Graph-based Algorithms in Computer Vision, Machine Learning, and Signal Processing

Jhony H. Giraldo

LA ROCHELLE UNIVERSITÉ

ÉCOLE DOCTORALE EUCLIDE

Laboratoire Mathématiques, Image et Applications

Ph.D Thesis

Jhony Heriberto Giraldo Zuluaga

Supervisor: **Thierry Bouwmans**

Discipline : **Applied Mathematics**

Graph-based Algorithms in Computer Vision, Machine Learning, and Signal Processing

---

Ph.D. Committee:

Laure Tougne, Full Professor, Université Lumière Lyon 2.

Marc Van Droogenbroeck, Full Professor, Université de Liège.

Lucia Maddalena, Director of Research, ICAR-CNR INdAM Research Unit.

Jenny Benois-Pineau, Full Professor, Université Bordeaux.

François Bremond, Director of Research, INRIA Sophia-Antipolis.

Antonino Staiano, Associate Professor, Università degli Studi di Napoli Parthenope.

Laura Toni, Associate Professor, University College London.

Date of Ph.D. Defense: 1st September 2022.

"Imagination will often carry us to worlds that never were, but without it we go nowhere."

Carl Sagan

Summary

In this thesis, entitled "*Graph-based Algorithms in Computer Vision, Machine Learning, and Signal Processing*", we propose novel approaches in video and image processing, graph neural networks, and recovery of time-varying graph signals. As a result, this document is divided into three main parts: 1) computer vision, 2) machine learning, and 3) signal processing. Our main motivation is to use the geometrical information that we can capture from the data to avoid data hungry methods, *i.e.*, learning with minimal supervision. All our contributions rely heavily on the developments of Graph Signal Processing (GSP) and spectral graph theory. In particular, the sampling and reconstruction theory of graph signals play a central role in this thesis. The main contributions of this thesis are as follows:

1. We propose Graph Moving Object Segmentation (GraphMOS) [1–4] in Chapter 3. GraphMOS is composed of segmentation, background initialization, graph construction, unseen sampling, and a semi-supervised learning method inspired by the theory of recovery of graph signals for Moving Object Segmentation (MOS). GraphMOS has the advantage of requiring less labeled data than deep learning methods while having competitive results on both static and moving camera videos. GraphMOS is evaluated on six publicly available datasets outperforming several state-of-the-art methods in challenging conditions at the time of publication.
2. We pose MOS as a node classification problem using Graph Convolutional Networks (GCNs) [5] in Chapter 4. Our algorithm, dubbed as GraphMOS-Net, presents several advantages regarding GraphMOS while keeping its good properties. In particular, GraphMOS-Net requires little labeled information to per-

form well as GraphMOS [1]. However, we train a neural network model in [5], which makes testing faster than in GraphMOS. Similarly, adding new videos does not require solving again the optimization problem as in [1].

3. We propose HyperGraph Convolutional Networks for Weakly-supervised Semantic Segmentation (HyperGCN-WSS) [6] in Chapter 5. HyperGCN-WSS follows the same line of research as in the previous algorithms [1, 5] about learning with minimal supervision. However, HyperGCN-WSS addresses the problem of Weakly-supervised Semantic Segmentation (WSS), using a similar structure as in GraphMOS and GraphMOS-Net. HyperGCN-WSS constructs hypergraphs from the images in the dataset. Then, we train a specialized HyperGraph Convolutional Network (HyperGCN) architecture using some weak signals. HyperGCN-WSS is evaluated on the PASCAL VOC 2012 dataset for semantic segmentation. HyperGCN-WSS shows competitive performance against some state-of-the-art methods at the time of publication.
4. We propose a new scalable convolutional operator for Graph Neural Networks (GNNs) based on the Sobolev norm of graph signals [7] in Chapter 6. To this end, we use Hadamard products between matrices to keep the same sparsity level in graph representations, and therefore we dub our proposed algorithm as Sparse Sobolev GCN (S-SobGNN). Our architecture computes a cascade of filters on each layer with increasing Hadamard powers to get a more diverse set of functions, and then an attention layer selects the best operations. We provide several mathematical notions of our filtering operation based on the spectral graph theory and the Schur product theorem. S-SobGNN is evaluated in several tasks including, semi-supervised learning, and graph node classification. S-SobGNN shows competitive performance in all applications as compared to several state-of-the-art methods at the time of publication.
5. We introduce a topological relationship between over-smoothing and over-squashing in Chapter 7. We use concepts of spectral graph theory and differential geometry to show that both problems are intrinsically related to the spectral gap of the Laplacian representation of the graph. As a result, there is a trade-off between

