0976	0.0776	0.1342	**0.7784**	**0.7931**
	Mixed	0.0888	0.1503	0.2929	0.3909	0.5428	**0.7209**

to reduce the size of graphs in the datasets before passing them to the graph classifier. We then evaluate the quality of graph coarsening based on the classification accuracy drop (as compared to the same classifier on the original graphs).

Evaluation We coarsen the graph samples until $n = N/5$, i.e., until their number of nodes is reduced by a factor of five. The classification performance are evaluated based on 10-fold cross validation—in accordance to previous works [5, 6, 71].

Datasets. We use five standard graph classification datasets for graph classification evaluation [80, 83, 84]. Each dataset contains a set of variable-sized graphs stemming from a variety of applications. The graph statistics can be found in Table A.1.

The graph classifier. We use the Network Laplacian Spectral Descriptor (**NetLSD**) combined with a 1-NN classifier as the graph classification method [71]. **NetLSD** was shown as an efficient graph feature extractor and achieve state-of-the art classification performance [71]. Note that **NetLSD** extracts graph features that only depend on the graph structure and does not consider

Table 3.3: Statistics of the graph benchmark datasets.

Datasets	MUTAG	ENZYMES	NCI1	NCI109	PROTEINS	PTC
Sample size	188	600	4110	4127	1108	344
Average $\lvert V\rvert$	17.93	32.63	29.87	29.68	39.06	14.29
Average $\lvert E\rvert$	19.79	62.14	32.3	32.13	72.70	14.69
# classes	2	6	2	2	2	2

node and edge features.

Results Table 3.1 shows the graph classification performance on coarse graphs. In all cases, the proposed graph coarsening algorithms yield better classification accuracy than alternative methods. Interestingly, for four out of the six datasets (**NCI1**, **NCI109**, **PROTEINS**, and **PTC**) there is almost no degradation to the classification accuracy induced by coarsening, even if the graphs in the coarse dataset are five times smaller—this, we believe, is an encouraging result.

3.6.2 Block Recovery in the Stochastic Block Model

In this experiment, we test whether coarsening algorithms can be used to recover the block structures of random graphs sampled from stochastic block models.

The stochastic block model is a random graph model that is commonly used to evaluate graph partitioning and clustering algorithms [85, 86]. The model is parameterized by a probability matrix $\boldsymbol{B} \in [0,1]^{n\times n}$, with graph nodes in blocks i and j being connected with probability $\boldsymbol{B}(i,j)$. Random graphs can be generated from the stochastic block model by sampling the upper triangular entries $\boldsymbol{W}(i,j)$ in accordance with the edge probability. The lower triangular entries are then set as $\boldsymbol{W}(j,i) = \boldsymbol{W}(i,j)$.

We parameterize $\boldsymbol{B}$ with $\boldsymbol{p}$ and $\boldsymbol{q}$ as follows:

- *Assortative*. The diagonal entries of $\boldsymbol{B}$ are $\boldsymbol{p}$ and the off-diagonal entries are $\boldsymbol{q}$.
- *Disassortative*. The diagonal entries of $\boldsymbol{B}$ are $\boldsymbol{q}$ and the off-diagonal entries are $\boldsymbol{p}$.
- *Mixed*. The entries of $\boldsymbol{B}$ are randomly assigned with $\boldsymbol{p}$ and $\boldsymbol{q}$ (each with probability 1/2).

Evaluation We evaluate the performance of graph coarsening algorithms by measuring the discrepancy between the recovered graph partitions and the ground-truth blocks. We use the *Normalized Mutual Information* (**NMI**) to measure the recovery error between any two graph partitions. The definition of **NMI** can be found in the supplementary material.

For each stochastic block model setting, we set $N = 200$ and $n = 10$, with 20 nodes for each partition. We repeat the experiment 10 times and report the average **NMI** metric achieved by each method.

We compare our graph coarsening algorithms with the graph coarsening and partitioning algorithms mentioned earlier. Table 3.2 reports the average **NMI** in three different stochastic block model configurations. Our proposed methods outperform other methods in almost all cases. In particular, our methods achieve high recovery accuracy for disassortative and mixed settings, where traditional graph partitioning algorithms fail to recover accurately. The **EM** and **LV** coarsening algorithms are not optimized for block recovery and thus exhibit far worse performance on this task.

3.7 Conclusion

In this chapter, we propose a new framework for graph coarsening. We leverage the spectral properties of normalized Laplacian matrices to define a new notion of graph distance that quantifies the differences between original and coarse graphs. We argue that the proposed

spectral distance naturally captures the structural changes in the graph coarsening process, and we propose graph coarsening algorithms that guarantee that the coarse graphs exhibit a bounded spectral distance. Experiments show that our proposed methods can outperform other graph coarsening algorithms on graph classification and block recovery tasks.

Chapter 4: Powerful Graph Neural Networks with Sequence Modeling

4.1 Introduction

Recently, graph neural network models (GNN) have emerged as a poweful tool to tackle challenging graph-related problems such as graph classification, link predictions and node classification [4, 5, 7, 26, 27, 28]. Most GNN models follow the message-passing paradigm where the nodes information are propagated along edges to form new node representations [4]. The final graph representations are the result of applying a pooling function over all the node representations.

Various message-passing GNN (MPNN) models have been proposed to address graph-related problems [87, 88]. For example, Xu et al. proposed to use multi-set functions for node aggregation and graph pooling [6, 89]. Gilmer et al. proposed a general message-passing scheme utilizing Set2Set to obtain graph representations [4, 90]. Gao and Ji proposed a *top-k* graph pooling function by downsampling the graph nodes [91].

Despite their recent success, MPNN models suffer from the fundamental limitation of expressiveness. As pointed out by previous studies, the expressive power of standard MPNN does not exceed the 1-dimensional Weisfeiler-Lehman (WL) Isomorphism Test [6, 39]. Therefore, MPNN models cannot distinguish pairs of graphs that the 1-dimensional WL cannot distinguish. For example, MPNN models cannot distinguish regular graphs of different structures. Note that k-regular graphs are defined as graphs with all the node degrees are k but could have different

structures. One example is shown in Figure 4.3).

More generally, Garg et al. recently show MPNN cannot capture certain graph properties, which hampers the model effectiveness in a wide range of applications [9, 92, 93, 94].

To tackle the above problem, previous studies proposed to equip additional features to the node representations to improve model expressiveness. For example, Sato et al. and Abboud et al. proposed to assign random node identifiers to improve the expressiveness [2, 95]. However, MPNN models with random node identifiers usually do not maintain the properties of equivariance and invariance, which are key properties in the GNN model design. Bouritsas et al. used subgraph counts as additional features [94]. The manual feature engineering requires domain-specific knowledge to achieve the optimal performance for certain applications[94].

Recent work introduced *permutation-sensitive* functions to formulate powerful *permutation-invariant* graph functions [30, 96, 97, 98]. For example, in the proposed GraphSAGE model, Hamilton et al. used a Long Short-Term Memory (LSTM) aggregator on sampled sets of neighbors to extract expressive representations [30]. Murphy et al. proposed *Relational Pooling* to build powerful graph presentations with permutation-sensitive functions. However, there are limitations of previous work: LSTM aggregators in GraphSAGE model randomly selects the node permutations which does not satisfy the permutation invariance; Relational Pooling methods requires expensive computation for all n! possible node permutations which makes it difficult to generalize to applications with large-scale graphs.

In this work, we propose SeqGNN, a new graph neural network paradigm that improves the expressiveness of GNN models. Leveraging on the rich expressiveness of sequence models, SeqGNN represents graphs as the composition of sequence representations. With the design of sequence modeling techniques, SeqGNN models can form expressive graph representations

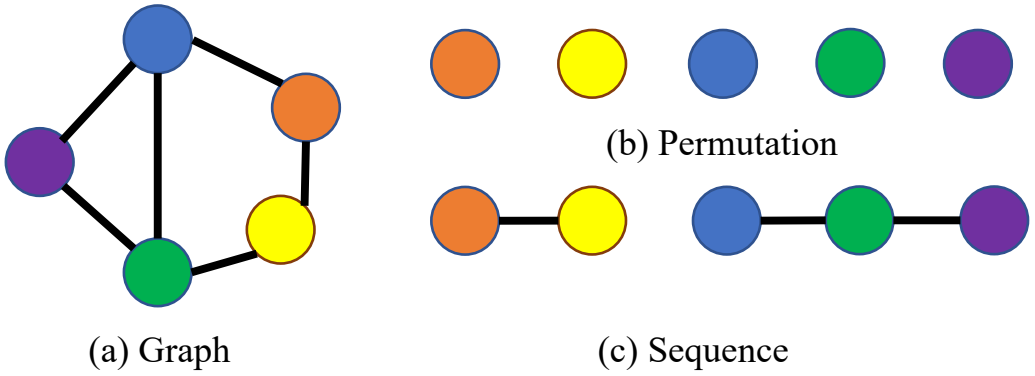


Figure 4.1: Relationships between graphs, node permutations and node sequences. The node permutation is the ordering of the graph nodes and the node sequence contain the edge information between the consecutive nodes.

while maintaining permutation invariance. In addition, SeqGNN can distinguish non-isomorphic graphs which cannot be distinguished by MPNN. We further show that SeqGNN models achieve competitive performance in the real-world graph classification tasks.

4.2 Related Work

Message Passing GNN Message-Passing GNN models have been the primary paradigms of GNN models. Previous work proposed various convolutional and pooling functions that achieve excellent performance in a variety of domains [6, 28, 29, 30, 39, 99, 100]. For example, Xu et al. proposed to use multi-set functions for graph aggregation and pooling [6]. Gilmer et al. proposed a general message-passing scheme utilizing Set2Set to obtain graph representations [4, 90].

Expressive Power of GNN Xu et al. and Morris et al. first pointed that MPNN cannot exceed the expressive power of 1-WL isomorphism tests. Abboud et al. and Sato et al showed that associating graph nodes with extra node features, which can be as simple as random initialization, can improve the expressive power [2, 101]. Loukas et al. theoretically proved that when the

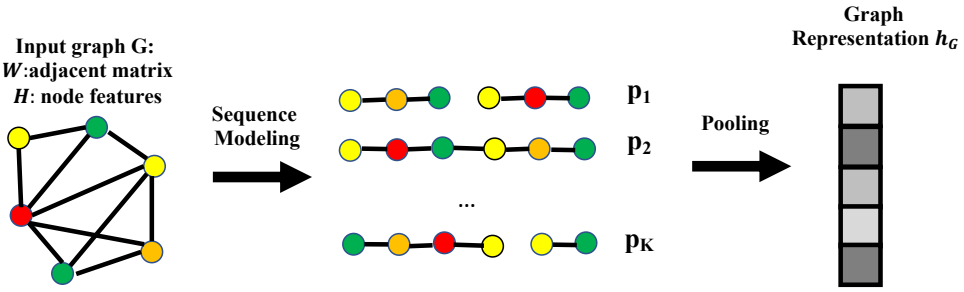


Figure 4.2: The overall framework of the SeqGNN model. (1) The first step is to represent graphs as the set of node sequences. The node sequence consists of the permutation of graph nodes and their associated edge information between consecutive nodes. The node sequences and the associated weights are determined by the sequence sampling methods and the specific graph structures. (2) The second step is learn the sequence representations which are further combined to form the graph representations.

nodes are equipped with the unique node identifiers, GNN models can approximate any Turing computable functions over connected attributed graphs [32]. Maron et al. proposed high-order graph networks that are more powerful GNN but suffer from high computational complexity[41, 42].

Sequence Modeling Sequence modeling is a type of machine learning task that involves predicting or understanding a sequence of data points. Sequence data consist of data points where each element in the sequence has a particular position, and the arrangement of these elements carries significant information. Sequence modeling methods have been extensively studied in the field of natural language processing and temporal data analysis [102, 103]. Recurrent Neural Network (RNN) and its variants are able to encode variable-sized sequence inputs to fixed-sized representations [102, 103]. Recently, attention-based models such as transformers have been effectively applied to a wide range of domains[100, 104].

4.3 Preliminaries

4.3.1 Notations

Graph A graph is represented as $\mathcal{G} = (\mathcal{V}, \boldsymbol{W}, \boldsymbol{H})$, with $\mathcal{V}$ as the set of graph nodes with $n = |\mathcal{V}|$, $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ as the adjacent matrix and $\boldsymbol{H} \in \mathbb{R}^{n \times d}$ as the node representations. We denote by $v_i \in \mathcal{V}$ as the node indexed at i and $\mathcal{N}_s(v_i)$ as the set of neighbors of v_i within s hops (We denote $\mathcal{N}(v_i) = \mathcal{N}_1(v_i)$. We use $\boldsymbol{h}_i$ as the node representation of node v_i and $\boldsymbol{h}_{\mathcal{G}}$ as the graph-level representation.

Permutation We denote a permutation over integers from 1 to n as a list of node indices $\boldsymbol{\pi} = \{v_1, v_2, ..., v_n\}$. The corresponding permutation matrix is denoted as $\boldsymbol{P}_{\boldsymbol{\pi}}$. We use $\boldsymbol{\Pi}$ to denote the set of all possible permutations with $|\boldsymbol{\Pi}| = n!$. $\boldsymbol{\pi}^{-1}$ is denoted as the inverse permutation of $\boldsymbol{\pi}$. A graph permutated with $\boldsymbol{\pi}$ is denoted as $\mathcal{G}_{\boldsymbol{\pi}} = (\mathcal{V}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}, \boldsymbol{H}_{\boldsymbol{\pi}})$ where the node indices, weight matrix and feature matrix are permutated over $\boldsymbol{\pi}$ denoted as $\mathcal{V}_{\boldsymbol{\pi}}$, $\boldsymbol{W}_{\boldsymbol{\pi}} = \boldsymbol{P}_{\boldsymbol{\pi}} \boldsymbol{W} \boldsymbol{P}_{\boldsymbol{\pi}}^T$ and $\boldsymbol{H}_{\boldsymbol{\pi}} = \boldsymbol{P}_{\boldsymbol{\pi}} \boldsymbol{H}$.

Sequence We denote a sequence as $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ where the elements are constructed from the graph $\mathcal{G}$ and the node permutation $\boldsymbol{\pi}$. The elements x_i contains the node information of v_i as well as other structural information such as the edge information between v_i and v_{i+1}. We use $\boldsymbol{X} \in \mathbb{R}^{n \times D}$ to denote the sequence where $\boldsymbol{X}_{i,:}$ contains the element at position i.

The relationships of the graph, permutation and sequence is captured in Figure 4.1. We use a function **SeqDec** : $(\mathcal{G}, \boldsymbol{\pi}) \to \mathbb{R}^{n \times D}$ to denote the relationship as

$$\boldsymbol{X}_{\boldsymbol{\pi}} = \textbf{SeqDec}(\mathcal{G}, \boldsymbol{\pi}) \tag{4.1}$$

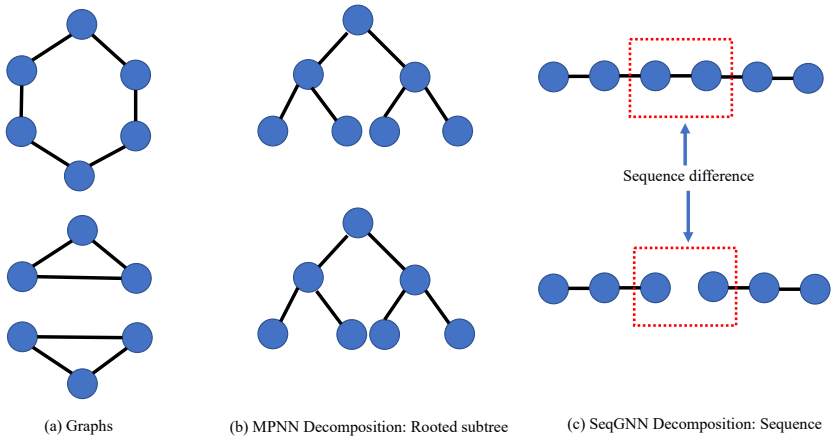


Figure 4.3: Example: (a) Regular graphs that cannot be distinguished by MPNN and 1-WL; (b) With k layers of message passing, node representations of MPNN models contain information from nodes that are k hops away. The information flow to compute the node representations can be represented as the rooted subtrees where root node absorb the information flowed from the leaf nodes, as shown in the figure [1, 2]; Due to the anonymity of graph nodes, the subtrees are indistinguishable for regular graphs. (c) SeqGNN formulates the graph as the composition of sequence representations with node permutation and edge information. Despite the anonymity of graph nodes, the sequences can still distinguish two graphs because of differences in the edge patterns.

4.3.2 Graph and Sequence Models

Graph Neural Network

GNN models compute the graph representations from the node features $\boldsymbol{H}$ and graph structures $\boldsymbol{W}$, denoted as $\textbf{GNN} : (\mathbb{R}^{n\times d}, \mathbb{R}^{n\times n}) \to \mathbb{R}^d$. In particular, MPNN, one of the most popular GNN models, iteratively updates node representations by aggregating information from neighboring nodes and the graph representation is obtained by applying the pooling function on the corresponding node representations [4, 6, 9],

$$\textbf{MPNN}(\boldsymbol{H}, \boldsymbol{W}) = \textbf{Pool}(\boldsymbol{h}_v, v \in \mathcal{V}) \tag{4.2}$$

where $\boldsymbol{h}_u^i$ is the representation of node u at iteration i and **Msg** is the message function that aggregates neighborhood information.

$$\begin{aligned} \boldsymbol{h}_v^{(i)} &= \textbf{Aggregate}^{(i)}(\boldsymbol{h}_v^{(i-1)}, \boldsymbol{m}_v^{(i)}), \\ \boldsymbol{m}_v^{(i)} &= \textbf{Msg}^{(i)}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}(v)\}) \end{aligned} \tag{4.3}$$

Sequence Model We define a sequence model as **Seq** : $\mathbb{R}^{n \times D} \to \mathbb{R}^d$ that takes a variable-length sequence input and output a fixed-length representation. In this work, we use Long Short-Term Memory (LSTM) as the main sequence function [96, 103].

Permutation Invariance *Permutation invariance* is one of the key properties of GNN models, that is, the graph representations remain invariant under any permutation applied on the nodes.

$$\textbf{GNN}(\boldsymbol{H}, \boldsymbol{W}) = \textbf{GNN}(\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}), \forall \boldsymbol{\pi} \in \boldsymbol{\Pi}. \tag{4.4}$$

Note that sequence models do not necessarily follow the permutation invariance property.

4.4 Designing Graph Neural Networks with Sequence Modeling

4.4.1 Overview

The core design of SeqGNN models is to represent the graph as the compositions of sequence representations as

$$\textbf{SeqGNN}(\boldsymbol{H}, \boldsymbol{W}) = \rho\left(\sum_{\boldsymbol{\pi} \in \Pi} \boldsymbol{p}_{\boldsymbol{\theta}}(\boldsymbol{\pi} | \boldsymbol{H}, \boldsymbol{W}) \cdot \textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}})\right) \tag{4.5}$$

where $\boldsymbol{p_\theta}(\boldsymbol{\pi}|\boldsymbol{H}, \boldsymbol{W})$ is the weight associated with parameters $\boldsymbol{\theta}$ and ρ is a non-linear function such as **MLP**. In this work, we normalize the total weights to 1 so that the graph representation is the nonlinear function applied to the expectation of the sequence function results as

$$\textbf{SeqGNN}(\boldsymbol{H}, \boldsymbol{W}) = \rho\left(\mathbb{E}_{\boldsymbol{\pi}\sim\boldsymbol{p_\theta}}\textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}})\right) \tag{4.6}$$

In the next, we present the key components that form the SeqGNN model. The overall framework is illustrated in Figure 4.2.

4.4.2 Express Graphs as Combinations of Sequences

One of the main components is to decompose the graph into a set of sequences $\boldsymbol{X}_{\boldsymbol{\pi}}$ and the associated weights $\boldsymbol{p_\theta}(\boldsymbol{\pi}|\boldsymbol{H}, \boldsymbol{W})$. The goal is to design the decomposition such that the resulting sequences and weights can maximally preserve the original graph information while maintaining the property of permutation invariance.

We start with the following proposition on the conditions of permutation invariance

Proposition 4.4.1 (Permutation Invariance)**.** ***SeqGNN***$(\boldsymbol{H}, \boldsymbol{W})$ *is permutation invariant if the sequence weights satisfies*

$$\boldsymbol{p_\theta}(\boldsymbol{\pi}\boldsymbol{\pi}'|\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}) = \boldsymbol{p_\theta}(\boldsymbol{\pi}'|\boldsymbol{H}, \boldsymbol{W}), \forall \boldsymbol{\pi}, \boldsymbol{\pi}' \in \boldsymbol{\Pi}. \tag{4.7}$$

Proof. The permutation-invariant pooling function on the permutated graph inputs $\mathcal{G}_{\boldsymbol{\pi}}$ can be

expressed as,

$$
\begin{aligned}
&\textbf{SeqGNN}(\boldsymbol{H}_\pi, \boldsymbol{W}_\pi) \\
&= \rho\left(\sum_{\pi'' \in \Pi} \boldsymbol{p}_{\boldsymbol{\theta}}(\pi''|\boldsymbol{H}_\pi, \boldsymbol{W}_\pi)\textbf{Seq}(\boldsymbol{X}_{\pi\pi''})\right) \\
&= \rho\left(\sum_{\pi' \in \Pi} \boldsymbol{p}_{\boldsymbol{\theta}}(\pi^{-1}\pi'|\boldsymbol{H}_\pi, \boldsymbol{W}_\pi)\textbf{Seq}(\boldsymbol{X}_{\pi'})\right) \\
&(\text{ Let } \pi' = \pi\pi'', \text{ then } \pi'' = \pi^{-1}\pi') \\
&= \rho\left(\sum_{\pi' \in \Pi} \boldsymbol{p}_{\boldsymbol{\theta}}(\pi'|\boldsymbol{H}, \boldsymbol{W})\textbf{Seq}(\boldsymbol{X}_{\pi'})\right) \\
&(\text{by the assumption}) \\
&= \textbf{SeqGNN}(\boldsymbol{H}, \boldsymbol{W})
\end{aligned}
$$

□

The proposition only requires the conditions on the weights of sequence without any assumptions of the sequence function. Guided by the proposition, we design the following sequence modeling method.

4.4.3 Sequence Modeling

Sequences of graph nodes and the associated edges are generally considered as the non-repeating node paths sampled from the graph. Random-walk based traversal algorithms have been proposed to sample node neighbors to learn effective node representations [105, 106, 107].

A major challenge is to design the sequence weights such that SeqGNN remains permutation invariant to random permutations. One trivial solution is to asssign equal probability to every possible sequence, which is the Relational Pooling (RP-GNN) formulation proposed by Murphy

Table 4.1: Comparison between MPNN and SeqGNN

Model	MPNN	SeqGNN
Basic component	Subtree	Sequence
Weight of components	Equal	Unequal
# of Components	n	# of samples
Permutation Invariance	yes	yes
Can distinguish regular graphs?	no	yes
Complexity	$O(nk)$	O(# of samples)

et al. [97, 98],

$$\textbf{RP-GNN}(\boldsymbol{H}, \boldsymbol{W}) = \rho\left(\frac{1}{n!}\sum_{\boldsymbol{\pi}\in\Pi}\textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}})\right) \tag{4.8}$$

However, the naive formulation requires iterating over all possible node permutations. Next, we will present the sequence modeling method that efficiently builds the permutation-invariant GNN models leveraging node information and specific graph structures.

We provide a general sequence model that generates a wide range of sequences as well as the associated weights satisfying Proposition 4.4.1.

The sequence path starts with equal probability over all possible nodes, the next node is selected from the unvisited neighbors with probability $1-\epsilon$ and other unconnected nodes with probability ϵ.

$$\boldsymbol{p}(v_{i+1}|v_i) = \begin{cases} (1-\epsilon)/|\mathcal{N}(v_i)| & \text{if } \mathcal{N}(v_i) > 0 \\ \epsilon/(n-i) & \text{otherwise} \end{cases} \tag{4.9}$$

When ϵ is small, the sequence models can be approximated by the algorithm given in Algorithm 3.

Algorithm 3 Sequence Modeling from Graphs

Input: $\mathcal{G} = (\mathcal{V}, \boldsymbol{W}, \boldsymbol{H})$
Output: sequence $\boldsymbol{x} = \{x_1, x_2, ..., x_n\}$ and the associated weight $\boldsymbol{p}$

$v_1 \sim$ Uniform$(1, n)$, unvisited_list = [1, 2, ..., n]
$p = 1/n$
Remove v_1 from unvisited_list
for i = 2, ..., n **do**
    Remove visited elements from $\mathcal{N}(v_{i-1})$
    if $\mathcal{N}(v_{i-1})$ is not null **then**
        $v_i \sim$ Uniform$(\mathcal{N}(v_{i-1}))$,
        $p = p \cdot 1/|\mathcal{N}(v_{i-1})|$
    else
        $v_i \sim$ Uniform(unvisited_list)
        $p = p \cdot 1/|$unvisited_list$|$
    end if
    Remove v_i from unvisited_list
end for
Generate $\boldsymbol{x} =$ **SeqDec**$(\mathcal{G}, \boldsymbol{\pi})$
Return $\boldsymbol{x}$ and $\boldsymbol{p}$

4.4.4 Expressive Power of SeqGNN

As shown in Figure 4.3, SeqGNN models have the expressive power to distinguish pairs of regular graphs which MPNN cannot distinguish. We assume the the sequence function is injective, then we have the following results,

Proposition 4.4.2. *Suppose two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ can be distinguished by MPNN, there exist a SeqGNN model that can distinguish the graphs.*

Proof. As pointed in Xu et al.'s work, graph representations from MPNN models are formed as the composition of subtrees. Supposing the multiset of subtree of $\mathcal{G}_1$ and $\mathcal{G}_2$ are $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ respectively, then at least two subtrees are different in $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$.

Since the subtrees are of the same heights (the number of graph convolutional layers), we have the following situations,

- At least one of the nodes are different at the certain level
- Edge information is different
- Different number of nodes at the certain level

All situations will lead to different sequences or (and) different weights based on Algorithm 1. As we assume the sequence function and pooling functions are injective, there exist a SeqGNN models can distinguis the two different sequence multiset (and the associated weights). □

The proposition formally states that SeqGNN models are at least as powerful as MPNN models. Combined with the case for regular graphs, we conclude that SeqGNN models are more powerful than MPNN models.

4.4.5 Complexity

The complexity of SeqGNN depends on the specific graph structures and the sequence modeling methods. For certain structures such as ring graphs or line graphs, the number of possible sequences is $O(n)$ while for other graphs such as complete graphs, the number of possible sequences could be $O(n!)$.

In the following, we present several practical techniques to improve the empirical performance in real-world applications.

4.4.6 Practical Techniques

4.4.6.1 Sequence Sampling

In practice, it is often infeasible to enumerate all possible sequences for graphs. Therefore, we propose to use the sequence sampling to approximate SeqGNN formulation. Suppose the

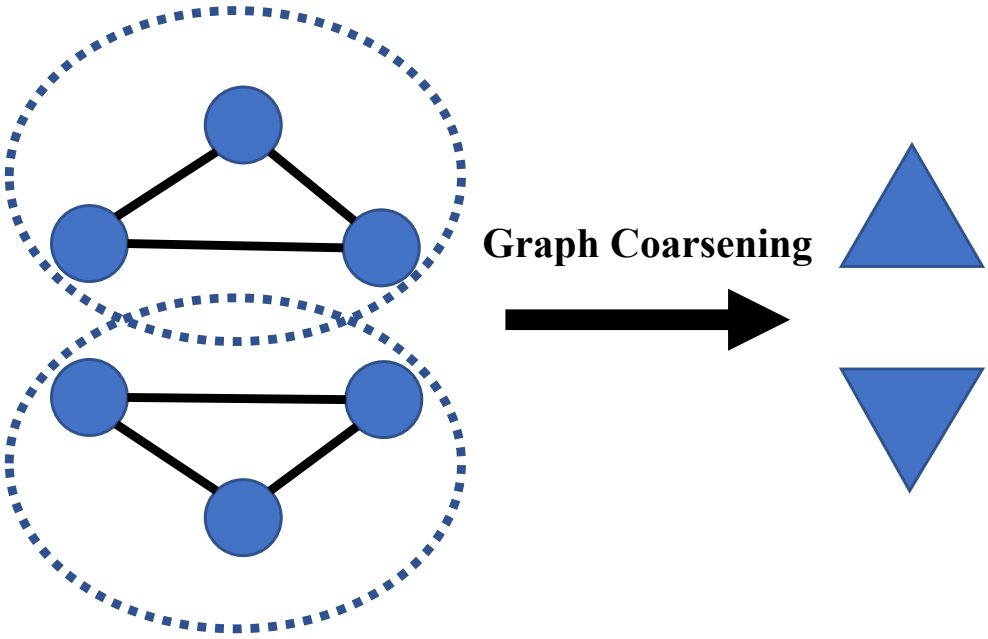


Figure 4.4: Graph coarsening techniques to reduce the complexity

.

number of samples is K and each permutation sample is $\boldsymbol{\pi}_i$ following Algorithm 3, then we rewrite the SeqGNN formulation as

$$\begin{aligned}\textbf{SeqGNN}'(\boldsymbol{H}, \boldsymbol{W}) &= \rho\left(\mathcal{E}_{\boldsymbol{\pi}\sim \boldsymbol{p_\theta}} \textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}})\right) \\ &= \rho\left(\frac{1}{K}\sum_{i=1}^{K} \textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}_i})\right) \\ &\approx \frac{1}{K}\cdot\sum_{i=1}^{K} \rho\left(\textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}_i})\right)\end{aligned} \tag{4.10}$$

Then we use the averaged sample output to approximate the SeqGNN representations.

4.4.6.2 Combine Message Passing and SeqGNN

SeqGNN models presented here can work directly on the raw graph information to form graph-level representations. In practice, the combination of message-passing and SeqGNN can result in improved empirical results. As we will show later, SeqGNN with message passing layers can outperform regular MPNN models with traditional pooling functions.

4.4.6.3 Graph Coarsening

Graph coarsening is one of the most commonly used techniques to reduce the graph size while preserving the key properties of graphs. To reduce the complexity, one approach is to encode certain graph substructures into units so that the sequences include the substructures as a whole structure. An example is illustrated in Figure 4.4.

4.4.6.4 Automorphism

The graph automorphisms are node permutations that preserves the graph structures, i.e. the isomorphisms from a graph to itself. The graph automorphisms capture the internal symmetries within graphs. Thus the sequence and the probability associated with the sequence starting with node i will be the same as j when node i and j are mapped in the graph automorphism. Thus the graph automorphic structures can significantly reduce the repeated weight assignment to similar sequences.

Table 4.2: Results (measured by accuracy: %) on TUDataset.

Model	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB
GIN [1]	76.2 ± 2.8	82.7 ± 1.7	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9
DGCNN [108]	75.5 ± 0.9	74.4 ± 0.5	70.0 ± 0.9	47.8 ± 0.9	73.8 ± 0.5
IGN [109]	76.6 ± 5.5	74.3 ± 2.7	72.0 ± 5.5	48.7 ± 3.4	78.4 ± 2.5
PPGN [41]	77.2 ± 4.7	83.2 ± 1.1	73.0 ± 5.8	50.5 ± 3.6	80.7 ± 1.7
CLIP [110]	77.1 ± 4.4	N/A	76.0 ± 2.7	52.5 ± 3.0	N/A
SIN [111]	76.5 ± 3.4	82.8 ± 2.2	75.6 ± 3.2	52.5 ± 3.0	N/A
CIN [112]	77.0 ± 4.3	83.6 ± 1.4	75.6 ± 3.7	52.7 ± 3.1	N/A
SeqGNN (Ours)	$\mathbf{77.8 \pm 2.6}$	$\mathbf{83.7 \pm 1.1}$	75.3 ± 3.4	$\mathbf{52.8 \pm 2.9}$	$\mathbf{80.6 \pm 1.2}$

4.4.7 Training and Inference

We consider the graph classification problem: given a set of graph samples $\mathcal{D} = \{(\mathcal{G}_1, \boldsymbol{y}_1), (\mathcal{G}_2, \boldsymbol{y}_2), ..., (\mathcal{G}_N, \boldsymbol{y}_N)\}$ where $\boldsymbol{y}_i \in \mathbb{Y}$ is the label of graph $\mathcal{G}_i$. The objective is to minimize the empirical loss as,

$$\begin{aligned} \min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) &= \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{y}_i, \textbf{SeqGNN}(\boldsymbol{H}_i, \boldsymbol{W}_i)) \\ &= \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{y}_i, \rho(\mathbb{E}_{\boldsymbol{\pi}}[\textbf{Seq}(\boldsymbol{X}_{i,\boldsymbol{\pi}})])) \end{aligned} \tag{4.11}$$

where $\boldsymbol{\theta}$ denotes the set of parameters in the **GNN** model and L is the loss function such as the cross entropy loss.

Training The objective is a standard stochastic optimization with learnable parameters $\boldsymbol{\theta}$ and random variable $\boldsymbol{\pi}$. We use the stochastic gradient descent to find the optimal parameters θ^* [97, 113, 114]. At step t, we uniformly sample a mini-batch of example graphs as $\mathcal{B} = \{(\mathcal{G}'_{(1)}, \boldsymbol{y}'_{(1)}), (\mathcal{G}'_{(2)}, \boldsymbol{y}'_{(2)}), ..., (\mathcal{G}'_{(b)}, \boldsymbol{y}'_{(b)})\}$ from the training set and the gradient is computed

as

$$\begin{aligned} \boldsymbol{g}_t &= \frac{1}{b}\sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}_i', \textbf{SeqGNN}(\mathcal{G}_i')) \\ &= \frac{1}{b}\sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}_i, \rho(\textbf{Seq}(\boldsymbol{X}_{i,\boldsymbol{\pi}_i}))) \end{aligned} \tag{4.12}$$

where $\boldsymbol{\pi}_i$ is the random permutation sampled following Algorithm 3.

We update the parameters by the following,

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{g}_t$$

where $\eta_t \in (0, 1)$ is the learning rate at step t with $\lim_{t\to\infty} \eta_t = 0$, $\sum \eta_t = \infty$ and $\sum \eta_t^2 < \infty$. Note that the algorithm is a standard stochastic optimization algorithms used in training neural networks.

The above stochastic gradient descent essentially optimizes the following modified objective with expectation outside the function L and ρ as in [97],

$$\min_{\boldsymbol{\theta}} \tilde{\mathcal{L}}(\mathcal{D}; \boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{\pi}}[L(\boldsymbol{y}_i, \rho(\textbf{Seq}(\boldsymbol{X}_{i,\boldsymbol{\pi}})))] \tag{4.13}$$

When the loss function L is convex and ρ is the identity function, the modified objective function is an upper bound of the original loss function following Jensen's inequality [97]. We denote $\boldsymbol{\theta}^*$ as the optimal parameters of the optimization problem 4.13. Similar to the analysis of stochastic gradient descent, the parameters $\boldsymbol{\theta}_t$ converges to $\boldsymbol{\theta}^*$ with probability 1 under mild conditions [97, 115].

Inference Assuming $\boldsymbol{\theta}^*$ are the optimal parameters, the output $\hat{\boldsymbol{y}}$ is estimated as the average of the predicted sample outputs as,

$$\hat{\boldsymbol{y}} = \frac{1}{n'}\sum_{i=1}^{n'} \textbf{SeqGNN}(\boldsymbol{H}, \boldsymbol{W}) = \frac{1}{n'}\sum_{i=1}^{n'} \rho\left(\textbf{Seq}(\boldsymbol{X}_{\boldsymbol{\pi}_i})\right) \tag{4.14}$$

where n' is the number of inference samples with $\boldsymbol{\pi}_i$ as the random permutation sampled from Algorithm 3.

4.5 Experiments

In the experiments, we evaluate the SeqGNN model on real-world benchmark datasets. All datasets have been commonly used to evaluate the effectiveness of GNN models [6, 39, 96]. The codes are publicly available [1].

4.5.1 Datasets

The datasets contain 5 real-world benchmarks from TUDataset. PROTEINS and NCI1 are bioinformatics datasets; IMDB-BINARY, IMDB-MULTI, and COLLAB are social network datasets. We follow the standard steps to preprocess the datasets. Specifically, the node features of bioinformatics graphs are categorical node labels, and the node features of social networks are node degrees. More details about the datasets can be found in the supplementary material.

[1]

4.5.2 Baseline models

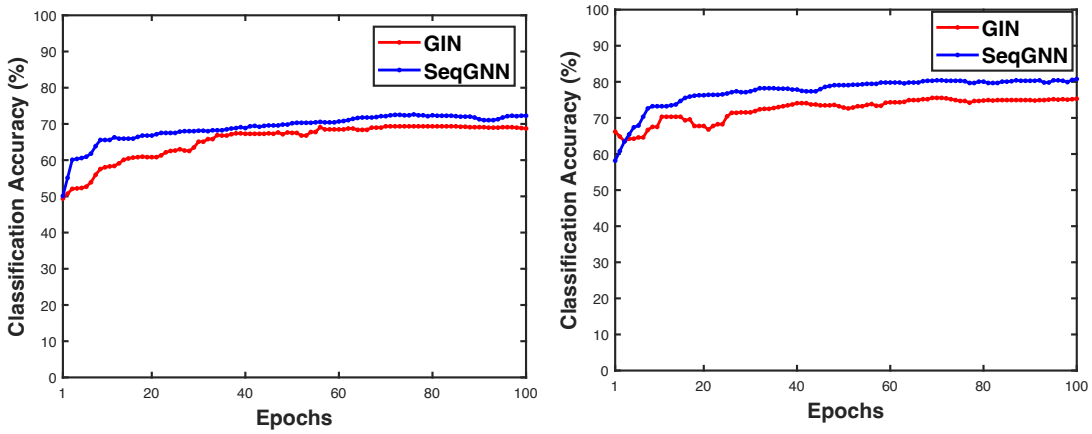
We compare SeqGNN models with the following state-of the art graph learning models: Graph Isomorphism Network (GIN)[6], Deep Graph Convolutional Neural Network (DGCNN) [108], Provably Powerful Graph Network (PPGN) [41],Principal Neighbourhood Aggregation (PNA) [116],Colored Local Iterative Procedure (CLIP) [110],Simplicial Isomorphism Network (SIN) [111], and Cell Isomorphism Network (CIN) [112].

4.5.3 Model Configuration

We use 10-fold cross validation and report the average classification accuracy and standard deviation. For SeqGNN models, we use LSTM as the main sequence function. We select the number of sequence samples as 10. The hyper-parameters are chosen by our model selection procedure as follows. For all datasets, 3 or 5 GNN layers (including the input layer) are applied, and the LSTMs are used as the sequence functions. Batch normalization [117] is applied to every hidden layer. All models are initialized using Glorot initialization [118] and trained using the Adam SGD optimizer [119] with an initial learning rate of 0.001. The learning rate is decayed by a factor of 0.5 every 50 epochs. The training is stopped when the number of epochs reaches the maximum value of 400.

4.5.4 Main Results

Table 4.2 show the experiment results for the graph classification and regression tasks. For all datasets, SeqGNN models achieve comparable or superior performance. Empirical results indicate that the SeqGNN models effectively capture the key graph properties that are useful for



(a) $k = 1$ (b) $k = 2$

Figure 4.5: Classification accuracy under different number of convolutional layers.

classification tasks.

4.5.5 Analysis of Expressive Power

In the experiment, we empirically compare the model performance under the same number of convolutional layers. Note that GIN models are proved to be the most expressive models under the message passing framework [1].

Figure 4.5 shows the plots of classification accuracy under the number of layers 1 and 2 for the NCI1 dataset. For both models, the classification accuracy improves as the number of the convolutional layer increases. However, under each setting, SeqGNN models constantly outperform GIN. In particular for $k = 1$, the MPNN models cannot learn much meaningful information due to its limited expressive power while SeqGNN models can still achieve 75% classification accuracy. The experiment validate the results in Proposition 4.4.2 that the SeqGNN models can represent and learn more complex graph functions even with small number of convolutional layers.

4.6 Conclusion

In this chapter, we propose a novel graph neural network paradigm SeqGNN that provably improves the expressiveness of GNN models. By formulating the graph representation as the compositions of sequence representations, SeqGNN models can distinguish graphs that MPNN cannot distinguish. Through the design of sequence modeling methods, SeqGNN achieves strong expressive power while maintaining the permutation invariance properties. Empirical results show that SeqGNN models achieve competitive performance on graph classification benchmarks. We further show that combining message passing and SeqGNN achieves better performance compared with traditional MPNN models.

Chapter 5: Fast Spectral Clustering in GPU-CPU platforms

5.1 Introduction

Spectral clustering algorithm has recently gained popularity in handling many graph clustering tasks such as those reported in [120, 121, 122]. Compared to traditional clustering algorithms, such as k-means clustering and hierarchical clustering, spectral clustering has a very well formulated mathematical framework and is able to discover non-convex regions which may not be detected by other clustering algorithms. Moreover, spectral clustering can be conveniently implemented by linear algebra operations using popular scientific software environments such as Matlab and Python. Most of the available software implementations are built upon CPU-optimized Basic Linear Algebra Subprograms (BLAS), usually accelerated using multi-thread programming. However, such implementations scale poorly as the problem size or the number of clusters grow very large. Recent results show that GPU accelerated BLAS significantly outperforms multi-threaded BLAS libraries such as the Intel MKL package, LAPACK and Goto BLAS [123, 124]. Moreover, hybrid computing environments, which collaboratively combine the computational advantages of GPUs and CPUs, further boost the overall performance and are able to achieve very high performance on problems whose sizes grow up to the capacity of CPU memory [125, 126, 127, 128, 129, 130]. In this chapter, we present a hybrid implementation of the spectral clustering algorithm which significantly outperforms the known implementations, most of which are purely based on multi-

core CPUs.

There have been reported efforts on parallelizing the spectral clustering algorithm. Zheng et al. [131] presented both CUDA and OpenMP implementations of spectral clustering. However, the implementation was targeted for a much smaller data size than the work in this chapter, and moreover, their implementation achieve a relatively limited speedup. Matam et al. [132] implemented a special case of spectral clustering, namely the spectral bisection algorithm, which was shown to achieve high speed-ups compared to Matlab and Intel MKL implementations. Chen et al. [133, 134] implemented the spectral clustering algorithm on a distributed environment using Message Passing Interface (MPI), which is targeted for problems whose sizes that could not fit in the memory of a single machine. Tsironis and Sozio [135] proposed an implementation of spectral clustering based on MapReduce. Both implementations were targeted for clusters, and involve frequent data communications which will clearly constrain the overall performance.

In this chapter, we present a hybrid implementation of spectral clustering on a CPU-GPU heterogeneous platform which significantly outperforms all the best implementations we are aware of, which are based on existing parallel platforms. We highlight the main contributions as follows:

- Our algorithm is the first work to comprehensively explore the hybrid implementation of spectral clustering algorithm on CPU-GPU platforms.
- Our implementation makes use of sparse representation of the corresponding graphs and can handle extremely large input sizes and generate a large number of clusters.
- The hybrid implementation is highly efficient and is shown to make a very good use of available resources.
- Our experimental results show superior performance relative to the common scientific

software implementations

The rest of the chapter is organized as follows. Section II gives an overview of the spectral clustering algorithm, while describing the important steps in some detail. Section III describes the operating environment and the necessary software dependencies. Section IV provides a description of our parallel implementation, while Section V evaluates the performance of our algorithm with a comparison with Matlab and Python implementations on both synthetic and real-world datasets. The codes are available on https://github.com/yuj-umd/fastsc.

5.2 Overview of Spectral Clustering Algorithm

Spectral clustering was first introduced in 1973 to study the graph partition problem [136]. Later, the algorithm was extended in [20, 137], and generalized to a wide range of applications, such as computational biology [138, 139], medical image analysis [121, 122], social networks [140, 141] and information retrieval [142, 143]. A standard procedure of the spectral clustering algorithm to compute k clusters is described next [19],

- Step 1: Given a set of data points $x_1, x_2, ..., x_n \in \mathbb{R}^d$ and some similarity measure $s(x_i, x_j)$, construct a sparse similarity matrix W that captures the significant similarities between the pairs of points.
- Step 2: Compute the normalized graph Laplacian matrix as $L_n = D^{-1}L$ where L is the unnormalized graph Laplacian matrix defined as $L = D - W$ and D is the diagonal matrix with each element $D_{i,i} = \sum_{j=1}^{n} W_{i,j}$.
- Step 3: Compute the k eigenvectors of the normalized graph Laplacian matrix L_n corresponding to the smallest k nonzero eigenvalues.

- Step 4: Apply the k-means clustering algorithm on the rows of the matrix whose columns are the k eigenvectors to obtain the final clusters.