these two problems from a topological perspective, *i.e.*, we cannot simultaneously alleviate both over-smoothing and over-squashing. Similarly, we use one bound of the Ollivier's Ricci curvature to propose a new rewiring algorithm for GNNs. Our proposed algorithm is less computationally expensive than previous curvature-based rewiring methods in the literature, while keeping fundamental theoretical properties. We also perform a thorough comparison of our algorithm with previous methods to alleviate over-smoothing and over-squashing, seeking to gain a better understanding of both problems and their practical solutions.

6. We introduce a novel algorithm based on the extension of a Sobolev smoothness function for the reconstruction of time-varying graph signals from discrete samples [8, 9] in Chapter 8. We explore some theoretical aspects of the convergence rate of our Time-varying Graph signal Reconstruction via Sobolev Smoothness (GraphTRSS) algorithm by studying the condition number of the Hessian associated with our optimization problem. Our algorithm has the advantage of converging faster than other methods that are based on Laplacian operators without requiring expensive eigenvalue decomposition or matrix inversions. GraphTRSS is evaluated on several datasets including two COVID-19 datasets and it has outperformed many existing state-of-the-art methods for time-varying graph signal reconstruction at the time of publication. GraphTRSS has also shown excellent performance on two environmental datasets for the recovery of particulate matter and sea surface temperature signals.

Figure 0-1 shows a graphical summary of the main contributions of this thesis.

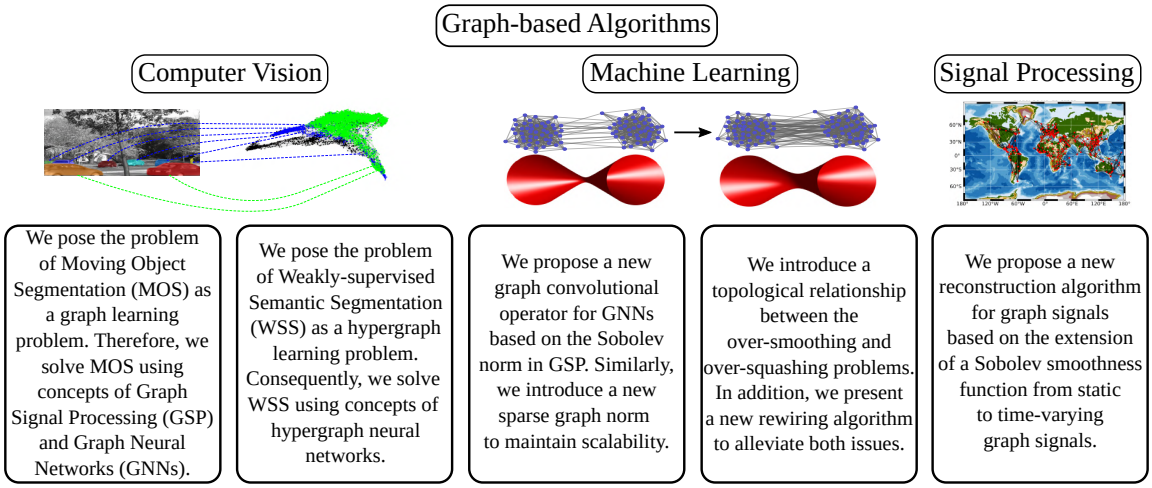

Figure 0-1: Main contributions presented in this thesis.