Given the similarity graph defined by the similarity matrix W, the basic idea behind spectral clustering is to partition the graph into k partitions such that some measure of the cut between the partitions is minimized. The traditional graph cut is defined as follows:

$$\text{Cut}(A_1, A_2, ..., A_k) = \frac{1}{2}\sum_{i=1}^{k} W(A_i, \bar{A}_i); \tag{5.1}$$

$$W(A, \bar{A}) := \sum_{i \in A, j \in \bar{A}} w_{ij} \tag{5.2}$$

To ensure that the each partition represents a meaningful cluster of reasonable size, two alternative cut measures are often used, namely **RatioCut** and normalized cut **Ncut**. Note that we use below $|A_i|$ as the number of nodes in A and $vol(A)$ as the sum of the degrees of all the nodes in A.

$$\text{RatioCut}(A_1, A_2, A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|}; \tag{5.3}$$

$$\text{Ncut}(A_1, A_2, A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{vol(A_i)}; \tag{5.4}$$

In our implementation, we focus on the problem of minimizing the **Ncut** which has an equivalent algebraic formulation as defined next.

$$\min_{H} \operatorname{trace}(H'LH) \text{ subject to } H'DH = I \tag{5.5}$$

That is, we need to determine a matrix $H \in \mathbb{R}^{n \times k}$ whose columns are indicator vectors, which minimizes the objective function introduced above.

Since this problem is NP-hard, we relax the discrete constraints on H are removed, thereby allowing H to be any matrix in $\mathbb{R}^{n \times k}$. Note that there is no theoretical guarantee on the quality of the solution of the relaxed problem compared to the exact solution of the discrete version. It turns out that the relaxed problem is a well-known trace minimization problem, which can be exactly solved by taking H as the eigenvectors with the smallest k eigenvalues of the matrix $L_n = D^{-1}L$ or equivalently the k generalized eigenvectors corresponding to the smallest k eigenvalues of $Lx = \lambda Dx$. The k-means clustering is then applied on the rows of H to obtain the desired clustering.

The algorithm described above begins with a set of d-dimensional data points and builds the similarity graph explicitly from the pair-wise similarity metric. The similarity graph is usually stored in a sparse matrix representation, which often reduces the memory requirement and computational cost to linear instead of quadratic. For the general graph clustering whose input is specified as a graph, our spectral clustering algorithm starts directly in Step 2. Otherwise, we build our sparse graph representation from the given set of data points.

5.3 Environment Setup

5.3.1 The Heterogeneous System

The CPU-GPU heterogeneous system used in our implementation is specified in Table 5.1.

The CPU and the GPU communicate through the PCIe bus whose theoretical peak bandwidth is 8 GB/s. The cost of data communication can be quite significant for large-scale problems. To

Table 5.1: CPU and GPU specifics

CPU Model	Intel Xeon E5-2690
CPU Cores	8
DRAM Size	128GB
GPU Model	Tesla K20c
Device Memory Size	5GB GDDR5
SMs and SPs	13 and 192
Compute Capability	3.5
CUDA SDK	7.5
PCIe Bus	PCIe x16 Gen2

achieve the best overall performance, our implementation leverages the GPU to compute the most computationally expensive part while minimizing the data transfer between the host and the device.

5.3.2 CUDA Platform

CUDA is a general-purpose multithreaded programming model that leverages the large number of GPU cores to solve complex data parallel problems. The CUDA programming model assumes a heterogeneous system with a host CPU and several GPUs as co-processors. Each GPU has an array of Streaming Multiprocessors (**SM**), each of which has a number of Streaming Processors (**SP**) that execute instructions concurrently. The parallel computation on GPU is invoked by calling customized kernel functions using thousands of threads. The kernel function is executed by blocks of threads independently. Each block of threads can be scheduled on any Streaming Multiprocessors (SP) as shown in Figure 5.1. The kernel function takes as parameters the number of blocks and the number of threads within a block.

In addition, NVIDIA provides efficient BLAS libraries for both sparse[1] and dense[2] matrix computations. Our implementation relies on the Thrust library, which resembles the C++ Standard Template Library (STL) that provides efficient operations such as sort, transform, which greatly improves productivity.

5.3.3 ARPACK Software

ARPACK is a software package designed to solve large-scale eigenvalue problems [144]. ARPACK is reliable and achieves high accuracy, and is widely used in modern scientific software environments. It contains highly optimized Fortran subroutines that are able to solve symmetric, non-symmetric and generalized eigenproblems. ARPACK is based on the Implicitly Restarted Arnoldi Method (IRAM) with non-trivial numerical optimization techniques [145, 146]. In our implementation, we adopt ARPACK++ [3] that provides C++ interfaces to the original ARPACK Fortran packages and utilizes efficient matrix solver libraries such as LAPACK, SuperLU. The eigenvalue problem is efficiently solved by collaboratively combining the interfaces of ARPACK++ and cuSPARSE library.

5.3.4 OpenBLAS

OpenBLAS[4] is an open-source CPU-based BLAS library utilized by ARPACK++. It supports multi-threaded acceleration through pthread programming or OpenMP by specifying corresponding environment variables. OpenBLAS is a highly optimized BLAS library developed based on

[1] http://docs.nvidia.com/cuda/cusparse/
[2] http://docs.nvidia.com/cuda/cublas/
[3] http://reuter.mit.edu/software/arpackpatch/
[4] http://www.openblas.net/

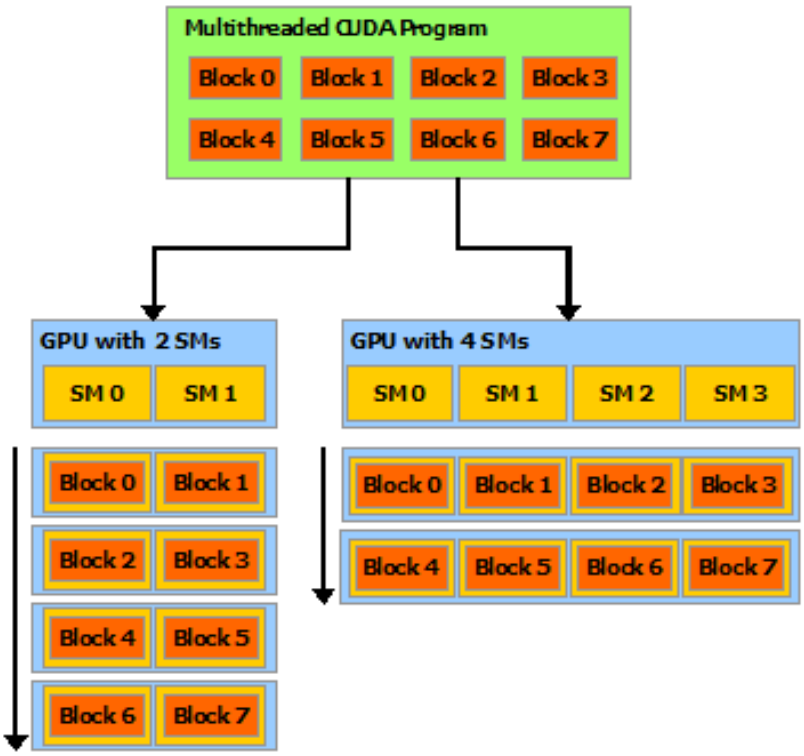


Figure 5.1: CUDA Program Model

GotoBLAS2, which has been shown to surpass other CPU-based BLAS libraries [123].

5.4 Implementation

5.4.1 Data Preprocessing

Given the d-dimensional data points, the preprocessing step constructs the similarity matrix from the data points. The clustering problem is reformulated as a graph clustering where the graph is represented by the similarity matrix.

As mentioned before, the similarity matrix is usually constructed to be sparse, which reduces the memory requirement and enables high computational efficiency. The sparsity patterns of the similarity matrices are highly dependent on the specific application. The following are

several common ways to construct a sparse similarity matrix [19].

- λ-threshold graph: The similarity graph is constructed where data points are connected if their similarity measure is above the threshold λ.
- ε-distance graph: The similarity matrix is construct by only connecting data points that are within a spatial distance ε.
- k-nearest-neighbor graph: The similarity graph is constructed where two data points x_i and x_j are connected only if either x_i is among the k most similar data points of x_j, or x_j is among the k most similar data points of x_i. Note that the parameter k is unrelated to the number k of clusters used in the next section.

The notion of the similarity measure between data points also varies depending on the application. Typical measures are the following.

- Cosine Similarity Measure

$$\text{CosineDist}(x_i, x_j) = \frac{\langle x_i, x_j \rangle}{\|x_i\|_2 \|x_j\|_2} \tag{5.6}$$

- Cross Correlation

$$\text{CrossCorr}(x_i, x_j) = \frac{\langle x_i - \bar{x_i}, x_j - \bar{x_j} \rangle}{\|x_i - \bar{x_i}\|_2 \|x_j - \bar{x_j}\|_2} \tag{5.7}$$

- Exponential decay function

$$\text{ExpDecay}(x_i, x_j) = e^{\frac{\|x_i - x_j\|_2}{2\sigma^2}} \tag{5.8}$$

Although the sparse patterns and similarity measures are different depending on the application, the general construction of the similarity matrix can be accelerated under the CUDA programming

Algorithm 4 Construction of Sparse Similarity Matrix

1: Transfer the input data X and edge lists E from CPU to GPU.
2: Initialize n-length vectors $X_{average}$ and X_{norm} on GPU.
3: Initialize nnz-length vector val on GPU.
4: Execute kernel function `compute_average` where each thread i computes $X_{average}(i) = \frac{1}{d}\sum_{j=1}^{d} X_{ij}$
5: Execute kernel function `update_data` where each thread i updates one row of data $X_{ij} = X_{ij} - X_{average}(i)$ and compute $X_{norm}(i) = \sqrt{\sum_{j=1}^{d} X_{ij}^2}$
6: Execute kernel function `compute_similarity` where each thread i computes the similarity between the i^{th} pair of data points in E.
7: The edge list and the vector val form the sparse graph represented in the Coordinate Format (COO) format.

model regardless of the preprocessing used. Here we provide a parallel implementation for a specific sparsity pattern and similarity measure.

We consider the input data as a matrix $X \in \mathbb{R}^{n\times d}$ where n is the number of data points and d is the dimension of each data point. The goal is to construct a sparse matrix representation of the similarity graph using the ε-distance graph structure and cross correlation as the similarity measure. We assume the neighborhood information is given by a list $E \in \mathbb{R}^{nnz\times 2}$, which contains all pairs of indices of data points that are within ε-distance. The number nnz of such pairs is the number of edges in the graph. The procedure for constructing the sparse similarity matrix represented in Coordinate Format (COO) format is described in Algorithm 4.

The above procedure is highly data parallel and easy to implement under the CUDA programming model. In general, there are two sparse matrix representations that we use in our work.

- Coordinate Format (**COO**): this format is the simplest sparse matrix representation. Essentially, COO uses tuples (i, j, w_{ij}) to represent all the non-zero entries. This can be done through three separate nnz-length arrays that respectively store the row indices, column indices, and the corresponding non-zero matrix values.

- Compressed Sparse Row Format (**CSR**): this consists of three arrays, one containing the non-zero values, the second containing the column indices of the corresponding non-zero values, and the third contains the prefix sums of the number of nonzero entries of the rows.

Other sparse formats such as Compressed Sparse Column Format (**CSC**), Block Compressed Sparse Row Format (**BSR**) are also supported in our implementation.

5.4.2 Parallel Eigensolvers

Given the similarity graph W represented in some sparse format and the desired number of clusters k, this step computes the k eigenvectors corresponding to the smallest k eigenvalues of normalized Laplacian $L_n = I - D^{-1}W$ where W is the sparse matrix and D is the diagonal matrix with each element $D_{i,i} = \sum_{j=1}^{n} W_{i,j}$. We assume that $D_{i,i}$ are all positive, otherwise the isolated nodes can be removed from the graph. The eigenvectors corresponding to the smallest k eigenvalues of the normalized Laplacian are exactly the eigenvectors corresponding to the largest k eigenvalues of $D^{-1}W$. Since computing the largest eigenvalues results in better numerical stability and convergent behavior, we focus our attention on computing the eigenvectors corresponding to the largest k eigenvalues of $D^{-1}W$.

The sparse matrix multiplication $D^{-1}W$ can easily be computed as follows:

$$\begin{bmatrix} d_{11}^{-1} & & & \\ & d_{22}^{-1} & & \\ & & \cdots & \\ & & & d_{nn}^{-1} \end{bmatrix} \times \begin{bmatrix} W_{1j} \\ W_{2j} \\ \cdots \\ W_{nj} \end{bmatrix} = \begin{bmatrix} d_{11}^{-1}W_{1j} \\ d_{22}^{-1}W_{2j} \\ \cdots \\ d_{nn}^{-1}W_{nj} \end{bmatrix} \tag{5.9}$$

Algorithm 2 Parallel Computation of $D^{-1}W$
1. Initialize a n-length vector x with 1.0 for all elements.
2. Compute the vector $y = Wx$ where each element $y_i = d_{ii}$ by calling `cusparseDcsrmv` in cuSPARSE library
3. Execute the kernel function `ScaleElements` where each thread i processes one item in COO format $< \text{r}, \text{c}, \text{val} >$ and scales the element value by the inverse of y_i.
4. Compress the row indices through the cuSPARSE interface `cusparseXcoo2csr`.
5. The compressed row indices, the column indices and the updated element value form the CSR representation of $D^{-1}W$

The corresponding computation is data parallel and has complexity O(nnz). We assume that the sparse similarity matrix initially resides in the device memory, represented in **COO** format. The parallel computation is described in Algorithm 2. Note that the $D^{-1}W$ will be transformed to the **CSR** format to perform the sparse matrix-vector multiplication at the next step.

An important feature of the ARPACK software is the **reverse communication interfaces**, which facilitate the process of solving large-scale eigenvalue problems. The reverse communication interfaces are CPU-based interfaces that encapsulate implicitly restarted Arnoldi/Lanczos method, which is an iterative method to obtain the required eigenvalues and corresponding eigenvectors. For each iteration, the interface provides a n-length vector used as input and the output of sparse matrix-vector multiplication is provided back to the interface. ARPACK interfaces combine the optimized Fortran routines and CPU-based BLAS library OpenBLAS, which is one of the most efficient CPU-based BLAS library. ARPACK provides the flexibility in choosing any matrix representation format and the function to obtain the results of matrix-vector multiplication. In our implementation, the matrix-vector multiplication is performed on the GPU. For each iteration, the input vector is transferred from the CPU to the GPU and the output vector is transfered back to the interface. The detailed implementation is shown in Algorithm 5.

Algorithm 5 Parallel Eigensolver

1: Initialize the object `Prob` with random parameters.
2: **while** `!Prob.converge()` **do**
3: `Prob.TakeStep()`.
4: Transfer the data located at `Prob.GetVector()` from host to device.
5: Call `cusparseDcsrmv` to perform matrix-vector multiplication on device.
6: Transfer the result from device to host and put it at the location addressed by `Prob.PutVector()`.
7: **end while**
8: Compute the eigenvectors by `Prob.FindEigenvectors()`.

The object `Prob` is initialized as the eigenvalue problem for the symmetric real matrix with the k largest-magnitude eigenvalues. `TakeStep()` is an interface that performs the necessary matrix operations based on the multi-threaded OpenBLAS library. For each iteration, the multiplication of sparse matrix and dense vector is computed on the GPU where 1) the sparse matrix is $D^{-1}W$ reside on GPU; 2) the input vector, whose location is indicated by `Prob.GetVector()`, is transferred from CPU to GPU; 3) the result is transfered back from GPU to CPU to the position `Prob.PutVector()`. After the object `Prob` reaches convergence, the eigenvectors are computed by `Prob.FindEigenvectors()`.

The complexity of Algorithm 3. largely depends on the interfaces `TakeStep()` and `FindEigenvectors()`. Both routines depend on the number m of Arnoldi/Lanczos vectors, which is usually set as $m = \max(n, 2k)$. `TakeStep()` involves the eigenvalue decomposition and iteratively QR factorization of $m \times m$ matrix, as well as a few dense matrix-vector multiplication. Therefore the complexity for `TakeStep()` is at least $(O(m^3) + O(nm) \times O(m-k))$. Moreover, the general complexity for sparse matrix-vector multiplication is $O(nnz \cdot m)$. The number of iteration # depends on the initial vector and properties of the matrix. The complexity `FindEigenvectors()`

is $O(nmk)$. Hence the overall complexity is,

$$(O(m^3) + O(nm^2) + O(nnz \cdot m)) \times \# + O(nmk) \tag{5.10}$$

As far as we know, the procedures described in Algorithm 3 are currently the most efficient and convenient way to solve general eigenvalue problems for large-scale matrices. We leverage the existing software ARPACK on CPU to perform the complex eigensolver procedures and the GPU to perform the expensive matrix computations. Results in Section V. will show that the data communication overhead is negligible compared to the overall computational cost and the overall implementation is very efficient compared to other software that relies on CPU-based sparse matrix-vector multiplication.

5.4.3 Parallel k-means clustering

The k-means clustering algorithm is an iterative algorithm to partition the input data points into k clusters whose objective function is to minimize the sum of squared distances between each point and its representative. In spectral clustering, the k-means algorithm is used to cluster the rows of the matrix consisting of the eigenvectors. Each such row can in fact be viewed as a reduced dimension representation of the original data point. There are several GPU-based implementations of the k-means clustering such as [147, 148]. However, none of these implementations seem to be efficient for large-scale problems, especially when k is very large. Our implementation is a revised version from an open-source project [5] which efficiently utilizes the Thrust and CUBLAS libraries and achieve significant speedups.

[5] https://github.com/bryancatanzaro/kmeans

Algorithm 6 Parallel K-means Algorithm

1: Transfer the data $V \in \mathbb{R}^{n \times d}$ from the CPU to the GPU.
2: Randomly select k points as the centroids of the k clusters stored in $C \in \mathbb{R}^{k \times d}$
3: **while** the centroids change **do**
4: Compute the pairwise distances $S \in \mathbb{R}^{n \times k}$ between data points and the centroids.
5: Update the new label of each data point.
6: Compute the new centroids of the clusters.
7: **end while**
8: Transfer the labeling result from GPU to CPU.

Algorithm 7 Parallel k-means++ Initialization

1: Pick the initial data point uniformly at random from 1 to n.
2: Initialize the n-length vector `Dist` where each element is the shortest distance between the data point v_i and the current centroids.
3: **for** i = 2 to k **do**
4: Compute the n-length vector P such that $P_j = \frac{\text{Dist}_j^2}{\sum_{l=1}^{n} \text{Dist}_l^2}$
5: Choose the i^{th} centroid as the data point x with probability P_x
6: Compute the vector `newDist` such that each i^{th} element as the distance between the data point v_i
7: and the new centroid
8: Update Dist $\text{Dist}_j = \text{minimum}(\text{Dist}_j, \text{newDist}_j)$
9: **end for**

We assume that the low-dimensional representation $V \in \mathbb{R}^{n \times k}$ initially resides in the CPU memory where n is the number of data points and k is the desired number of clusters. The implementation is described in Algorithm 6.

Step 2 is the most common way to initialize the centroids. However, we use a more effective initialization strategy, referred to as the k-means++ initialization, which has been shown to converge faster and achieve better results than the traditional k-means algorithm [149]. This initialization is simple to implement in parallel using basic routines in CUDA Thrust library, as described in Algorithm 7,

Step 3 in Algorithm 6 is the main loop that iteratively updates the labels of the data points and the corresponding centers of the clusters until convergence (or the maximum number of

iterations is reached). Given the data points $V \in \mathbb{R}^{n \times d}$ and centroids $C \in \mathbb{R}^{k \times d}$, the pair-wise distance matrix $S \in \mathbb{R}^{n \times k}$ is computed as follows.

$$S_{ij} = \sum_{l=1}^{d} (V_{il} - C_{jl})^2 \tag{5.11}$$

After expanding the right hand side, the distance matrix S can be expressed as

$$S_{ij} = \sum_{l=1}^{d} (V_{il})^2 + \sum_{l=1}^{d} (C_{jl})^2 - 2 \sum_{l=1}^{d} V_{il} C_{jl} \tag{5.12}$$

Hence, we compute two additional vectors $V_{norm} \in \mathbb{R}^{n \times 1}$ and $C_{norm} \in \mathbb{R}^{n \times 1}$,

$$V_{norm}(i) = \sum_{l=1}^{d} (V_{il})^2, \tag{5.13}$$

$$C_{norm}(j) = \sum_{l=1}^{d} (C_{jl})^2 \tag{5.14}$$

The matrix S can be initialized as the sum of the corresponding elements in V_{norm} and C_{norm}

$$S_{ij} = V_{norm}(i) + C_{norm}(j) \tag{5.15}$$

The pair-wise distance matrix S is then computed by level-3 BLAS function provided in the cuBLAS library.

$$S = S - 2VC^T \tag{5.16}$$

For each data point, the new label is updated by as the index of centroid which has the

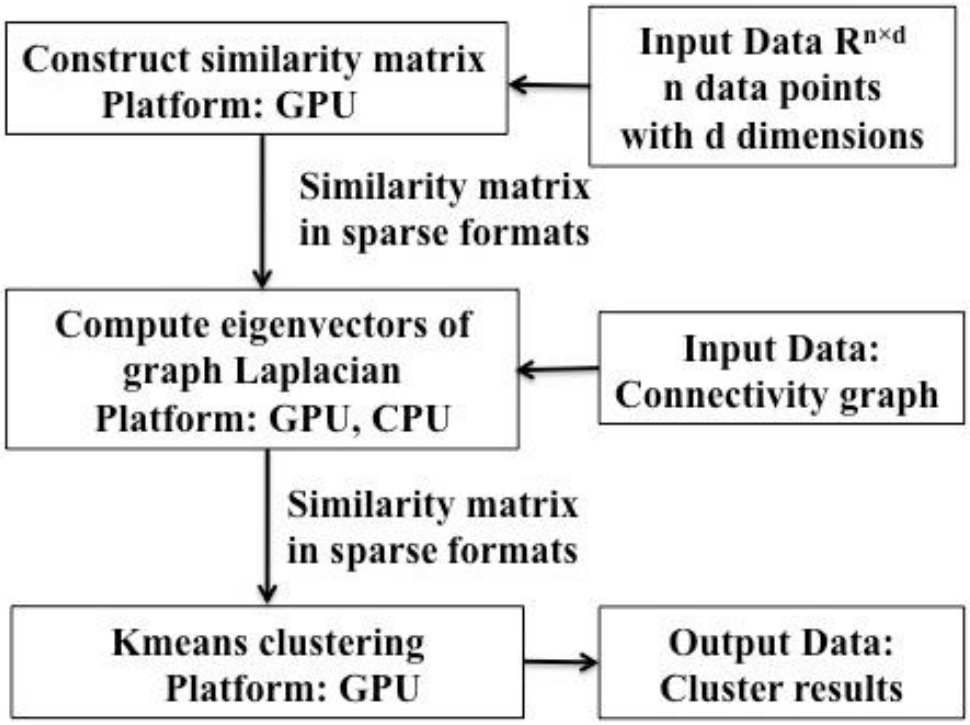


Figure 5.2: Parallel Implementation of Spectral Clustering

minimum distance to the data point. Meanwhile, a global variable is maintained to record the number of label changes during the update.

The new centroids are updated as the mean value of all the data points sharing the same label. To identify the points in each cluster, we sort the data points according to their new labels. Each GPU thread will then independently work on a consecutive portion of the sorted data points where most of these points share the same label.

The entire workflow of our implementation is summarized in Figure 5.2.

5.5 Evaluation

5.5.1 Datasets

We evaluate our parallel implementation on several real-world and synthetic datasets. The **Diffusion Tensor Imaging (DTI)** dataset is given as a set of data points, each of which is characterized by a 90-dimensional array. The other datasets are specified by an undirected graph data where the edges are given by an edge list. The problem sizes and the numbers of clusters generated are shown in Table 5.2. A brief description of each dataset is given next.

- **DTI**: The Diffusion Tensor Imaging(**DTI**) dataset is the brain image data of a subject chosen from a publicly accessible medical dataset provided by Nathan Kline Institute (NKI). The dataset captures the diffusion of the water molecules in the brain tissues, which can be used to deduce information about the fiber connectivity in the human brain. After preprocessing steps [121], the input data consists of $142K$ data points, each of which represents a 2mm$\times$2mm$\times$2mm brain voxel. The entire data points constitute the brain volume. Each data point is characterized by a 90-dimensional array representing the connectivity strength of the voxel to 90 brain regions (representing a segmentation of the grey matter). The task is to cluster the voxels that share similar connectivity profiles. To facilitate the construction of the similarity matrix, an edge list is provided which contains all pair of voxels that are within 4 millimeter distance.
- **FB**: This dataset is a dataset collected by a Facebook application. It contains the graph where each node represents an anonymous user and edges exist between users that share similar political interests[150].

Table 5.2: Datasets

Dataset	Nodes	Edges	Clusters
DTI	142541	3992290	500
FB	4039	88234	10
DBLP	317080	1049866	500
Syn200	20000	773388	200

- **DBLP**: This dataset consists of a comprehensive co-authorship network in a computer science bibliography. The nodes represent the authors. Authors are connected if they coauthored at least one publication[150]. The dataset contains more than 5000 communities. Here we set the number of clusters to 500 for experimental purposes.
- **Syn200**: The synthetic dataset is randomly generated by the **stochastic block model** [151]. The stochastic block model assumes that the data points are partitioned into r disjoint subsets, $C_1, C_2, ..., C_r$. A symmetric $r \times r$ matrix P is provided to model the inter-community edge probability. The synthetic sparse graph is randomly generated such that two nodes are connected with probability $p = 0.3$ if they are within the same cluster and $q = 0.01$ if they are in different clusters.

5.5.2 Environment and Software

The computing environment is a heterogeneous CPU-GPU platform with CPU and GPU specifics shown in Table 5.1. The software and packages used are as follows,

- **Matlab**: Matlab is a high-level language that provides interactive programing environment, which is widely used by scientists and engineers. The version of Matlab used for our implementation is 2015a. The sparse matrix representation and operations are the built-

in functions. The k-means clustering is the function in Statistical and Machine Learning toolbox.

- **Python**: Python software packages, such as Numpy, Scipy and sklearn, are popular tools to perform scientific computations. The version of Python binary for our implementation is 2.7.11. The sparse representation and functions to solve the eigenvalue problems are from *Scipy* package. The k-means clustering function is from *sklearn.cluster* module. The module versions are Numpy-1.10.4, Scipy-0.16.1 and sklearn-0.17 respectively.

Linear algebra and numeric functions are by default multi-threaded in Matlab on multicore and multiprocessor machines [6]. In addition, the Python packages are built on highly optimized CPU-based BLAS routines, some of which have been accelerated using multi-threaded programming.

5.5.3 Performance Analysis

We measure the running time of our spectral clustering algorithm on the three components separately: 1) computation of the similarity matrix; 2) sparse matrix eigensolver; and 3) the k-means clustering algorithm. For the CUDA implementation, we measure the time costs that include both the computational time as well as the extra time for library initialization time and data communication. Specifically, we evaluate the performance of each of the following components:

- **Computation of the similarity matrix**:
  - initialize CUDA libraries.
  - transfer data and edge list from CPU to GPU.
  - construct the similarity matrix.

[6] http://www.mathworks.com/discovery/matlab-multicore.html

Table 5.3: Running Time of Spectral Clustering on DTI Dataset

Time/s	CUDA	Matlab	Python
Compute Similarity Matrix	**0.0331**	221.249	220.880
Sparse Eigensolver	**475.442**	603.165	3281.973
K-means Clustering	**5.407**	1785.17	2154.7818

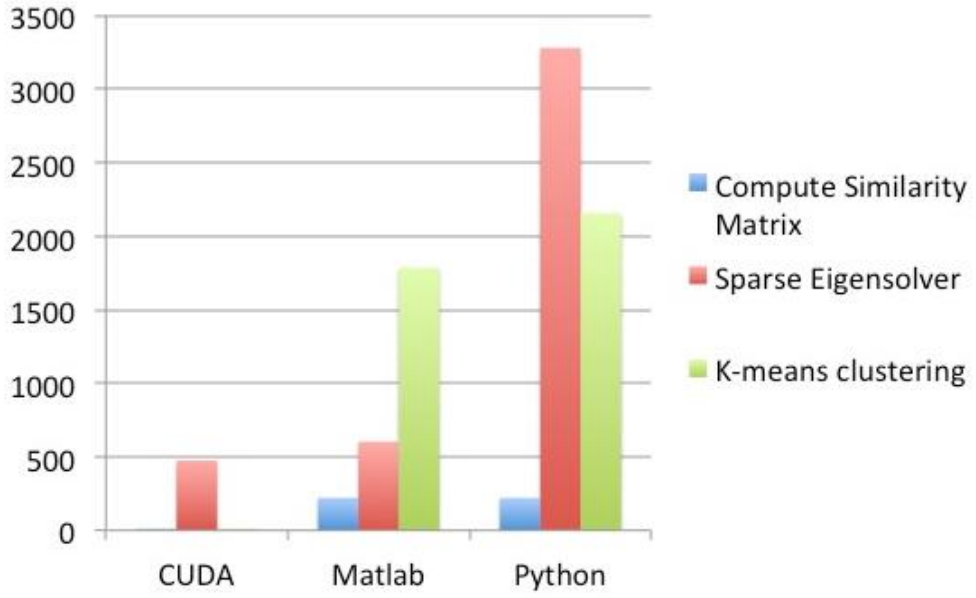


Figure 5.3: Time Costs of Spectral Clustering on DTI Dataset

- **Sparse matrix eigensolver**:
  - data communication between CPU and GPU;
  - computation of the eigenvectors;
  - transfer of the eigenvectors from CPU to GPU.
- **K-means clustering**:
  - perform the k-means clustering;
  - tranfer the clustering result from GPU to CPU.

Figure 5.3. and Table 5.3. show the time costs of each step corresponding to the **DTI** dataset.

It is clear that our CUDA implementation significantly outperforms the currently fastest known Matlab and Python implementations at each step. Since the computation of the similarity matrix is highly parallel, the CUDA implementation achieves linear speedups by taking advantage of the GPU with thousands of threads computing the cross correlation coefficients concurrently. For the Matlab and Python implementations, the results are based on the serial implementation which loops over the edge list and computes the correlation coefficient explicitly using the built-in function. We also tested an alternative implementation which takes advantage of *vectorization* techniques that recast the loop-based operation into matrix and vector operations. The optimized Matlab and Python implementationd take $5.753s$ and $6.271s$ trespectively to compute the similarity matrix.

Both Matlab and Python packages utilize the *reverse communication interfaces* of ARPACK to compute the eigenvectors of large-scale symmetric matrix, and hence all of the three implementations share similar procedures and interfaces. The basic difference is related to the function to compute the sparse matrix-vector multiplication. Our CUDA implementation utilizes the GPU and the cuSPARSE library to compute the multiplication while Matlab and Python utilize their built-in routines. Since the GPU performs significantly better than the CPU on BLAS operations [124], the CUDA implementation achieves better performance than Matlab and Python even with the communication overhead. However, since the time complexity of implicitly restarted Lanczos method is approximately $O(m^3 + nm^2)$, the time spent on the *reverse communication interfaces* scales relatively poorly, which may become the most computationally expensive part when k is large.

As for the *kmeans clustering* algorithm, our CUDA implementation achieves more than 300x speedup over the Matlab and Python implementations. The running time of this step

Table 5.4: Running Time of Spectral Clustering on FB Dataset

Time/s	CUDA	Matlab	Python
Sparse Eigensolver	**0.0216**	0.1027	0.0851
K-means Clustering	**0.007251**	0.0205	0.0259

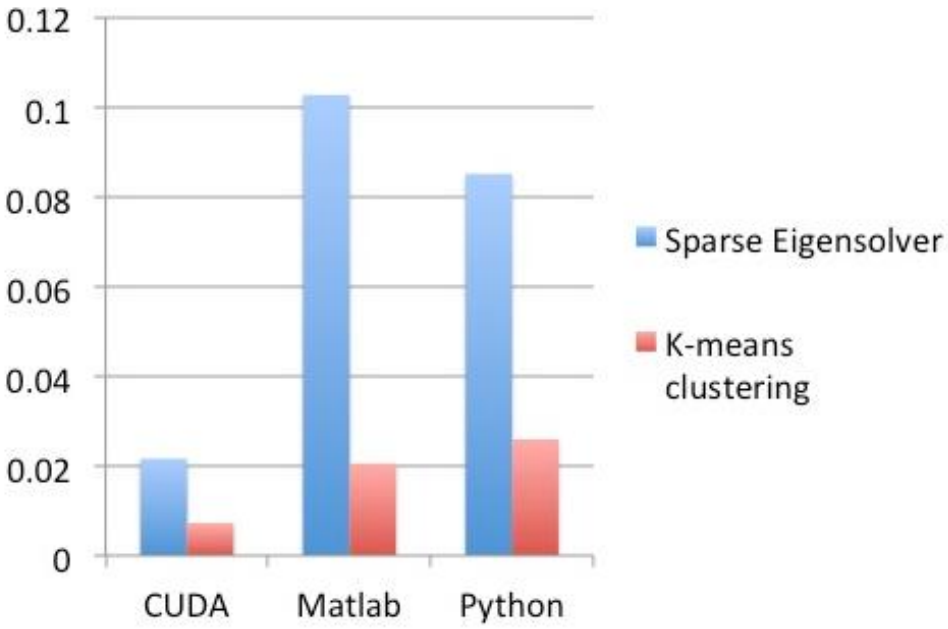


Figure 5.4: Time Costs of Spectral Clustering on FB Dataset

depends on the centroid initialization. The CUDA and Python implementations utilize the k-means++ initialization, which leads to fewer number of iterations in general than Matlab. Moreover, in the CUDA implementation, the process of transforming the computation of the pair-wise distance matrix to the BLAS operations significantly accelerates the running time of the algorithm.

The performance results for the graph datasets (**FB**, **Syn200**, **dblp**) are shown in Table 5.4 through Table 5.6 and Figure 5.4 through Figure 5.6. Similar to the previous results, our CUDA implementation achieves the best performance among the three implementations at each step. However, the speedup ratio depends on the specific problem size.

The FB dataset contains a very small graph with 4039 nodes and involves very few clusters

Table 5.5: Running Time of Spectral Clustering on Syn200 Dataset

Time/s	CUDA	Matlab	Python
Sparse Eigensolver	**4.1153**	6.9531	18.915
K-means Clustering	**0.02478**	38.3728	2.4719

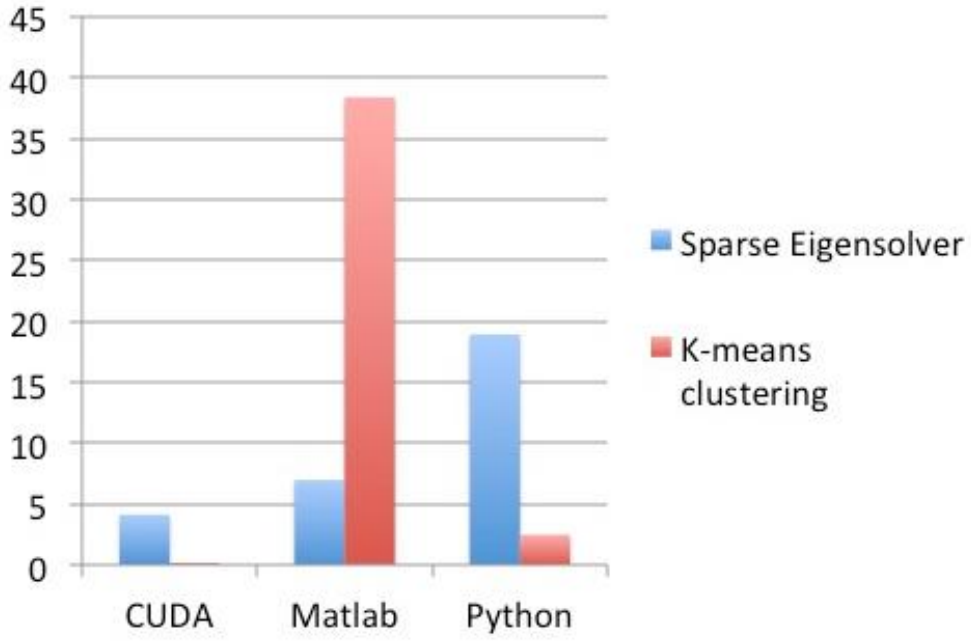


Figure 5.5: Time Costs of Spectral Clustering on Syn200 Dataset

$k = 10$. Because the number of clusters is small, the most expensive computation of *sparse eigensolver* is the sparse matrix-vector multiplication. Therefore for this step,the CUDA implementation achieves around 5x speedup over the other implementations. For the *k-means clustering* step, the CUDA implementation shows only a minor speedup by a factor of around 4x.

The Syn200 dataset contains a medium-sized synthetic graph with 200 clusters. The CUDA implementation achieves a slight improvement in computing the eigenvectors since the performance is mainly constrained by the CPU-based routines. For the of k-means clustering step, the CUDA implementation achieves over 100x speedup.

The dblp dataset contains a large-scale graph with 500 clusters. Both Matlab and Python

Table 5.6: Running Time of Spectral Clustering on dblp Dataset

Time/s	CUDA	Matlab	Python
Sparse Eigensolver	**682.643**	1885.2303	9338.31
K-means Clustering	**1.79456**	1012.92	719.686

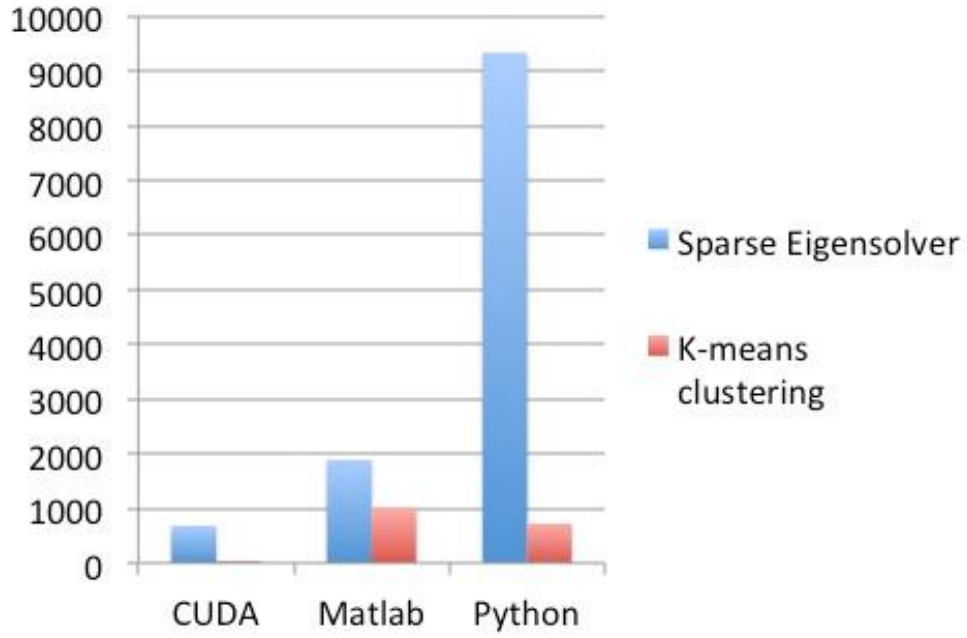


Figure 5.6: Time Costs of Spectral Clustering on dblp Dataset

implementations perform poorly for such a problem size. Our CUDA implementation achieve around 3x speedup in *sparse eigensolver* in spite of the fact that the performance is still constrained by the CPU-based interfaces. In the k-means clustering step, the CUDA implementation achieves over 400x speedup.

Table 5.7 shows a comparison between data communication time and computation time for the CUDA implementation on each of our four datasets. The data communication time includes 1) input data transfered from CPU to GPU; 2) data communication between CPU and GPU during the execution of the eigensolver stage; 3) output results that are transferred from GPU to CPU. Given that the bandwidth remains constant during the execution of the algorithm, the

Table 5.7: Comparison Between Data Communication Time and Computation Time

Time/s	Communication	Computation
DTI	2.248	475.213
FB	0.002131	0.02635
DBLP	2.731	680.31
Syn200	0.0741	3.8201

time complexity of data communication is $O(n^2 + m \times \# + nk)$ depending on the sparsity ratio of the similarity matrix and the number of Arnoldi iterations $\#$ n; the time complexity of computation is $O(nd^2 + O(nm^2) \times \# + O(n^2k))$. Therefore we expect the data communication time to be less than the computational time as in fact illustrated in the Table 5.7, especially for large-scale problems.

In conclusion, our CUDA implementation always achieves better performance than Matlab and Python implementations for each step. The speedup ratio largely depends on the specific problem size. Our traget applications involve problems with a large number of clusters. Our implementation achieves significant speedups for the steps of *computing the similarity matrix* and the *k-means clustering* due to the massive computational power of GPU. Moreover, we always achieve some speedups for the *sparse eigensolver* step by accelerating the computations involving matrix-vector multiplications.

5.6 Conclusion

We presented a high performance implementation of the spectral clustering algorithm on CPU-GPU platforms. Our implementation leverages the GPU to accelerate highly parallel computations and Basic Linear Algebra Subprograms (BLAS) operations. We focused on the acceleration

of the three major steps of the spectral clustering algorithm: 1) construction of the similarity matrix; 2) computation of eigenvectors for large-scale similarity matrices; 3) k-means clustering algorithm. We believe that we are the first to accelerate the large-scale eigenvector computation by combining the interfaces of traditional CPU-based software packages ARPACK and GPU-based CUDA library. Such a combination achieves good speedups compared to other CPU-based software. We deploy a smart seeding strategy and utilize BLAS operations to implement the fast k-means clustering algorithm. Our implementation is shown to achieve significant speedup compared to Matlab and Python software packages, especially for large-scale problems.

Chapter 6: Connectivity-based Brain Parcellations

6.1 Introduction

Diffusion Magnetic Resonance Imaging (MRI) technology non-invasively reveals white matter fiber structures and provide a model of the brain fiber tracts at a relatively high resolution. This opens up new research opportunities to generate, explore and analyze complex brain networks derived from Diffusion Tensor Imaging (DTI) based structural connectivity information [51, 152, 153]. Researchers have successfully applied graph theoretical analysis on specialized structural networks to shed light on differences between different population groups and on brain disorders such as dementia [154] and schizophrenia [155]. Brain network analysis requires a reasonably accurate anatomical segmentation of the cerebral cortex, called parcellation, in which structurally homogeneous regions constitute the nodes of the network. Traditional anatomical brain regions may not incorporate connectivity information, and are typically identified by the distribution of cell types [156], myelinated fibers [157], or neurotransmitter receptors [158]. Common widely-used anatomical brain parcellations include Brodman's areas [47], Automated Anatomical Labeling (AAL) [8], and Jülich histological parcellations [48]. However, there are no generally accepted anatomical parcellations and atlases of the whole brain that are based purely on the anatomic brain connectivity information revealed by diffusion MRI data. Most of existing DTI-based parcellation studies focus on particular parts of the cerebral cortex, such as the human inferior

parietal cortex complex (IPCC) [159], the lateral parietal cortex [160], the temporoparietal junction area (TPJ) [161], the dorsal frontal cortex [162], the ventral frontal cortex [163], cingulate and orbitofrontal cortex [164], and Broca's areas [165]. The parcellations generated specifically for these regions have a small number of subregions but achieve high consistency among subjects of a population sample. Connectivity-based parcellation of the whole brain is challenging due to a number of factors that include: (i) the very large size of the connectivity matrix produced by tractography of each subject's DTI data; (ii) spatial constraints among the voxels of each region that must be respected in addition to the connectivity information; (iii) enforcing consistency for any structurally homogeneous population sample; and (iv) the lack of effective techniques to evaluate, and validate good parcellations.

In this chapter, we propose a novel iterative method based on spectral clustering applied to a sparse representation of the connectivity information which also incorporates the necessary spatial constraints. Our goal is to generate reproducible whole-brain parcellations based purely on DTI data, which are stable and subject-reproducible, achieve highly structurally homogeneous regions, and are consistent among structurally similar population samples. Such parcellations can be used as the basis for conducting graph-theoretic analysis on the resulting anatomic connectivity networks. Our method uses probabilistic tractography to generate the connectivity matrix that represents connectivity strength between any two gray voxels. A sparse representation of the connectivity matrix is defined by a graph whose edges capture spatial connectivity within a small spatial neighborhood and whose edge weights provide a similarity measure of the connectivity profiles of the endpoints. We show that our method is effective in generating parcellations that are highly consistent among subjects in the same population sample and that capture anatomic connectivity patterns that can be used to distinguish between population samples with known

structural differences. Moreover, the methods are computationally efficient and robust to various random factors.

We note two particular works that are directly related to this chapter. The first, reported by Craddock et al. [122], focuses on a data-driven approach for generating atlases based on *resting-state functional* MRI. The main goal there is to parcellate the whole brain into coherent regions of interests that are homogeneous in their resting-state functional connectivity (FC). They develop independently a graph formulation that is similar to ours, and apply spectral clustering in a straightforward way. The resulting atlas, while better than several of the standard anatomical atlases in term of FC homogeneity, has similar characteristics to a random atlas. Moreover, the input size is significantly smaller than the size of the problem we are dealing with here. The second work reported in [166] addresses the same problem tackled in this chapter and uses hierarchical clustering to generate a hierarchy of whole brain parcellations. Hierarchical clustering techniques have serious limitations since they use a local greedy strategy, and each successive refinement cannot modify the clustering determined in previous steps. In addition, the evaluation methodology carried out there is limited to either known results for small regions such as the inferior parietal cortex convexity or to other well-known cytoarchitectonic parcellations that do not incorporate the connectivity information provided by DTI.