Acknowledgments

First and foremost, I would like to thank **Thierry Bouwmans** for giving me the opportunity to pursue a Ph.D under his supervision. I am tremendously grateful for his continuous support, and specially for giving me the freedom to pursue a variety of research directions in the past three years. Many thanks also go to **Fragkiskos Malliaros**, **Sajid Javed**, and **Arif Mahmood** for playing the role of co-supervisor at different stages of my Ph.D. **Fragkiskos**, I very much enjoyed my five months at CentraleSupélec with you. I would like to thank also my lab **Mathématiques, Image et Applications (MIA)**, the **Centre de Vision Numérique (CVN)**, the **OPIS team from Inria**, and the **CVPR Lab "Alfredo Petrosino"** for the support provided in my Ph.D.

I also would like to thank **Lucia Maddalena** and **Marc Van Droogenbroeck** for agreeing to review this Ph.D. thesis. Similarly, I would like to thank the rest of my Ph.D. committee for your valuable time: **Thierry Bouwmans**, **Laure Tougne**, **Jenny Benois-Pineau**, **François Bremond**, **Antonino Staiano**, **Laura Toni**, and **Fragkiskos D. Malliaros**.

I had the chance to work with several talented people closely during my Ph.D, including: **Dorina Thanou**, **Antonino Staiano**, **Naoufel Werghi**, **Ananda S. Chowdhury**, **Maryam Sultana**, **Vincenzo Scarrica**, **Anindya Mondal**, **Wieke Prummel**, **Marwa Chendeb El Rai**, and **Shashant R**. It was a pleasure and I learned a lot from working with you.

I would like to thank my friends from **High School**, and my best friends **Luis David Goyes**, **Nicanor García**, and **Alexander Gomez Villa** for their emotional support along this stage of my life. Thanks also go to **Viviana Beltrán** for helping

me to make the decision to do my Ph.D at La Rochelle, and for being our friend when my wife and I did not know anyone in France.

My past and present lab mates in the US and France, thank you so much for all of your friendship and help throughout this process. I will miss the great moments we lived together: **Belmar Garcia-Garcia**, **Wieke Prummel**, **Ségolène Martin**, **Kavya Gupta**, **Claire Rossignol**, **Thomas Guilmeau**, and **Sagar Verma**. From the US side many thanks go to: **Karelia Peña-Peña**, **Carlos Mendoza**, **Jhon Alejandro Castro**, **Christian Escobar**, **Carlos Restrepo**, **Angela Cuadros**, **Mariano Burich**, and **Claudio César Claros**, you were the hardest part of leaving the US. **Carlos M.** and **Angela**, you helped me out taking one of the most important and hardest decisions in my life, thank you very much.

I also thank **Pierre Rideau**, **Alain and Dominique Breysse**. You were our French parents when my wife and I arrived in France without speaking the language. You taught us French and helped us to navigate the French bureaucracy like if we were your sons. Many thanks also go to **Jean-Christophe and Sophie Pauget**, you welcomed us in your house like if we were family. I am immensely grateful to you, you all are like my French family, the success of this journey and the love I have now for French culture is in part thanks to you.

I would like to thank my family: **Gloria Zuluaga**, **Heriberto Giraldo**, **Diana Giraldo**, **Leonor Suarez**, **Omaira Zuluaga**, and **Nidia Botello** for supporting me spiritually throughout writing this research work and my life in general. Finally, I would like to thank my beloved wife **Camila Botello**. We have walked a long path together, from Colombia, to the US, and France. You are the person who holds me together, and the one who gives balance to my life.

The research in Chapter 7 was supported by **DATAIA** convergence institute as part of the "Programme d'Investissement d'Avenir", (ANR-17-CONV-0003) operated by CentraleSupélec.

List of Publications

This thesis is based on the following publications:

Journal Publications:

- **Jhony H. Giraldo**, Sajid Javed, Thierry Bouwmans, "Graph Moving Object Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, issue 5, pp. 2485-2503, 2022.
- **Jhony H. Giraldo**, Arif Mahmood, Belmar Garcia-Garcia, Dorina Thanou, Thierry Bouwmans, "Reconstruction of Time-varying Graph Signals via Sobolev Smoothness," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 8, pp. 201-214, 2022.
- **Jhony H. Giraldo**, Sajid Javed, Arif Mahmood, Fragkiskos D. Malliaros, Thierry Bouwmans, "Sparse Sobolev Graph Neural Networks," *IEEE Transactions on Knowledge and Data Engineering.* (To be submitted).