We summarize our main contributions in this chapter as follows:

- We develop efficient, scalable algorithms based on a sparse representation of the whole brain connectivity matrix, which reduces the number of edges from around a half billion to a few million while incorporating the necessary spatial constraints.
- For an arbitrary subject from a population sample and for any value k of the number of regions, we show that our algorithm converges to a stable parcellation after a few iterations,

Table 6.1: Subject Demographics

Subject Group	**Male**	**Female**	**Age 18-30**	**Age 31-50**	**Age 51-60**	**Total**
Normal Controls	41	35	23	28	25	76
Schizophrenia	31	17	16	17	15	48

defined by k structurally homogeneous regions.

- Our parcellations of subjects within a population sample are consistent using any of a number of similarity metrics between parcellations of different subjects.
- Our method captures structural patterns to allow us to distinguish effectively between structurally different population groups such as Males vs Females, Normal Controls vs Schizophrenia, and different age groups in Normal Controls.

The rest of the chapter is organized as follows. We start in the next section by describing the data and tools used to generate the connectivity matrix of each subject. Our iterative method is described in Section III, while the stability and reproducibility results at the individual subject level or group level are covered in Section IV. Section V covers the discriminative power of the resulting parcellations. We end with a brief discussion in Section VI.

6.2 Data and Preprocessing Steps

6.2.1 Data Acquisition

Imaging was performed at the University of Maryland Center for Brain Imaging Research using a Siemens 3T TRIO MRI (Erlangen, Germany) system and 32 channel phase array head coil. The high-angular resolution diffusion imaging protocol was used to assess white matter integrity as measured by fractional anisotropy. Diffusion tensor data were collected using a

single- shot, echo-planar, single refocusing spin-echo, T2-weighted sequence with a spatial resolution of 1.7×1.7×3.0mm. The sequence parameters were: TE/TR=87/8000ms, FOV=200mm, axial slice orientation with 50 slices and no gaps, 64 isotropically distributed diffusion weighted directions, two diffusion weighting values (b=0 and 700s/mm^2) and six b=0 images. These parameters were calculated using an optimization technique that maximizes the contrast to noise ratio for FA measurements. The total scan time was approximately 9 minutes per subject. For each subject, the image data consists of 70 volumes of 3D images of dimensions 128×128×53, each voxel representing 1.718mm×1.718mm×3mm brain volume. We collected data from 76 normal (NC) subjects and 48 schizophrenia (SZ) subjects. The subject demographics are shown in Table 6.1.

6.2.2 Nonlinear Registration

The diffusion images of all subjects are registered to Montreal Neurological Institute (MNI) standard space using nonlinear registration package FNIRT in FSL [44]. The nonlinear registration process generates the warping coefficients that balance the similarity between the diffusion image and the standard MNI152 image, and the smoothness of the warping coefficients. The registration process facilitates group atlas generation and comparison with other standard atlases.

6.2.3 Probabilistic tractography

The preprocessing step of probabilistic tractography is used to model cross fiber distributions for each voxel through the BEDPOSTX package in FSL [45]. Probabilistic tractography is processed through the diffusion toolbox in FSL [46]. The standard white matter atlas is specified as a seed region. The AAL mask is specified as the target region, which is the whole brain

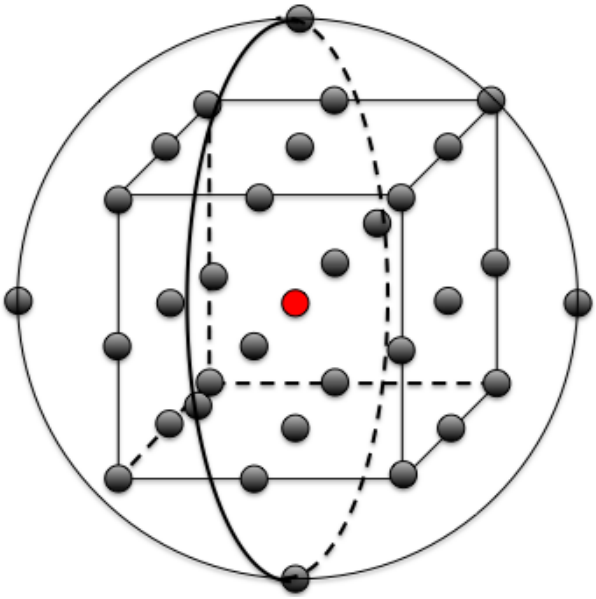


Figure 6.1: 32 neighbors of voxel within sphere of radius $r = 2$. Note that each voxel represents 1.718mm×1.718mm×3mm brain volume. The figure shows the symbolic neighbors of voxels rather than the actual volume size.

cortex region. We generate 50 streamlines from every voxel in a seed region. These streamlines are propagated following the cross fiber distribution computed from the preprocessing step. Curvature threshold is enforced to eliminate unqualified streamlines. The distance correction option is set to correct for the fact that the distribution drops as travel distance increases. The tractography output is a structural connectivity network modeled as a weighted graph where each node is a voxel in the target region space and each edge weight corresponds to relative connectivity strength in terms of the number of streamlines connecting the corresponding pair of voxels.

6.3 Our Approach

Our main method takes as input a subject's connectivity matrix. The number of voxels in the AAL mask is $155,794$ and the connectivity matrix is a sparse matrix of size $155,794 \times$

$155,794$. Given a positive integer value k, our problem is to parcellate the cerebral cortex into k spatially contiguous regions, such that each region possesses a high degree of structural homogeneity. Moreover, these parcellations must be stable and reproducible, as well as, consistent among members of a population sample with similar connectivity patterns. We first introduce our notion of a *connectivity profile* followed by a description of our method.

6.3.1 Connectivity Profile

For each voxel, the connectivity profile is the signature that discriminates a voxel from the rest of voxels based on connectivity. Parcellations are built by clustering voxels with similar connectivity profiles together. In principle, we can take the row of the connectivity matrix corresponding to a voxel as its connectivity profile, but that would be computationally expensive to process even if we compress each row into a list that contains only connectivity values above a certain threshold. In our approach, the connectivity profile of a voxel is computed as an array of weights, where each element represents the cumulative connectivity strengths of the voxel to a set of the regions determined initially by a predefined brain segmentation. Not only does the use of a coarser version of the connectivity profile leads to much more efficient computations, but it also helps to smooth out errors introduced by the tractography process through aggregation. More importantly, we will show later that our method converges to the same parcellation regardless of the initial segmentation used.

We explore several possibilities to initialize brain segmentations. An obvious choice is to use the regions of interests (ROIs) defined by any of the well-known anatomical atlases such as the 90 regions of the AAL-90 atlas. Note that the initial number of spatial regions is *completely*

unrelated to the number k of parcellated regions and is used merely to initialize the connectivity profile of each voxel. Another possibility is to use a brain segmentation generated using a spatially constrained version of the k-means++ algorithm with randomized centers [149]. The third possibility that we consider is to spatially segment the volume into almost equal-size sub-cubes. The last two segmentation methods result in any specified number of contiguous regions. We will show that our method results in consistent and similar parcellations regardless of which connectivity profile we use.

6.3.2 Spatially Constrained Similarity Graph

A spatial-constraint similarity graph, considerably sparser than the weighted graph defined by the connectivity matrix, is formed using spatial adjacency and the connectivity profiles as follows. The voxels define the nodes of our graph. Two nodes are connected by an edge if and only if the corresponding voxels lie within a sphere of radius r. In our implementation, we have used $r = 2$ such that the number of neighbors of any node is at most 32 as shown in Fig. 6.1. Each edge is weighted by the similarity between the connectivity profiles of its end points. We can use any of several similarity metrics, including the correlation coefficient or the cosine function; our tests show that the results are very similar regardless of the similarity measure used. We assume from now on that we are using the correlation coefficient as our similarity measure between the connectivity profiles of two voxels.

Algorithm 8 Iterative Parcellation Method

1: Generate the connectivity matrix of a subject using probabilistic tractography.
2: Construct a spatial graph as a sparse representation of the 3-D brain.
3: Initialize a random spatially-coherent brain parcellation, to be used to define the connectivity profile of each voxel.
4: **while** the similarity measurement not exceeds some threshold. **do**
5: Use the current brain parcellation to define the connectivity profiles of all the voxels based on the connectivity matrix.
6: Apply spectral clustering algorithm to generate the brain parcellation of a predefined level of granularity.
7: Measure the similarity between the new parcellation and the previous parcellation used to define connectivity profiles.
8: **end while**
9: Return the parcellation result.

6.3.3 Minimum Graph-Cut Problem and Iterative Refinement

Our parcellation algorithm starts by partitioning our spatial similarity graph into several subgraphs with the objective of minimizing the total weight of the edges connecting the subgraphs subject to a constraint on the relative sizes of the subgraphs. More specifically, our objective function is to minimize the normalized cut rather than just the cut, which is standard in the literature (see for example [19, 137, 167]). This will more or less ensure that we won't have subgraphs with very few vertices. The subgraphs induce a spatial segmentation of the 3D image data, which is then used to redefine the connectivity profile of each voxel, after which we iterate until the generated parcellations are almost unchanged. Our algorithm results in a solution where the voxels within the same region have similar connectivity profiles and voxels across different regions are relatively dissimilar. The most efficient method to solve the graph cut problem during each iteration is spectral clustering [19, 137, 167]. In particular, we use the normalized spectral clustering method, which can be summarized as follows, where $W \in \mathbb{R}^{n \times n}$ is the weight matrix associated with the spatial similarity graph and k is the number of desired regions.

- Compute the normalized Laplacian matrix $L = D - W$. D is the diagonal matrix with each element $D_{i,i} = \sum_{j=1}^{n} W_{i,j}$.
- Compute the k eigenvectors of $D^{-1/2}LD^{-1/2}$ corresponding to the smallest k eigenvalues.
- Apply the k-means clustering algorithm on the rows of the eigenvectors to obtain the final clusters.

To make the clustering result consistent against the random initializations in the k-means step, we run the k-means++ algorithm [149] several times and choose the result with the minimum within-cluster sum of point-to-centroid distances [168]. Note that each run of the k-means++ involves $155,794$ points (voxels) each of dimension k. Algorithm 8 provides a high-level description of our method.

By applying the spectral clustering algorithm, we expect voxels within the same region to possess successively higher degrees of similarity in terms of structural connectivity during successive iterations. This iterative refinement approach converges to a stable parcellation as we will later show. At that point, we will also introduce a quantitative stopping criterion to be used to terminate the algorithm.

6.4 Reproducibility and Stability Analysis

This section presents the methodology used and the results achieved to illustrate the reproducibility of our results both at the individual subject level and at the group level. We start by introducing two well-known methods to quantitatively measure the similarity between two arbitrary clustering solutions of a dataset.

6.4.1 Parcellation Similarity Metrics

We use the following metrics to measure the similarity between any two parcellations with the same level of granularity (that is, the same value of k). The cluster labels generated by our method are essentially arbitrary in the sense that regions with the same labels in two different parcellations are not necessarily spatially related. Moreover, as the level of granularity increases, we may not be able to determine a reasonable one-to-one mapping between the regions. In this chapter, we will use the following two metrics.

6.4.1.1 Normalized Mutual Information (NMI)

Mutual information has been used in information theory to measure the relationship between any two probability distributions [169]. Essentially, it provides a measure of how similar the joint distribution of two random variables is to the product of their marginal distributions. The normalized mutual information (NMI) is an approximate discrete version commonly used to measure the similarity between pairs of clusters of a dataset. It has a value between 0 and 1, with the value 0 indicating the two clusterings are completely independent of each other, whereas the value 1 indicates that they are identical. The NMI between two parcellations A and B is defined as

$$NMI\left(A,B\right)=\frac{MI\left(A,B\right)}{\left(H\left(A\right)+H\left(B\right)\right)/2} \tag{6.1}$$

The entropy for individual parcellations and the mutual information are approximated from the marginal and joint distributions as follows. A_i is the set of voxels that are labeled as i in

parcellation A. Similarly, B_j is the set of voxels that are labeled as j in parcellation B.

$$H(A) = -\sum_{i=1}^{k} p(A_i) \log p(A_i) \tag{6.2}$$

$$H(B) = -\sum_{j=1}^{k} p(B_j) \log p(B_j) \tag{6.3}$$

$$MI(A, B) = \sum_{i=1}^{k} \sum_{j=1}^{k} p(A_i, B_j) \log \left(\frac{p(A_i, B_j)}{p(A_i)\, p(B_j)} \right) \tag{6.4}$$

The marginal probability for any label is approximated as the fraction of the number of voxels with that label over the total number of voxels. Similarly, the joint distribution $p(A_i, B_j)$ is computed as the fraction of the number of voxels with label i in parcellation A and with label j in parcellation B over the total number of voxels. Here the total number of voxels is the number of voxels in the AAL mask, which is the same for all parcellations.

$$p(A_i) = \frac{size(A_i)}{\sum_{i=1}^{k} size(A_i)}, p(B_j) = \frac{size(B_j)}{\sum_{j=1}^{k} size(B_i)} \tag{6.5}$$

$$p(A_i, B_j) = \frac{size(A_i \cap B_j)}{\sum_{i=1}^{k} size(A_i)} \tag{6.6}$$

As stated previously, if the parcellations are identical, except for label reordering, then the mutual information and the entropy for each parcellation are equal, and hence the resulting NMI is equal to 1. The higher the value of the NMI, the more similar the two parcellations are.

6.4.1.2 Dice's Coefficient

Dice's coefficient measures the similarity directly from the clustering matrix $C \in \mathbb{R}^{n \times n}$ defined by

$$C_{i,j} = \begin{cases} 1, & L_i = L_j \\ 0, & L_i \neq L_j \end{cases} \tag{6.7}$$

where L is the vector that contains the label of every voxel. That is, the (i, j) entry of the clustering matrix is equal to 1 if, and only if, voxels i and j belong to the same region. Given the matrices corresponding to two parcellations, the Dice's coefficient is computed as twice the number of common nonzero entries normalized by the total number of nonzero entries in both clustering matrices [170]. Dice's coefficient is always between 0 and 1, and the larger it is, the more similar the two parcellations are.

Both NMI and Dice's coefficient capture the similarity between two parcellations of any level of granularity. But for NMI, the joint distribution $p(A_i, B_j)$ is in general greater than the product of the marginal distributions $p(A_i)\, p(B_j)$, which may cause NMI to overestimate the similarity between the parcellations. We note that in general NMI is larger than the Dice's coefficient.

6.4.2 Stability and Reproducibility of Subject Parecellations

The main factors that affect the parcellations generated by our algorithm are the choice of the brain segmentation that is used to define connectivity profiles and the random initialization

of the k-means++ algorithm used in the last step of spectral clustering. The effect of random initialization could be mitigated by running the k-means++ initialization [149] several times, as stated before. Here we consider only the effect of the initial brain segmentation used to define the connectivity profiles. Note that the connectivity profiles encapsulate the only information we have from the DTI data for each subject since the rest of the information captured by the spatial similarity graph does not involve anything related to the connectivity data.

The initial brain segmentation can be defined as any arbitrary spatial segmentation of the brain mask. However it would be more intuitive to use initial segmentations with comparable region sizes. Note that the number of regions in the initial segmentation is completely independent of the desired number k of parcellated regions. The main result of this section is that, regardless of the initial segmentation and for any value of k, our algorithm will converge to a stable parcellation for each subject that captures the critical connectivity information embodied in the DTI data.

The following brain segmentations, shown in Fig. 6.2, are used to define initial connectivity profiles that are used to generate 40-region parcellations.

6.4.2.1 Automated Anatomical Labeling (AAL)

The AAL atlas defines 90 anatomical regions with 45 volumes of interest in each hemisphere, which were delineated following the courses of the main sulci of the brain. In fact, we have used the AAL mask to define cerebral cortex to be parcellated. Here we use it as the initial segmentation that defines the connectivity profiles of all voxels.

(a) AAL

(b) Random segmentation with 90 regions and 1000 regions

(c) Regular grid segmentation with cube size $5 \times 5 \times 5$

(d) Synthetic 40-region parcellation

Figure 6.2: Brain segmentations used to define connectivity profiles.

6.4.2.2 Random Spatial Segmentation

We generate a random spatial segmentation with any level of granularity using the k-means++ algorithm, based only on spatial coordinate. The purpose is to generate segmentations that have regions that are spatially contiguous and compact. Random initialization of the k-means++ using 90, 1000, and 2000 regions were generated.

6.4.2.3 Regular Grid Segmentation

A regular grid segmentation consists of a set of almost equal-sized cubes that cover the whole brain. The cube size determines the granularity of the segmentation. We set the cube size to 5 and therefore, this segmentation consists of 1,987 cubes that cover all brain voxels.

6.4.2.4 Synthetic Parcellations

The synthetic parcellations are generated from the similarity graph in which the weights of all the edges are set to 1. A similar approach was reported in [122], which concludes that the synthetic parcellations are almost as good as real parcellations in terms of FC cluster homogeneity. We use the synthetic parcellation with the same number of regions to define the connectivity profile and show that starting from the same synthetic parcellation, our iterative method will incorporate the underlying connectivity information and converge to the subject's characteristic parcellations.

For each subject, we show that our algorithm will yield essentially the same parcellation for all these initial segementations. The NMI and Dice's coefficient are computed between all pairs of parcellations generated from different brain segmentations after each iteration. Tables

Abbreviations

AAL: Automatic Anatomical Labeling.
R#: Random brain segmentation with # number of regions.
Grid: Regular grid segmentaton with grid size $5 \times 5 \times 5$.
S#: Synthetic parcellation generated from spatial-constrained similarity graph with all edges' weights as 1.

Table 6.2: NMI Between Parcellations After 1^{st} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.8673	0.8791	0.8497	0.8734	0.8568
R90	-	1.0000	0.9009	0.8622	0.8849	0.8804
R1000	-	-	1.0000	0.8774	0.9039	0.8657
R2000	-	-	-	1.0000	0.8855	0.8433
Grid	-	-	-	-	1.0000	0.8666
S40	-	-	-	-	-	1.0000

Table 6.3: NMI Between Parcellations After 2^{nd} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.9173	0.9085	0.9117	0.8999	0.8990
R90	-	1.0000	0.9042	0.9031	0.8880	0.8954
R1000	-	-	1.0000	0.9018	0.8971	0.8908
R2000	-	-	-	1.0000	0.8840	0.8780
Grid	-	-	-	-	1.0000	0.8980
S40	-	-	-	-	-	1.0000

6.2 through 6.9 show the corresponding results.

The above tables show that similarity, in terms of NMI or Dice's coefficients, between all pairs of parcellations from different brain segmentations increase with the number of iteration. After the 4^{th} iteration, most of NMI values are

above 0.90 and most of Dice's coefficients are above 0.80, which indicates very consistent parcellations. The iterative method mitigates the random effect caused by the initial arbitrary

Table 6.4: NMI Between Parcellations After 3^{rd} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.9194	0.8940	0.9412	0.8985	0.9035
R90	-	1.0000	0.9050	0.9266	0.9242	0.9266
R1000	-	-	1.0000	0.9103	0.8899	0.8992
R2000	-	-	-	1.0000	0.9087	0.9064
Grid	-	-	-	-	1.0000	0.9101
S40	-	-	-	-	-	1.0000

Table 6.5: NMI Between Parcellations After 4^{th} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.9405	0.9052	0.9382	0.9004	0.9258
R90	-	1.0000	0.9181	0.9211	0.9141	0.9494
R1000	-	-	1.0000	0.8949	0.8828	0.9148
R2000	-	-	-	1.0000	0.9225	0.9075
Grid	-	-	-	-	1.0000	0.9020
S40	-	-	-	-	-	1.0000

Table 6.6: Dice's Coefficient Between Parcellations After 1^{st} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.7448	0.7709	0.7083	0.7666	0.7230
R90	-	1.0000	0.8374	0.7456	0.8046	0.7810
R1000	-	-	1.0000	0.7778	0.8396	0.7553
R2000	-	-	-	1.0000	0.7874	0.7010
Grid	-	-	-	-	1.0000	0.7585
S40	-	-	-	-	-	1.0000

segmentations and leads to stable parcellations regardless of the initial definition of connectivity profiles. Note that the k-means++ step of our algorithm introduces a small uncertainty, which explains the few deviations in the tables above. However, it is clear that the parcellations generated

Table 6.7: Dice's Coefficient Between Parcellations After 2^{nd} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.8557	0.8378	0.8483	0.8142	0.8236
R90	-	1.0000	0.8203	0.8199	0.7852	0.8068
R1000	-	-	1.0000	0.8205	0.8097	0.7936
R2000	-	-	-	1.0000	0.7737	0.7717
Grid	-	-	-	-	1.0000	0.8099
S40	-	-	-	-	-	1.0000

Table 6.8: Dice's Coefficient Between Parcellations After 3^{rd} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.8543	0.8007	0.9021	0.8047	0.8197
R90	-	1.0000	0.8316	0.8759	0.8700	0.8787
R1000	-	-	1.0000	0.8352	0.7932	0.8103
R2000	-	-	-	1.0000	0.8317	0.8341
Grid	-	-	-	-	1.0000	0.8378
S40	-	-	-	-	-	1.0000

Table 6.9: Dice's Coefficient Between Parcellations After 4^{th} Iteration

Segmentation	AAL	R90	R1000	R2000	Grid	S40
AAL	1.0000	0.9070	0.8279	0.8936	0.8125	0.8786
R90	-	1.0000	0.8526	0.8547	0.8387	0.9247
R1000	-	-	1.0000	0.7960	0.7653	0.8472
R2000	-	-	-	1.0000	0.8699	0.8290
Grid	-	-	-	-	1.0000	0.8171
S40	-	-	-	-	-	1.0000

at the end of third and fourth iterations are very close to each other. Fig. 6.3 illustrates the increase of the average, over all the different intial segmentations, of NMI and Dice's coefficient after each iteration.

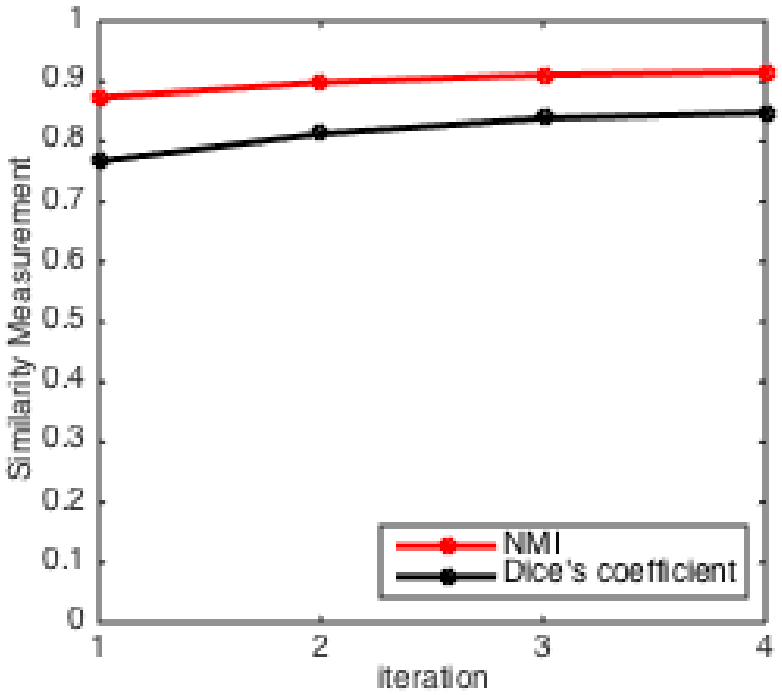


Figure 6.3: Subject reproducibility after each iteration.

Table 6.10: NMI Between Parcellations in Consecutive Iteration Stages

Segmentation	1^{st} / 2^{nd}	2^{nd} / 3^{rd}	3^{rd} / 4^{th}
AAL	0.9131	0.9353	0.9539
R90	0.9125	0.9325	0.9631
R1000	0.9199	0.9292	0.9198
R2000	0.8873	0.9224	0.9314
Grid	0.9185	0.9380	0.9267
S40	0.9151	0.9341	0.9486

Table 6.10 and 6.11 show the similarity between parcellations in consecutive iteration stages for a given initial segmentation. Taking into consideration the uncertainty introduced by the k-means++ step of our algorithm, it is clear that successive iterations of the algorithm generate more similar parcellations, for any of the initialization methods of the connectivity profiles.

Table 6.11: Dice's coefficient Between Parcellations in Consecutive Iteration Stages

Segmentation	1^{st} / 2^{nd}	2^{nd} / 3^{rd}	3^{rd} / 4^{th}
AAL	0.8448	0.8886	0.9179
R90	0.8402	0.8838	0.9475
R1000	0.8714	0.8787	0.8495
R2000	0.7997	0.8556	0.8697
Grid	0.8700	0.8960	0.8709
S40	0.8520	0.8813	0.9124

6.4.3 Group Consistency and Atlas Generation

Table 6.12 shows the average similarity between every pair of parcellations from subjects in the NC group. As can be seen from entries in this table, the parcellations are reasonably consistent within the NC group; similar results hold for the SZ group.

Table 6.13 shows the average similarity between a random parcellation and the parcellations generated for the subjects in the NC group. As can be seen from the column of the Dice coefficients, our generated parcellations are significantly different from random parcellations. As mentioned before, the NMI coefficients tend to overestimate the similarity between the parcellations, and hence the slightly higher numbers in the second column of Table 6.13, but still significantly lower than the similarity of the generated parcellations between the subjects of the NC group (Table 6.12).

Atlas generation: We employ the following atlas generation procedure to further validate within-group consistency. In generating our parcellations, regions are labeled randomly; therefore, regions with the same index are not necessarily spatially matched. The first step of atlas generation is to align all parcellations to a reference parcellation that is randomly chosen from the group.

Table 6.12: Average Similarity Between Parcellations of Different Subjects within the NC Group

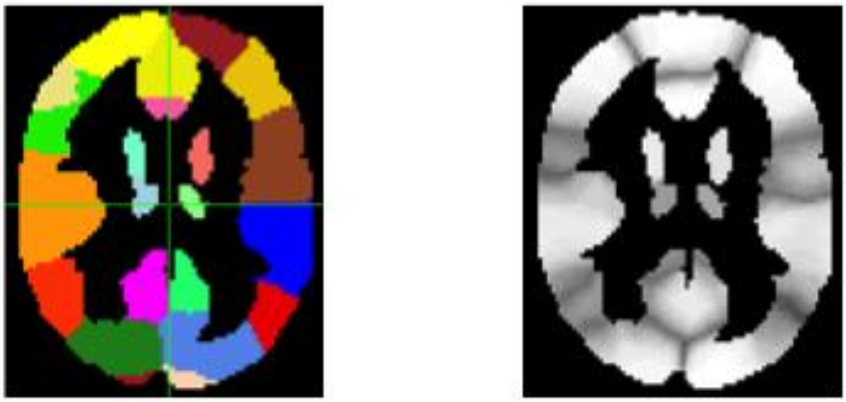
Number of regions	**NMI**	**Dice's Coefficient**
40	0.7734	0.5503
50	0.7786	0.5323
60	0.7939	0.5507
70	0.7988	0.5415
90	0.8040	0.5326
120	0.8151	0.5287

Table 6.13: Average Similarity Between Parcellations of Subjects within the NC Group and Randomly Generated Parcellation

Number of regions	**NMI**	**Dice's Coefficient**
40	0.6923	0.3994
50	0.6857	0.3679
60	0.7140	0.3871
70	0.7164	0.3771
90	0.7393	0.3995
120	0.7452	0.3720

We relabel each of the regions using the region index of the reference parcellation that shares the largest overlapped area. For a group of N relabeled subjects, we generate an atlas as follows. For each voxel, we associate a vector of length N consisting of the label index from each subject. We set the voxel's label to be the most frequent index in its vector, thereby generating an atlas as shown in Fig. 4(a).

The confidence map is a gray-scale image, where the gray level of each voxel represents the uncertainty of the labeling across all subjects, in terms of the proportion of the frequent index in the N-length vector. The confidence map in Fig. 4(b) shows that for almost all voxels, except possibly along the region boundaries, most subjects are consistently labeled as indicated by the



(a) Atlas of healthy group (b) Confidence map

Figure 6.4: Atlas and confidence map for the NC group. Note that for confidence map, the grey scale represents the ratio of overlapped regions.

Table 6.14: P-value and T-statistic for Gender Study within the NC Group

Similarity comparison	**p-value**	**t-statistic**
Male vs Male-female	9.0286e-5	-3.9205
Female vs Male-female	1.4025e-8	5.6911
Male vs Female	3.3752e-20	-9.2766

atlas.

6.5 Discriminative Analysis

In this section, we show how our parcellations can be used to shed light on structural differences between different experimental groups. We have selected cases that were known to have significant differences in white matter integrity and structural networks. We include a discussion of three significant different groups: Male vs Female, Age groups, and SZ vs NC. The subject demographics in our data are shown in Table 6.1.

We adopt two strategies to discriminate among experimental groups. The first strategy focuses on the heterogeneity of the parcellations within a group sample and is based on the pair-

Table 6.15: P-value and T-statistic for Age Study within the NC Group

Similarity comparison	**p-value**	**t-statistic**
Group I vs Group I-II	0.5133	-0.6539
Group II vs Group I-II	0.5566	0.5880
Group I vs Group II	0.2009	-1.2798
Group II vs Group II-III	2.4175e-4	3.6800
Group III vs Group II-III	0.0028	-2.9921
Group II vs Group III	6.6455e-11	6.5814

Table 6.16: P-value and T-statistic for Schizophrenic Study

Similarity comparison	**p-value**	**t-statistic**
NC vs NC-SZ	2.9995e-89	20.2514
SZ vs NC-SZ	1.4025e-8	-10.4867
NC vs SZ	1.1636e-198	30.9687

wise similarities between all pairs of parcellations in a group. As shown in the previous section, our parcellations are consistently labeled across subjects of a population sample except for some boundary voxels. The boundary differences reflected by the pair-wise similarity may be used to determine some features that are specific to particular subgroups. In particular, we will show that the parcellations of the subjects in the SZ group have substantially more variability that those of the NC group and that healthy males seem to exhibit more heterogeneity within their group than healthy females do.

The other strategy is to analyze the structural connectivity network built from the parcellations and tractography results, where the nodes correspond to the parcellation regions and the edge weights correspond to the cumulative connectivity strength between voxels in the two regions; this strategy is commonly used in the literature [171, 172]. Our iterative method generates parcellations where voxels within the same region share similar connectivity profiles that are

defined as the accumulated connectivity strength to every other region. Hence the parcellations obtained are consistent with the structural connectivity network where the connectivity pattern of each node summarizes the connectivity profile of the voxels in that region. The "connectome" analysis shows more powerful discriminative ability of our parcellations than using existing anatomical atlases.

6.5.1 Similarity-based Analysis

The analysis is based solely on pair-wise similarity between pairs of parcellations. We start by analyzing the similarities relative to female and male subgroups of the NC sample using parcellations with 90 regions and NMI as the similarity measure. Results corresponding to other values of k or to the use of Dice's coefficient as a similarity measure exhibit the same patterns.

A two-sample t-test was performed on the pair-wise similarity between parcellations within the female subgroup, the male subgroup, and pair-wise similarity between parcellations from different groups. The p-value and t-statistics are shown in Table 6.14.

Our results indicate that the similarity of parcellations of either healthy females or healthy males is significantly different that the similarity between a female parcellation and a male parcellation. More importantly, the last row of Table 6.14 indicates that the female parcellations are much closer to each other that the male parcellations, which may indicate more structural brain heterogeneity among the male subjects than among the female subjects.

In the age study, we divide the NC sample into three age groups, which are: Group I: Age 18-29, 23 subjects; Group II: Age 30-49, 28 subjects; Group III: Age 50-62, 25 subjects. The p-value and t-statistics are shown in table 6.15. For parcellations in Group I and II, their similarities

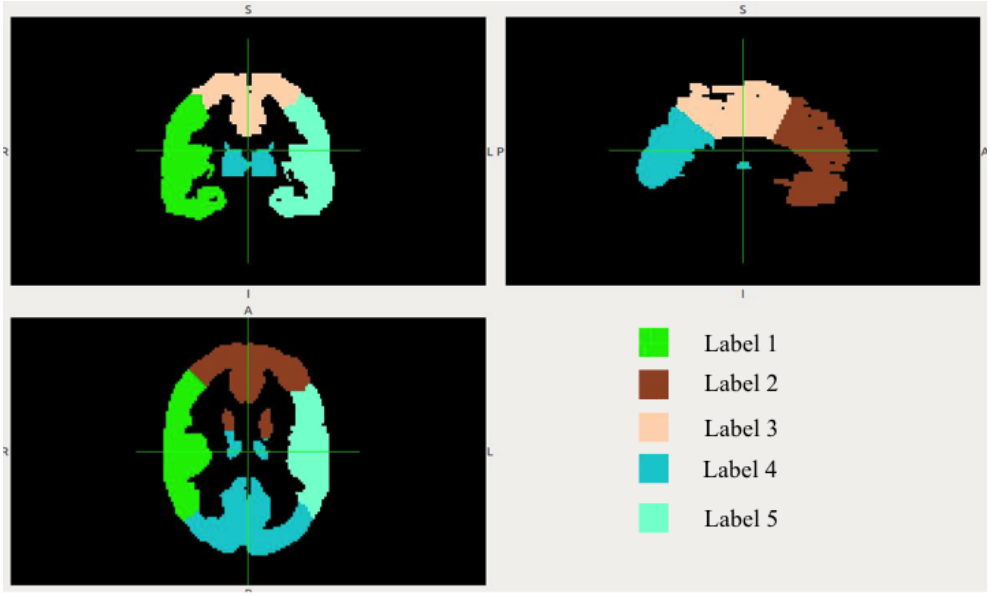


Figure 6.5: Parcellation with 5 regions.

did not show significant differences. But parcellation similarities within Group II have significant differences than the similarity between parcellations in Group II and Group III. And parcellations in Group III are more heterogeneous than those in Group I and Group II.

Perhaps more interesting is the similarity comparison of the parcellations of NC vs SZ groups, illustrated in 6.16. These results clearly show that the parcellations within the SZ group show much more heterogeneity than those for the NC group.

6.5.2 Connectome Analysis

There is much evidence supporting that schizophrenia is a disorder related to brain connectivity. Our previous work analyzed the structural connectivity network based on individual parcellations refined from the AAL atlas to discriminate schizophrenic and normal control groups with high accuracy [173]. We apply the same strategy to discriminate among the two groups using the 5-region parcellations generated from our iterative approach as shown in Fig. 6.5. The reason we

choose a small number of regions is the high consistency across subjects, and because regions can be trivially mapped spatially, one-to-one, between any pair of parcellations.

We first relabel all parcellations based on a randomly selected subject. The connectomes are built by defining nodes as regions in the parcellation and edge weights represent cumulative connectivity strength between regions. Table 6.17 shows the p-value and t-statistcs of pair-wise connectivity between the two groups.

A large portion of pair-wise connectivity shows significant differences between the two groups. Moreover, most pair-wise connectivity strengths of NC subjects are greater than those of SZ subjects, a fact that is consistent with the previous findings that SZ subjects have decreased inter-hemispheric and intra-hemispheric connectivity [174]. We select the three pairs with the most significant p-values and use their connectivity values as features to train a support vector machine classifier. We test our classifier using a 10-fold cross-validation and are able to achieve up to 75% accuracy, which is significantly better than our earlier result in [173].

We also carried out an additional test to confirm the discriminative capabilities of our parcellations. Consider 40-region parcellations for the two population samples and the corresponding structural connectivity networks. For each edge, we perform a two-sample t-test between the sequence of connectivity strengths of the NC group and that of the SZ group. We find that many of the edges result in p-values less than 0.00005 as shown in Fig. 6.6. Fig. 6.7 shows the corresponding results when we use the AAL atlas and determine connectivity strengths on the corresponding edges (pairs of regions). As shown by the binary maps, the proportion of entries in the AAL-based network which have significant connectivity strength difference between healthy controls and schizophrenic subjects is much smaller than that those obtained through the network built from our 40-region parcellations.

Table 6.17: P-value and T-statistic of Pair-wise Connectivity Between Normal Controls and Schizophrenic Groups

Label	1	2	3	4	5
1	0.0137	0.0395	0.0038	0.0989	0.4954
2	0.0300	0.0279	0.0029	0.0514	0.0058
3	0.0029	0.0042	4.11e-4	3.04e-5	0.0019
4	0.0798	0.0579	3.14e-5	0.0775	0.1127
5	0.4928	0.0083	0.0028	0.1216	0.2377

Label	1	2	3	4	5
1	2.5019	2.0809	2.9479	1.6630	-0.6839
2	2.1963	2.2250	3.0391	1.9671	2.8055
3	3.0437	2.9197	3.6325	4.3330	3.1707
4	1.7666	1.9149	4.3244	1.7806	1.5976
5	-0.6879	2.6846	3.0493	1.5588	1.1865

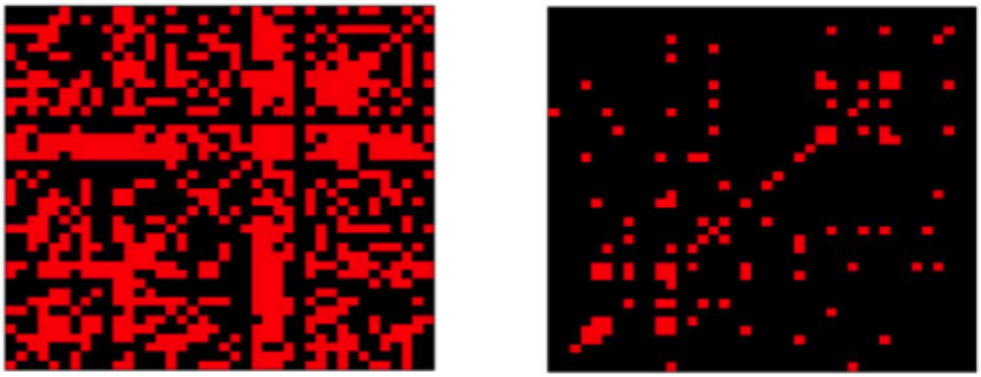


Figure 6.6: Binary maps where entries in red color have p-values <0.05 and <0.00005 respectively in terms of connectivity strengths between the two population groups using our 40-region parcellations.

It seems clear that our pacellations seem to effectively capture the inherent connectivity information present in the DTI data and hence are more suitable for studying structural connectivity than anatomical atlases.

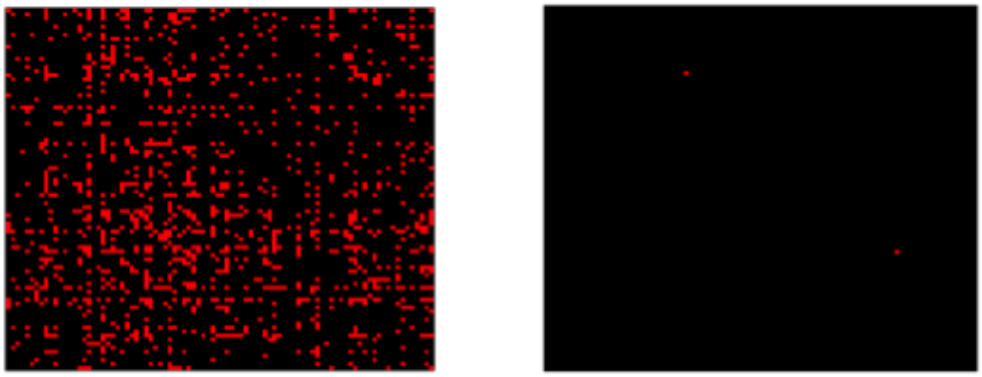


Figure 6.7: Binary map where entries in red color have p-values $<$0.05 and $<$0.00005 respectively between connectivity strengths of the two population groups using the AAL atlas.

6.6 Conclusion

We herein propose a sparse representation of the connectivity information derived from DTI data and a novel method that generates whole-brain parcellations for any number k of regions. Our method is computationally efficient and is able to consistently generate stable and reproducible parcellations that seem to capture inherent structural patterns present in the data. The results are validated through the use of a number of methods, including subject reproducibility, group consistency, and discriminative characteristics between different population groups.

Chapter 7: Concluding Remarks and Future Work

In the thesis, we have developed a number of methods to learn graph representations for large-scale graphs. We first propose a general graph coarsening framework which coarsens large graphs to smaller ones with preserved spectral properties. Based on the insights from the spectral graph theory, we define a new distance function that capture the key graph information loss in the graph coarsening process. We propose efficient algorithms to minimize the distance during the coarsening process. Empirical results show that the coarse graphs can achieve similar performance compared with using the original graphs in graph classification tasks.

Next we propose a new graph neural network model which can generate graph representations capturing key graph characteristics. The new model represents graphs as the weighted composition of sequence representations. Through the design of the weight and sequence functions, the graph representations can achieve strong expressiveness while maintaining the permutation invariance property. Experiments show that our proposed graph representation model achieves superior performance in graph classification tasks.

To address the high computational cost of graph spectral methods, we provide an efficient implementation of the spectral clustering on CPU-GPU heterogeneous platforms. Our implementation significantly accelerates the computation of the spectral clustering subroutines leveraging on the advantages of multi-core CPU and SIMD capabilities of GPUs. We show that the implementation

achieves significantly acceleration compared with CPU implementations on large-scale graphs.

Lastly, we study the graph representations of structural brain network from the Diffusion Tensor Imaging data for schizophrenic and demographic study. We develop efficient algorithms to generate connectivity-based brain parcellations which significantly reduce the processing time of the raw brain connectivity graphs. Moreover, the structural brain networks built from the new brain parcellations have better discriminative power compared with existing brain parcellations and atlases.

There are a number of additional research directions worth pursuing.

In Chapter 3, we described a general graph coarsening method based on the spectral graph theory. We explore the relationship between the graph coarsening process and normalized graph Laplacian and derive the graph coarsening method based on the insight. One potential research direction is to extend the results to the *combinatorial Laplacian*, which will complement the study presented in this work. Another line of extension is to apply the spectral coarsening methods to applications involving large-scale graphs.

In Chapter 4, we described a general framework to represent graphs as weighted combinations of sequence representations. The key part is to define the sequence representation model and weight functions associated with the sequences. One example of the graph representation model is presented in our work. There is potential to further improve the effectiveness with novel combinations of sequence and weight functions.

In Chapter 5, we provided an implementation of the spectral clustering algorithm. The proposed implementation consists two separate subroutines, i.e. eigenvalue computation and k-means clustering, which can be independently used for other applications. Another research direction is to support graphs with sizes that don't fit in the GPU and CPU memory of a single

machine where more advanced techniques such as Message Passing Interface (MPI) are needed to cluster very large-scale graphs.

In Chapter 6, we described a brain parcellation method to build structural brain networks from DTI data. Similar methods can be extended to other imaging modalities such as functional MRI. In addition, there are opportunities to extend the method to other brain disorder and demographic study.

Appendix A: Supplementary Materials of Chapter 3

A.1 Proof of Proposition 3.4.1, 3.4.2

For the simplicity of the proof, we use the $\boldsymbol{\mathcal{L}}^{rw} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W}$ to replace the original normalized Laplacian $\boldsymbol{\mathcal{L}}$ to compute the Laplacian eigenvalues. Note that $\boldsymbol{\mathcal{L}}^{rw}$ has the same set of eigenvalues as the original normalized Laplacian $\boldsymbol{\mathcal{L}}$ and the relation of the eigenvalues and eigenvectors satisfy,

$$\boldsymbol{\mathcal{L}}^{rw} = \boldsymbol{D}^{-1/2}\boldsymbol{\mathcal{L}}\boldsymbol{D}^{1/2}, \quad \boldsymbol{u}^{rw} = \boldsymbol{D}^{-1/2}\boldsymbol{u}$$

A.1.1 Proof of Proposition 3.4.1

Proof. We show that under the assumption above, the eigenvalues of the original normalized Laplacian contain the eigenvalues of coarse graph $\mathcal{G}_c$ plus eigenvalue 1 with $N-n$ multiplicities.

The random-walk Laplacian of the coarse graph satisfies,

$$\begin{aligned}
\boldsymbol{\mathcal{L}}_c^{rw} &= \boldsymbol{I}_n - \boldsymbol{D}_c^{-1}\boldsymbol{W}_c \\
&= \boldsymbol{P}\boldsymbol{I}_N\boldsymbol{P}^{\mp} - \boldsymbol{P}\boldsymbol{D}\boldsymbol{P}^{\mp}\boldsymbol{P}\boldsymbol{W}\boldsymbol{P}^{\mp} \\
&= \boldsymbol{P}\boldsymbol{I}_N\boldsymbol{P}^{\mp} - \boldsymbol{P}\boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{P}^{\mp} \\
&= \boldsymbol{P}(\boldsymbol{I}_N - \boldsymbol{D}^{-1}\boldsymbol{W})\boldsymbol{P}^{\mp} \\
&= \boldsymbol{P}\boldsymbol{\mathcal{L}}^{rw}\boldsymbol{P}^{\mp}
\end{aligned}$$

The third equation holds because of the assumption in Equation (9). Then, the eigenvalue and eigenvector of $\boldsymbol{\mathcal{L}}_c^{rw}$ satisfy the following:

$$\begin{aligned}
\boldsymbol{\mathcal{L}}_c^{rw}\boldsymbol{u}^{rw} &= \lambda\boldsymbol{u}_c^{rw} \\
\boldsymbol{P}\boldsymbol{\mathcal{L}}^{rw}\boldsymbol{P}^{\mp}\boldsymbol{u}^{rw} &= \lambda\boldsymbol{u}_c^{rw} \\
\boldsymbol{P}^{\mp}\boldsymbol{P}\boldsymbol{\mathcal{L}}^{rw}\boldsymbol{P}^{\mp}\boldsymbol{u}^{rw} &= \lambda\boldsymbol{P}^{\mp}\boldsymbol{u}_c^{rw} \\
\boldsymbol{\mathcal{L}}^{rw}\boldsymbol{P}^{\mp}\boldsymbol{u}^{rw} &= \lambda\boldsymbol{P}^{\mp}\boldsymbol{u}_c^{rw}
\end{aligned}$$

that is, $\boldsymbol{\mathcal{L}}^{rw}$ has the eigenvalue λ with the corresponding eigenvector $\boldsymbol{P}^{\mp}\boldsymbol{u}^{rw}$.

To see that the original graph contains $N-n$ eigenvalue 1, we consider $\boldsymbol{D}^{-1}\boldsymbol{W} = \boldsymbol{I}_N - \boldsymbol{\mathcal{L}}^{rw}$ which consists of rows of normalized edge weights with row i as $\frac{\boldsymbol{w}(i)}{d(i)}$. From the assumption in Equation (9), we have identical rows for each partition $\mathcal{S}_r$. Thus $\boldsymbol{D}^{-1}\boldsymbol{W}$ is at most rank-n, which indicates $\boldsymbol{\mathcal{L}}^{rw}$ contains $N - n$ eigenvalue 1.

Thus, the original normalized Laplacian has the same eigenvalues as the lifted graph. Both

definition of spectral distances are 0. □

A.1.2 Proof of Proposition 3.4.2

Proof. The normalized Laplacian of the original graph can be viewed as a perturbation of the normalized Laplacian of the lifted graph as

$$\boldsymbol{\mathcal{L}}^{rw} = \boldsymbol{\mathcal{L}}_l^{rw} + \boldsymbol{E},$$

where $\boldsymbol{E}$ is the perturbation matrix.

We expand the entries of $\boldsymbol{\mathcal{L}}^{rw}$ as follows:

$$\boldsymbol{\mathcal{L}}^{rw}(i,j) = \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}(i,j)}{d(i)}.$$

As the coarse graph is coarsened from merging one pair of nodes, the edge weights of the lifted graph $\mathcal{G}_l$ can be expressed as,

$$\boldsymbol{W}_l(i,j) = \begin{cases} \frac{\boldsymbol{W}(a,a)+\boldsymbol{W}(a,b)+\boldsymbol{W}(b,a)+\boldsymbol{W}(b,b)}{4} & \text{if } i \in \{a,b\} \text{ and } j \in \{a,b\} \\ \frac{\boldsymbol{W}(a,j)+\boldsymbol{W}(b,j)}{2} & \text{if } i \in \{a,b\} \text{ and } j \notin \{a,b\} \\ \frac{\boldsymbol{W}(i,a)+\boldsymbol{W}(i,b)}{2} & \text{if } i \notin \{a,b\} \text{ and } j \in \{a,b\} \\ \boldsymbol{W}(i,j) & \text{otherwise.} \end{cases}$$

and the corresponding node degree d_l is

$$d_l(i) = \begin{cases} \frac{d(a)+d(b)}{2} & \text{if } i \in \{a, b\} \\ d(i) & \text{otherwise.} \end{cases}$$

The above imply that $\boldsymbol{\mathcal{L}}_l^{rw}$ can be expanded as follows:

$$\boldsymbol{\mathcal{L}}_l^{rw} = \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}_l(i,j)}{d_l(i)} = \begin{cases} \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}(a,a)+\boldsymbol{W}(a,b)+\boldsymbol{W}(b,a)+\boldsymbol{W}(b,b)}{2(d(a)+d(b))} & \text{if } i \in \{a,b\} \text{ and } j \in \{a,b\} \\ \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}(a,j)+\boldsymbol{W}(b,j)}{d(a)+d(b)} & \text{if } i \in \{a,b\} \text{ and } j \notin \{a,b\} \\ \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}(i,a)+\boldsymbol{W}(i,b)}{2d(i)} & \text{if } i \notin \{a,b\} \text{ and } j \in \{a,b\} \\ \boldsymbol{I}(i,j) - \frac{\boldsymbol{W}(i,j)}{d(i)} & \text{otherwise} \end{cases}$$

and the perturbation matrix $\boldsymbol{E} = \boldsymbol{\mathcal{L}}^{rw} - \boldsymbol{\mathcal{L}}_l^{rw}$ is given by

$$\boldsymbol{E}(i,j) = \begin{cases} \frac{\boldsymbol{W}(i,j)}{d(i)} - \frac{\boldsymbol{W}(a,a)+\boldsymbol{W}(a,b)+\boldsymbol{W}(b,a)+\boldsymbol{W}(b,b)}{2(d(a)+d(b))} & \text{if } i \in \{a,b\} \text{ and } j \in \{a,b\} \\ \frac{\boldsymbol{W}(i,j)}{d(i)} - \frac{\boldsymbol{W}(a,j)+\boldsymbol{W}(b,j)}{d(a)+d(b)} & \text{if } i \in \{a,b\} \text{ and } j \notin \{a,b\} \\ \frac{\boldsymbol{W}(i,j)}{d(i)} - \frac{\boldsymbol{W}(i,a)+\boldsymbol{W}(i,b)}{2d(i)} & \text{if } i \notin \{a,b\} \text{ and } j \in \{a,b\} \\ 0 & \text{otherwise.} \end{cases}$$

From [175], we have the following bound on the eigenvalue gap between $\boldsymbol{\lambda}(i)$ and $\boldsymbol{\lambda}_l(i)$:

$$|\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| \leq \|E\|_2$$

Moreover, [176] proved that the spectral norm $\|E\|_2$ admits the simple upper bound:

$$\|\boldsymbol{E}\|_2^2 \leq \max_{i,j} \boldsymbol{r}_i \boldsymbol{c}_j = \max_i \boldsymbol{r}_i \max_j \boldsymbol{c}_j,$$

where $\boldsymbol{r}_i = \sum_j |\boldsymbol{E}(i,j)|$ and $\boldsymbol{c}_j = \sum_i |\boldsymbol{E}(i,j)|$.

Let us focus on term $\boldsymbol{r}_i$.

Case 1: $i \notin \{a, b\}$,

$$\begin{aligned}\boldsymbol{r}_i &= |\frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(i,a)+\boldsymbol{W}(i,b)}{2d(i)}| + |\frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(i,a)+\boldsymbol{W}(i,b)}{2d(i)}| \\ &= |\frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(i,b)}{d(i)}| \leq \left\| \frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(i,b)}{d(i)} \right\|_1 \leq \epsilon\end{aligned}$$

Case 2: $i \in \{a, b\}$, and suppose $d(a) \leq d(b)$ w.l.o.g.,

$$\begin{aligned}\boldsymbol{r}_i &= |\frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(a,a)+\boldsymbol{W}(a,b)+\boldsymbol{W}(b,a)+\boldsymbol{W}(b,b)}{2(d(a)+d(b))}| + |\frac{\boldsymbol{W}(i,b)}{d(i)} - \frac{\boldsymbol{W}(a,a)+\boldsymbol{W}(a,b)+\boldsymbol{W}(b,a)+}{2(d(a)+d(b))} \\ &+ \sum_{j \notin \{a,b\}} |\frac{\boldsymbol{W}(i,j)}{d(i)} - \frac{\boldsymbol{W}(a,j)+\boldsymbol{W}(b,j)}{d(a)+d(b)}| \\ &\leq |\frac{\boldsymbol{W}(a,a)}{d(a)} - \frac{\boldsymbol{W}(b,a)}{d(b)}| + |\frac{\boldsymbol{W}(a,b)}{d(a)} - \frac{\boldsymbol{W}(b,b)}{d(b)}| + \sum_{j \notin \{a,b\}} |\frac{\boldsymbol{W}(a,j)}{d(a)} - \frac{\boldsymbol{W}(b,j)}{d(b)}| \\ &= \left\| \frac{\boldsymbol{W}(i,a)}{d(i)} - \frac{\boldsymbol{W}(i,b)}{d(i)} \right\|_1 \leq \epsilon \end{aligned} \tag{A.1}$$

We have $\max_i \boldsymbol{r}_i \leq \epsilon$. Similarly, we can show that $\boldsymbol{c}_j \leq \epsilon$. The spectral norm of the perturbation matrix $\boldsymbol{E}$ then is bounded by

$$\|\boldsymbol{E}\|_2 \leq \sqrt{\max_i \boldsymbol{r}_i \max_j \boldsymbol{c}_j} \leq \epsilon. \tag{A.2}$$

Combining the above, we have the bound of each term in the spectral distance as,

$$|\boldsymbol{\lambda}(i) - \boldsymbol{\lambda}_l(i)| \leq \epsilon \tag{A.3}$$

The bounds of the full and partial spectral distance follow the Equation A.3 as they contain N and n eigengap terms respectively. □

A.2 Proof of Corollary 3.5.1

Proof. We denote the intermediate graphs at iteration s as $\mathcal{G}^{(s)}$ with $\mathcal{G}^{(N)}$ as the original graph $\mathcal{G}$ and $\mathcal{G}^{(n)}$ as the coarse graph $\mathcal{G}_c$. From Proposition 4.2 and the spectral distance is a distance metric over the Laplacian eigenvalues, we have the following,

$$SD_{full}(\mathcal{G}, \mathcal{G}_c) \leq \sum_{s=N}^{n+1} SD_{full}(\mathcal{G}^{(s)}, \mathcal{G}^{(s-1)}) \leq N \sum_{s=N}^{n+1} \epsilon_s$$

and

$$SD_{part}(\mathcal{G}, \mathcal{G}_c) \leq \sum_{s=N}^{n+1} SD_{part}(\mathcal{G}^{(s)}, \mathcal{G}^{(s-1)}) \leq N \sum_{s=N}^{n+1} \epsilon_s$$

□

A.3 Proof of Theorem 3.5.2

Proof. We rewrite the objective of the k-means algorithm as the following,

$$\mathcal{F}(\boldsymbol{U}, \mathcal{P}) = \sum_{i=1}^{N} \left(\boldsymbol{r}(i) - \sum_{j \in \mathcal{S}_i} \frac{\boldsymbol{r}(j)}{|\mathcal{S}_i|} \right)^2 = \|\boldsymbol{U} - \boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{U}\|_F^2,$$

where the matrix $\boldsymbol{C} \in \mathbb{R}^{n \times N}$ is the normalized coarsening matrix corresponding to the graph partition $\mathcal{P}$. With the notation $\boldsymbol{\Pi} = \boldsymbol{C}\boldsymbol{C}^\top$ and $\boldsymbol{\Pi}^\perp = \boldsymbol{I} - \boldsymbol{\Pi}$ from Section 3.3.2, the k-means objective is written as

$$\mathcal{F}(\boldsymbol{U}, \mathcal{P}) = \|\boldsymbol{\Pi}^\perp \boldsymbol{U}\|_F^2.$$

We express the partial spectral distance as in Definition 3.4.5

$$SD_{\text{part}}(\mathcal{G}, \mathcal{G}_c) = \sum_{i=1}^{k_1} (\boldsymbol{\lambda}_c(i) - \boldsymbol{\lambda}(i)) + \sum_{j=k_2+1}^{N} (\boldsymbol{\lambda}(j) - \boldsymbol{\lambda}_c(j+n-N)) \tag{A.4}$$

where $k_1 = \arg\max_i \{i : \boldsymbol{\lambda}_c(i) < 1\}, k_2 = N - n + k_1$.

Because of the interlacing property 3.4.1, we remove the absolute sign on the terms.

Correspondingly, we separate the k-means cost in two terms as,

$$\mathcal{F}(\boldsymbol{U}, \boldsymbol{C}) = \|\boldsymbol{U}_{k_1} - \boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{U}_{k_1}\|_F^2 + \|\boldsymbol{U}'_{k_2} - \boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{U}'_{k_2}\|_F^2 = \|\boldsymbol{\Pi}^\perp \boldsymbol{U}_{k_1}\|_F^2 + \|\boldsymbol{\Pi}^\perp \boldsymbol{U}'_{k_2}\|_F^2$$

where $\boldsymbol{U}_{k_1}$ and $\boldsymbol{U}'_{k_2}$ denote the eigenvectors corresponding to the smallest k_1 and largest $n - k_1$ eigenvalues of the original graph. We also denote $\delta_{k_1} = \|\boldsymbol{\Pi}^\perp \boldsymbol{U}_{k_1}\|_F^2$ and $\delta'_{k_2} = \|\boldsymbol{\Pi}^\perp \boldsymbol{U}'_{k_2}\|_F^2$.

We will prove the results of the two terms separately.

For the first k_1 eigenvalue gaps, we start by the following generalization of the Courant-Fisher theorem:

$$\sum_{i\leq k_1}\boldsymbol{\lambda}_c(i)=\min_{\boldsymbol{V}^\top\boldsymbol{V}=\boldsymbol{I}_k}\text{tr}(\boldsymbol{V}^\top\boldsymbol{\mathcal{L}}_c\boldsymbol{V}).$$

We write $\boldsymbol{\mathcal{L}}=\boldsymbol{S}^\top\boldsymbol{S}$ where $\boldsymbol{S}\in\mathbb{R}^{M\times N}$ denotes the incidence matrix of the normalized Laplacian $\boldsymbol{\mathcal{L}}$ with the following form

$$\boldsymbol{S}(v,e)=\begin{cases}\frac{1}{\sqrt{d(i)}},\text{if } v=i\\ -\frac{1}{\sqrt{d(j)}},\text{if } v=j,\end{cases}$$

where $e\in\mathcal{E}$ with i and j as the connecting nodes. Then, the first k_1 eigenvalues are

$$\sum_{i\leq k_1}\boldsymbol{\lambda}_c(i)=\min_{\boldsymbol{V}^\top\boldsymbol{V}=\boldsymbol{I}_k}\text{tr}(\boldsymbol{V}^\top\boldsymbol{C}\boldsymbol{S}^\top\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{V})=\min_{\boldsymbol{V}^\top\boldsymbol{V}=\boldsymbol{I}_k}\|\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{V}\|_F^2$$

Set $\boldsymbol{Z}=\boldsymbol{C}\boldsymbol{U}_{k_1}$, and suppose that $\boldsymbol{Z}^\top\boldsymbol{Z}$ is invertible (this will be ensured in the following). We select

$$\boldsymbol{V}=\boldsymbol{Z}(\boldsymbol{Z}^\top\boldsymbol{Z})^{-1/2}$$

for which we have

$$\boldsymbol{V}^\top\boldsymbol{V}=(\boldsymbol{Z}^\top\boldsymbol{Z})^{-1/2}\boldsymbol{Z}^\top\boldsymbol{Z}(\boldsymbol{Z}^\top\boldsymbol{Z})^{-1/2}=\boldsymbol{I}_{k_1}$$

as required.

We expand the sum of eigenvalues as follows:

$$\sum_{i\leq k_1}\boldsymbol{\lambda}_i=\min_{\boldsymbol{V}^\top\boldsymbol{V}=\boldsymbol{I}_{k_1}}\|\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{V}\|_{\boldsymbol{F}}^2\leq\|\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{Z}(\boldsymbol{Z}^\top\boldsymbol{Z})^{-1/2}\|_F^2\leq\|\boldsymbol{S}\boldsymbol{C}^\top\boldsymbol{C}\boldsymbol{U}_{k_1}\|_F^2\,\|(\boldsymbol{Z}^\top\boldsymbol{Z})^{-1/2}\|_2^2.$$

and use the matrix $\mathbf{\Pi} = \boldsymbol{C}^\top \boldsymbol{C}$ and $\mathbf{\Pi}^\perp = \boldsymbol{I} - \mathbf{\Pi}$ defined in Section 3.3.2.

For the first term, we employ the triangle inequality.

$$\begin{aligned}
\|\boldsymbol{S}\boldsymbol{C}^\top \boldsymbol{C}\boldsymbol{U}_{k_1}\|_F^2 &= \|\boldsymbol{S}\mathbf{\Pi}\boldsymbol{U}_{k_1}\|_F^2 \\
&= (\|\boldsymbol{S}(\boldsymbol{I} - \mathbf{\Pi}^\perp)\boldsymbol{U}_{k_1}\|_F)^2 \\
&\leq (\|\boldsymbol{S}\boldsymbol{U}_{k_1}\|_F + \|\boldsymbol{S}\mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_F)^2 \\
&\leq (\|\boldsymbol{S}\boldsymbol{U}_{k_1}\|_F + \|\boldsymbol{S}\mathbf{\Pi}^\perp\|_2 \|\mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_F)^2 \qquad \text{(A.5)}
\end{aligned}$$

The result for $\|\boldsymbol{S}\boldsymbol{U}_{k_1}\|_F$ is

$$\|\boldsymbol{S}\boldsymbol{U}_{k_1}\|_F = \sqrt{\text{tr}(\boldsymbol{U}_{k_1}^\top \boldsymbol{S}^\top \boldsymbol{S}\boldsymbol{U}_{k_1})} = \sqrt{\sum_{i \leq k_1} \boldsymbol{\lambda}(i)}.$$

On the other hand, the norm $\|\boldsymbol{S}\mathbf{\Pi}^\perp\|_2$ is bounded by

$$\|\boldsymbol{S}\mathbf{\Pi}^\perp\|_2 = \sqrt{\lambda_{\max}(\mathbf{\Pi}^\perp \boldsymbol{S}^\perp \boldsymbol{S}\mathbf{\Pi}^\perp)} = \sqrt{\lambda_{\max}(\boldsymbol{\mathcal{L}})} \leq \sqrt{2}$$

To analyze the second term, denote by σ_i the singular values of the $k \times k$ matrix $\boldsymbol{U}_{k_1}^\top \mathbf{\Pi} \boldsymbol{U}_{k_1}$ and $\delta_{k_1} = \mathcal{F}(\boldsymbol{U}_{k_1}, C) = \|\mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_F^2$. The following inequality holds:

$$\delta_{k_1} \geq \|\mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_2^2 = \|\boldsymbol{U}_{k_1}^\top \mathbf{\Pi}^\perp \mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_2 = \|\boldsymbol{U}_{k_1}^\top \mathbf{\Pi}^\perp \boldsymbol{U}_{k_1}\|_2 = \|\boldsymbol{U}_{k_1}^\top (\boldsymbol{I} - \mathbf{\Pi})\boldsymbol{U}_{k_1}\|_2 = \|\boldsymbol{I}_k - \boldsymbol{U}_{k_1}^\top \mathbf{\Pi} \boldsymbol{U}_{k_1}\|_2$$

The inequality is equivalent to asserting that the singular values of $\boldsymbol{U}_{k_1}^\top \mathbf{\Pi} \boldsymbol{U}_{k_1}$ are concentrated around one, i.e.,

$$1 - \delta_{k_1} \leq \sigma_i \leq 1 + \delta_{k_1} \;\text{ for all }\; i \leq k_1.$$

It follows that the smallest eigenvalue of the PSD matrix $\boldsymbol{Z}^\top \boldsymbol{Z}$ is bounded by

$$\begin{aligned}
\boldsymbol{\lambda}_1(\boldsymbol{Z}^\top \boldsymbol{Z}) &= \min_{\|\boldsymbol{x}\|_2=1} x^\top \boldsymbol{U}_{k_1}^\top \boldsymbol{C}^\top \boldsymbol{C} \boldsymbol{U}_{k_1} \boldsymbol{x} \\
&= \min_{\boldsymbol{x} \in \text{span}(\boldsymbol{U}_{k_1}),\ \|\boldsymbol{x}\|_2=1} x^\top \boldsymbol{C}^\top \boldsymbol{C} x \\
&= \min_{\boldsymbol{x} \in \text{span}(\boldsymbol{U}_{k_1}),\ \|\boldsymbol{x}\|_2=1} \boldsymbol{x}^\top \boldsymbol{\Pi} x \\
&\geq 1 - \delta_{k_1}
\end{aligned}$$

We deduce that the matrix is invertible when $\delta_{k_1} < 1$ and $\boldsymbol{C}$ is full row-rank. In addition, we have

$$\|(\boldsymbol{Z}^\top \boldsymbol{Z})^{-1/2}\|_2^2 = \|(\boldsymbol{Z}^\top \boldsymbol{Z})^{-1}\|_2 \leq \frac{1}{1-\delta_{k_1}}.$$

Putting the bounds together, gives

$$\sum_{i \leq k_1} \boldsymbol{\lambda}_c(i) \leq \frac{\left(\sqrt{\sum_{i \leq k} \boldsymbol{\lambda}(i)} + \sqrt{2\,\delta_{k_1}}\right)^2}{1-\delta_{k_1}}$$

or equivalently

$$\sum_{i \leq k} (\boldsymbol{\lambda}_c(i) - \boldsymbol{\lambda}(i)) \leq \frac{\left(\sqrt{\sum_{i \leq k} \boldsymbol{\lambda}(i)} + \sqrt{2\,\delta_{k_1}}\right)^2}{1-\delta_{k_1}} - \sum_{i \leq k_1} \boldsymbol{\lambda}(i) = \frac{\delta_{k_1}(2 + \sum_{i \leq k} \boldsymbol{\lambda}(i)) + \sqrt{8\delta_{k_1} \sum_{i \leq k_1} \boldsymbol{\lambda}(i)}}{1-\delta_{k_1}}$$

To prove the result for the second term in equation A.4, we introduce the *signless normalized Laplacian* $\tilde{\boldsymbol{\mathcal{L}}} = \boldsymbol{I} + \boldsymbol{D}^{-1/2} \boldsymbol{W} \boldsymbol{D}^{-1/2}$ to obtain the results of the second term in Equation. A.5. We follow the similar arguments using the signless normalized Laplacian. Note that the spectral

properties of signless normalized Laplacian follow the relation:

$$\tilde{\boldsymbol{\lambda}}(i) = 2 - \boldsymbol{\lambda}(N+1-i) \text{ and } \tilde{\boldsymbol{U}}(i) = \boldsymbol{U}(N+1-i)$$

Then, the eigengaps between largest eigenvalues abide to

$$\begin{aligned}\sum_{j=k_2+1}^{N} (\boldsymbol{\lambda}(j) - \boldsymbol{\lambda}_c(j+n-N)) &= \sum_{j=1}^{n-k} \boldsymbol{\lambda}(N+1-j) - \boldsymbol{\lambda}_c(n+1-j) \\ &= \sum_{j=1}^{n-k} (\tilde{\boldsymbol{\lambda}}_c(j) - \tilde{\boldsymbol{\lambda}}(j)) \\ &\leq \frac{\delta'_{k_2}(\sum_{j\leq n-k_1} 2 + \tilde{\boldsymbol{\lambda}}(j)) + \sqrt{8\delta'_{k_2} \sum_{j\leq n-k_1} \tilde{\boldsymbol{\lambda}}(j)}}{1-\delta'_{k_2}}.\end{aligned}$$

Combining the above, we obtain the following result:

$$\begin{aligned}SD(\mathcal{G}, \mathcal{G}_c) &\leq \frac{\delta_{k_1}(2 + \sum_{i\leq k} \boldsymbol{\lambda}(i)) + \sqrt{8\delta_{k_1} \sum_{i\leq k_1} \boldsymbol{\lambda}(i)}}{1-\delta_{k_1}} + \frac{\delta'_{k_2}(\sum_{j\leq n-k_1} 2 + \tilde{\boldsymbol{\lambda}}(j)) + \sqrt{8\delta'_{k_2} \sum_{j\leq n-k_1} \tilde{\boldsymbol{\lambda}}(j)}}{1-\delta'_{k_2}} \\ &\leq \frac{(n+2)\mathcal{F}(\boldsymbol{U}, \boldsymbol{C}) + 4\sqrt{\mathcal{F}(\boldsymbol{U}, \boldsymbol{C})}}{1-\mathcal{F}(\boldsymbol{U}, \boldsymbol{C})}\end{aligned}$$

In the last step, we use the following bounds:

$$\delta_{k_1} \leq \mathcal{F}(\boldsymbol{U}, \boldsymbol{C}), \delta'_{k_2} \leq \mathcal{F}(\boldsymbol{U}, \boldsymbol{C}),$$

$$\sum_{i\leq k_1} \boldsymbol{\lambda}(i) \leq k_1, \sum_{j\leq n-k_1} \tilde{\boldsymbol{\lambda}}(j) \leq n - k_1$$

$$\sqrt{k_1} + \sqrt{n-k_1} \leq \sqrt{2n}.$$

□

A.4 Additional Material for Experiments

A.4.1 Graph Classification Dataset

The statistics of the graph classification benchmarks are in Table A.1.

Table A.1: Statistics of the graph benchmark datasets.

Datasets	**MUTAG**	**ENZYMES**	**NCI1**	**NCI109**	**PROTEINS**	**PTC**
Sample size	188	600	4110	4127	1108	344
Average $\|V\|$	17.93	32.63	29.87	29.68	39.06	14.29
Average $\|E\|$	19.79	62.14	32.3	32.13	72.70	14.69
# classes	2	6	2	2	2	2

A.4.2 Definition of Normalized Mutual Information

We denote C_1 and C_2 are two where $C(i)$ represents the set of nodes with label i. We define the NMI as,

$$NMI(C_1, C_2) = \frac{MI(C_1, C_2)}{\frac{1}{2}(H(C_1) + H(C_2))}$$

where $MI(C_1, C_2)$ is the mutual information defined as,

$$MI(C_1, C_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} p(C_1(i) \cap C_2(j)) \log \left(\frac{p\left(C_1(i) \cap C_2(j)\right)}{p\left(C_1(i)\right) p\left(C_2(j)\right)} \right)$$

$H(C)$ is the entropy defined as,

$$H(C) = -\sum_{i=1}^{n} p(C(i)) \log p(C(i))$$

The probability $p(C(i))$ is approximated as the ratio of partition i as $p(C(i)) = \frac{|C(i)|}{N}$.

Bibliography

[1] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[2] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

[3] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[4] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *ICML*, 2017.

[5] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[6] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[7] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[8] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Octave Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *Neuroimage*, 15(1):273–289, 2002.

[9] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

[10] David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic spectral vertex sparsifiers and applications. *arXiv preprint arXiv:1906.10530*, 2019.

[11] Vikas K Garg and Tommi Jaakkola. Solving graph compression via optimal transport. *arXiv preprint arXiv:1905.12158*, 2019.

[12] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432. ACM, 2008.

[13] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[14] Jie Chen, Yousef Saad, and Zechen Zhang. Graph coarsening: from scientific computing to machine learning. *SeMA Journal*, pages 1–37, 2022.

[15] Nafiseh Ghoroghchian, Gautam Dasarathy, and Stark Draper. Graph community detection from coarse measurements: Recovery conditions for the coarsened weighted stochastic block model. In *International Conference on Artificial Intelligence and Statistics*, pages 3619–3627. PMLR, 2021.

[16] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[17] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 10–pp. IEEE, 2006.

[18] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[19] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[20] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[21] Timothee Cour, Florence Benezit, and Jianbo Shi. Spectral segmentation with multiscale graph decomposition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 1124–1131. IEEE, 2005.

[22] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[23] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[24] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[25] Lawrence K Saul, Kilian Q Weinberger, Fei Sha, Jihun Ham, and Daniel D Lee. Spectral methods for dimensionality reduction. *Semi-supervised learning*, 3, 2006.

[26] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *arXiv preprint arXiv:1611.08097*, 2016.

[27] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[28] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2016.

[29] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[30] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[32] Andreas Loukas. What graph neural networks cannot learn: Depth vs width. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[33] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[34] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

[35] Tailin Wu, Hongyu Ren, Pan Li, and Jure Leskovec. Graph information bottleneck. *Advances in Neural Information Processing Systems*, 33:20437–20448, 2020.

[36] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[37] AA Leman and Boris Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[38] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[39] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[40] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3438–3445, 2020.

[41] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167, 2019.

[42] Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4363–4371, 2019.

[43] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[44] Jesper LR Andersson, Mark Jenkinson, Stephen Smith, et al. Non-linear registration, aka spatial normalisation fmrib technical report tr07ja2. *FMRIB Analysis Group of the University of Oxford*, 2007.

[45] TEJ Behrens, H Johansen Berg, Saad Jbabdi, MFS Rushworth, and MW Woolrich. Probabilistic diffusion tractography with multiple fibre orientations: What can we gain? *Neuroimage*, 34(1):144–155, 2007.

[46] TEJ Behrens, MW Woolrich, M Jenkinson, H Johansen-Berg, RG Nunes, S Clare, PM Matthews, JM Brady, and SM Smith. Characterization and propagation of uncertainty in diffusion-weighted mr imaging. *Magnetic resonance in medicine*, 50(5):1077–1088, 2003.

[47] Korbinian Brodmann. *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Barth, 1909.

[48] Simon B Eickhoff, Klaas E Stephan, Hartmut Mohlberg, Christian Grefkes, Gereon R Fink, Katrin Amunts, and Karl Zilles. A new spm toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. *Neuroimage*, 25(4):1325–1335, 2005.

[49] Stephen M Smith, Mark Jenkinson, Heidi Johansen-Berg, Daniel Rueckert, Thomas E Nichols, Clare E Mackay, Kate E Watkins, Olga Ciccarelli, M Zaheer Cader, Paul M Matthews, et al. Tract-based spatial statistics: voxelwise analysis of multi-subject diffusion data. *Neuroimage*, 31(4):1487–1505, 2006.

[50] Rahul S Desikan, Florent Ségonne, Bruce Fischl, Brian T Quinn, Bradford C Dickerson, Deborah Blacker, Randy L Buckner, Anders M Dale, R Paul Maguire, Bradley T Hyman, et al. An automated labeling system for subdividing the human cerebral cortex on mri scans into gyral based regions of interest. *Neuroimage*, 31(3):968–980, 2006.

[51] Olaf Sporns. Structure and function of complex brain networks. *Dialogues in clinical neuroscience*, 15(3):247, 2013.

[52] Marcus Kaiser. A tutorial in connectome analysis: topological and spatial features of brain networks. *Neuroimage*, 57(3):892–907, 2011.

[53] Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: uses and interpretations. *Neuroimage*, 52(3):1059–1069, 2010.

[54] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[55] Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Physical review letters*, 87(19):198701, 2001.

[56] Alex Fornito, Andrew Zalesky, and Edward Bullmore. *Fundamentals of brain network analysis*. Academic press, 2016.

[57] Yike Liu, Abhilash Dighe, Tara Safavi, and Danai Koutra. A graph summarization: A survey. *arXiv preprint arXiv:1612.04883*, 2016.

[58] Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. In *International Conference on Learning and Intelligent Optimization*, pages 191–205. Springer, 2009.

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

[60] Gecia Bravo Hermsdorff and Lee M Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. *arXiv preprint arXiv:1902.09702*, 2019.

[61] Edwin R Van Dam and Willem H Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003.

[62] Anirban Banerjee. *The spectrum of the graph Laplacian as a tool for analyzing structure and evolution of networks*. PhD thesis, 2008.

[63] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[64] Irena Jovanović and Zoran Stanić. Spectral distances of graphs. *Linear Algebra and its Applications*, 436(5):1425–1435, 2012.

[65] 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, 2013.

[66] Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. *arXiv preprint arXiv:1802.07510*, 2018.

[67] Jiao Gu, Bobo Hua, and Shiping Liu. Spectral distances on graphs. *Discrete Applied Mathematics*, 190:56–74, 2015.

[68] Irena M Jovanovic. Some results on spectral distances of graphs. *Rev. Un. Mat. Argentina*, 56(2), 2015.

[69] Irena Jovanović and Zoran Stanić. Spectral distances of graphs based on their different matrix representations. *Filomat*, 28(4):723–734, 2014.

[70] Manish Purohit, B Aditya Prakash, Chanhyun Kang, Yao Zhang, and VS Subrahmanian. Fast influence-based coarsening for large networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1296–1305. ACM, 2014.

[71] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2347–2356. ACM, 2018.

[72] Austin R. Dong, Kun Benson and David Bindel. Network density of states. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[73] Steven Kay Butler. *Eigenvalues and structures of graphs*. ProQuest, 2008.

[74] Steve Butler. Interlacing for weighted graphs using the normalized laplacian. *Electronic Journal of Linear Algebra*, 16(1):8, 2007.

[75] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973. ACM, 2011.

[76] Irene Sciriha. A characterization of singular graphs. *Electronic Journal of Linear Algebra*, 16(1):38, 2007.

[77] Ali Sltan AL-Tarimshawy. Singular graphs. *arXiv preprint arXiv:1806.07786*, 2018.

[78] Nicolas Tremblay and Andreas Loukas. Approximating spectral clustering via sampling: a review. In *Sampling Techniques for Supervised or Unsupervised Tasks*, pages 129–183. Springer, 2020.

[79] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11), 2007.

[80] Yu Jin and Joseph F JaJa. Learning graph-level representations with gated recurrent neural networks. *arXiv preprint arXiv:1805.07683*, 2018.

[81] Hae-Jeong Park and Karl Friston. Structural and functional brain networks: from connections to cognition. *Science*, 342(6158):1238411, 2013.

[82] Yu Jin, Joseph F JaJa, Rong Chen, and Edward H Herskovits. A data-driven approach to extract connectivity structures from diffusion tensor imaging data. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 944–951. IEEE, 2015.

[83] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[84] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[85] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

[86] Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Transactions on Information Theory*, 62(1):471–487, 2015.

[87] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

[88] Martin Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017.

[89] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[90] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[91] Hongyang Gao and Shuiwang Ji. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

[92] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*, 2020.

[93] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3419–3430, 2020.

[94] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[95] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[96] Zhongyu Huang, Yingheng Wang, Chaozhuo Li, and Huiguang He. Going deeper into permutation-sensitive graph neural networks. *arXiv preprint arXiv:2205.14368*, 2022.

[97] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

[98] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. *arXiv preprint arXiv:1903.02541*, 2019.

[99] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[100] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[101] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 2112–2118, 2021.

[102] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[103] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[104] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[105] 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*, pages 701–710. ACM, 2014.

[106] 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*, pages 855–864. ACM, 2016.

[107] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

[108] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[109] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[110] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 2126–2132, 2020.

[111] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1026–1037, 2021.

[112] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido F Montúfar, and Michael Bronstein. Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, pages 2625–2640, 2021.

[113] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[114] Léon Bottou and Yann L Cun. Large scale online learning. In *Advances in neural information processing systems*, pages 217–224, 2004.

[115] Alan L Yuille. The convergence of contrastive divergences. In *Advances in neural information processing systems*, pages 1593–1600, 2005.

[116] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[117] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[118] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[119] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[120] Yves van Gennip, Blake Hunter, Raymond Ahn, Peter Elliott, Kyle Luh, Megan Halvorson, Shannon Reid, Matthew Valasik, James Wo, George E Tita, et al. Community detection using spectral clustering on sparse geosocial data. *SIAM Journal on Applied Mathematics*, 73(1):67–83, 2013.

[121] Yu Jin, Joseph F JaJa, Rong Chen, and Edward H Herskovits. A data-driven approach to extract connectivity structures from diffusion tensor imaging data. In *2015 IEEE International Conference on Big Data*. IEEE, 2015.

[122] R Cameron Craddock, G Andrew James, Paul E Holtzheimer, Xiaoping P Hu, and Helen S Mayberg. A whole brain fmri atlas generated via spatially constrained spectral clustering. *Human brain mapping*, 33(8):1914–1928, 2012.

[123] Dirk Eddelbuettel. Benchmarking single-and multi-core blas implementations and gpus for use with r, 2010.

[124] Christopher Cullinan, Christopher Wyant, Timothy Frattesi, and Xinming Huang. Computing performance benchmarks among cpu, gpu, and fpga. *Internet: www. wpi. edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking Final*, 2013.

[125] Changmin Lee, Won Woo Ro, and Jean-Luc Gaudiot. Boosting cuda applications with cpu–gpu hybrid computing. *International Journal of Parallel Programming*, 42(2):384–404, 2014.

[126] JI Agulleiro, F Vazquez, EM Garzon, and JJ Fernandez. Hybrid computing: Cpu+ gpu co-processing and its application to tomographic reconstruction. *Ultramicroscopy*, 115:109–114, 2012.

[127] Jing Wu and Joseph JaJa. Optimized fft computations on heterogeneous platforms with application to the poisson equation. *Journal of Parallel and Distributed Computing*, 74(8):2745–2756, 2014.

[128] Jing Wu and Joseph JaJa. Achieving native gpu performance for out-of-card large matrix multiplication. *Parallel Processing Letters*, 2015.

[129] Kenli Li, Wangdong Yang, and Keqin Li. A hybrid parallel solving algorithm on gpu for quasi-tridiagonal system of linear equations. *IEEE Transactions on Parallel and Distributed Systems*.

[130] Elena N Akimova and Dmitry V Belousov. Parallel algorithms for solving linear systems with block-tridiagonal matrices on multi-core cpu with gpu. *Journal of Computational Science*, 3(6):445–449, 2012.

[131] Jing Zheng, Wenguang Chen, Yurong Chen, Yimin Zhang, Ying Zhao, and Weimin Zheng. Parallelization of spectral clustering algorithm on multi-core processors and gpgpu. In *Computer Systems Architecture Conference, 2008. ACSAC 2008. 13th Asia-Pacific*, pages 1–8. IEEE, 2008.

[132] Kiran Kumar Matam and Kishore Kothapalli. Gpu accelerated lanczos algorithm with applications. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 71–76. IEEE, 2011.

[133] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

[134] Yangqiu Song, Wen-Yen Chen, Hongjie Bai, Chih-Jen Lin, and Edward Y Chang. Parallel spectral clustering. In *Machine Learning and Knowledge Discovery in Databases*, pages 374–389. Springer, 2008.

[135] Serafeim Tsironis, Mauro Sozio, Michalis Vazirgiannis, and LIX-Ecole Poltechnique. Accurate spectral clustering for community detection in mapreduce. Citeseer.

[136] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

[137] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[138] William Pentney and Marina Meila. Spectral clustering of biological sequence data. In *AAAI*, volume 5, pages 845–850, 2005.

[139] Desmond J Higham, Gabriela Kalna, and Milla Kibble. Spectral clustering and its use in bioinformatics. *Journal of computational and applied mathematics*, 204(1):25–37, 2007.

[140] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graph. volume 5, pages 76–84. SIAM, 2005.

[141] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert E Tarjan. Clustering social networks. In *Algorithms and Models for the Web-Graph*, pages 56–67. Springer, 2007.

[142] Adrian-Gabriel Chifu, Florentina Hristea, Josiane Mothe, and Marius Popescu. Word sense discrimination in information retrieval: A spectral clustering-based approach. *Information Processing & Management*, 51(2):16–31, 2015.

[143] Brian McFee and Daniel PW Ellis. Analyzing song structure with spectral clustering.

[144] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*, volume 6. SIAM, 1998.

[145] Richard B Lehoucq and Danny C Sorensen. Deflation techniques for an implicitly restarted arnoldi iteration. *SIAM Journal on Matrix Analysis and Applications*, 17(4):789–821, 1996.

[146] Danny C Sorensen. Implicit application of polynomial filters in ak-step arnoldi method. *Siam journal on matrix analysis and applications*, 13(1):357–385, 1992.

[147] Mario Zechner and Michael Granitzer. Accelerating k-means on the graphics processor via cuda. In *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on*, pages 7–15. IEEE, 2009.

[148] Jiadong Wu and Bo Hong. An efficient k-means algorithm on cuda. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1740–1749. IEEE, 2011.

[149] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[150] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[151] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

[152] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[153] Olaf Sporns, Dante R Chialvo, Marcus Kaiser, and Claus C Hilgetag. Organization, development and function of complex brain networks. *Trends in cognitive sciences*, 8(9):418–425, 2004.

[154] Elise GP Dopper, Serge ARB Rombouts, Lize C Jiskoot, Tom den Heijer, J Roos A de Graaf, Inge de Koning, Anke R Hammerschlag, Harro Seelaar, William W Seeley, Ilya M Veer, et al. Structural and functional brain connectivity in presymptomatic familial frontotemporal dementia. *Neurology*, 83(2):e19–e26, 2014.

[155] Martijn P van den Heuvel, René CW Mandl, Cornelis J Stam, René S Kahn, and Hilleke E Hulshoff Pol. Aberrant frontal and temporal complex network structure in schizophrenia: a graph theoretical analysis. *The Journal of Neuroscience*, 30(47):15915–15926, 2010.

[156] K Amunts, A Schleicher, and K Zilles. Cytoarchitecture of the cerebral cortex—more than localization. *Neuroimage*, 37(4):1061–1065, 2007.

[157] Oskar Vogt. Die myeloarchitektonik des isocortex parietalis. *J Psychol Neurol*, 18:379–390, 1911.

[158] Karl Zilles and Katrin Amunts. Receptor mapping: architecture of the human cerebral cortex. *Current opinion in neurology*, 22(4):331–339, 2009.

[159] Michael Ruschel, Thomas R Knösche, Angela D Friederici, Robert Turner, Stefan Geyer, and Alfred Anwander. Connectivity architecture and subdivision of the human inferior parietal cortex revealed by diffusion mri. *Cerebral Cortex*, 24(9):2436–2448, 2014.

[160] Rogier B Mars, Saad Jbabdi, Jérôme Sallet, Jill X O'Reilly, Paula L Croxson, Etienne Olivier, MaryAnn P Noonan, Caroline Bergmann, Anna S Mitchell, Mark G Baxter, et al. Diffusion-weighted imaging tractography-based parcellation of the human parietal cortex and comparison with human and macaque resting-state functional connectivity. *The Journal of Neuroscience*, 31(11):4087–4100, 2011.

[161] Rogier B Mars, Jérôme Sallet, Urs Schüffelgen, Saad Jbabdi, Ivan Toni, and Matthew FS Rushworth. Connectivity-based subdivisions of the human right "temporoparietal junction area": evidence for different areas participating in different cortical networks. *Cerebral cortex*, 22(8):1894–1903, 2012.

[162] Jérôme Sallet, Rogier B Mars, MaryAnn P Noonan, Franz-Xaver Neubert, Saad Jbabdi, Jill X O'Reilly, Nicola Filippini, Adam G Thomas, and Matthew F Rushworth. The organization of dorsal frontal cortex in humans and macaques. *The Journal of Neuroscience*, 33(30):12255–12274, 2013.

[163] Franz-Xaver Neubert, Rogier B Mars, Adam G Thomas, Jerome Sallet, and Matthew FS Rushworth. Comparison of human ventral frontal cortex areas for cognitive control and language with areas in monkey frontal cortex. *Neuron*, 81(3):700–713, 2014.

[164] Franz-Xaver Neubert, Rogier B Mars, Jérôme Sallet, and Matthew FS Rushworth. Connectivity reveals relationship of brain areas for reward-guided learning and decision making in human and monkey frontal cortex. *Proceedings of the National Academy of Sciences*, 112(20):E2695–E2704, 2015.

[165] Alfred Anwander, Marc Tittgemeyer, D Yves von Cramon, Angela D Friederici, and Thomas R Knösche. Connectivity-based parcellation of broca's area. *Cerebral Cortex*, 17(4):816–825, 2007.

[166] David Moreno-Dominguez, Alfred Anwander, and Thomas R Knösche. A hierarchical method for whole-brain connectivity-based parcellation. *Human brain mapping*, 35(10):5000–5025, 2014.

[167] Stella X Yu and Jianbo Shi. Multiclass spectral clustering. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 313–319. IEEE, 2003.

[168] http://www.mathworks.com/help/stats/kmeans.html.

[169] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.

[170] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[171] Madhura Ingalhalikar, Alex Smith, Drew Parker, Theodore D Satterthwaite, Mark A Elliott, Kosha Ruparel, Hakon Hakonarson, Raquel E Gur, Ruben C Gur, and Ragini Verma. Sex differences in the structural connectome of the human brain. *Proceedings of the National Academy of Sciences*, 111(2):823–828, 2014.

[172] Gaolang Gong, Pedro Rosa-Neto, Felix Carbonell, Zhang J Chen, Yong He, and Alan C Evans. Age-and gender-related differences in the cortical anatomical network. *The Journal of neuroscience*, 29(50):15684–15693, 2009.

[173] Q. Wang, R. Chen, J. JaJa, Y Jin, L. Hong, and E. Herzkovits, "Connectivity-Based Brain Parcellation: A Connectivity-Based Atlas for Schizophrenia Research", submitted to *Neuroinformatics*.

[174] Edward H Herskovits, L Elliot Hong, Peter Kochunov, Hemalatha Sampath, and Rong Chen. Edge-centered dti connectivity analysis: Application to schizophrenia. *Neuroinformatics*, pages 1–9, 2015.

[175] Hermann Weyl. Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). *Mathematische Annalen*, 71(4):441–479, 1912.

[176] Henry Wolkowicz and George PH Styan. Bounds for eigenvalues using traces. *Linear algebra and its applications*, 29:471–506, 1980.