Conference Publications:

- **Jhony H. Giraldo**, Thierry Bouwmans, "Semi-supervised Background Subtraction of Unseen Videos: Minimization of the Total Variation of Graph Signals," *IEEE International Conference on Image Processing*, 2020.
- **Jhony H. Giraldo**, Thierry Bouwmans, "On the Minimization of Sobolev Norms of Time-Varying Graph Signals: Estimation of New Coronavirus Disease 2019 Cases," *IEEE International Workshop on Machine Learning for Signal Processing*, 2020.
- **Jhony H. Giraldo**, Thierry Bouwmans, "GraphBGS: Background Subtraction via Recovery of Graph Signals," *International Conference on Pattern Recognition*, 2021.

- **Jhony H. Giraldo**, Sajid Javed, Maryam Sultana, Soon Ki Jung, Thierry Bouwmans, "The Emerging Field of Graph Signal Processing for Moving Object Segmentation," *International Workshop on Frontiers of Computer Vision*, 2021.
- **Jhony H. Giraldo**, Sajid Javed, Naoufel Werghi, Thierry Bouwmans, "Graph CNN for Moving Object Detection in Complex Environments from Unseen Videos," *IEEE/CVF International Conference on Computer Vision - Workshops*, 2021.
- **Jhony H. Giraldo**, Vincenzo Scarrica, Antonino Staiano, Francesco Camastra, Thierry Bouwmans, "Hypergraph Convolutional Networks for Weakly-Supervised Semantic Segmentation," *IEEE International Conference on Image Processing*, 2022.
- **Jhony H. Giraldo**, Fragkiskos D. Malliaros, Thierry Bouwmans, *"On the Trade-off between Over-smoothing and Over-squashing in GNNs"*. (To be submitted).

Book Chapters:

- **Jhony H. Giraldo**, Huu Ton Le, Thierry Bouwmans, "Deep Learning Based Background Subtraction: A Systematic Survey", *6th Handbook of Pattern Recognition and Computer Vision, World Scientific Publishing*, pp. 51-73, 2020.
- **Jhony H. Giraldo**, Thierry Bouwmans, "Moving Objects Detection in Video Processing: A Graph Signal Processing Approach for Background Subtraction," *Artificial Intelligence Technologies, Applications, and Challenges, CRC Press, Taylor, Francis Group*, pp. 171-181, 2021.

Ideas, text, figures, and experiments originate mostly from the first author. Some authors had important advisory roles, while others helped with specific coding parts.

The author has further collaborated in other publications during the time of his Ph.D. However, these publications are not included in this thesis:

- Anindya Mondal, Shashant R., **Jhony H. Giraldo**, Thierry Bouwmans, Ananda S. Chowdhury, "Moving Object Detection for Event-based Vision using Graph Spectral Clustering," *IEEE/CVF International Conference on Computer Vision - Workshops*, 2021.

- Maryam Sultana, Thierry Bouwmans, **Jhony H. Giraldo**, Soon Ki Jung, "Robust Foreground Segmentation in RGBD Data from Complex Scenes using Adversarial Networks," *International Workshop on Frontiers of Computer Vision*, 2021.
- Marwa Chendeb El Rai, **Jhony H. Giraldo**, Mina Al-Saad, Muna Darweech, Thierry Bouwmans, "SemiSegSAR: A Semi-supervised Segmentation Algorithm for Ship SAR Images," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022.

Contents

List of Figures

List of Tables

Dedicated to my beloved family: Camila, Gloria, Heriberto, and Diana.

Chapter 1

Introduction

1.1 Graphs

A graph is a mathematical entity that can represent relationships between discrete objects. These objects are mathematical abstractions called nodes or vertices, where the relationships between them are captured with the so-called edges or links. A graph is thus given by a set of nodes and a set of edges. Graphs can be undirected like in friendship: a person A is friend of a person B, and vice versa. Graphs can also be directed like in a virtual social network of followers: a person A can follow a person B, but it could not be the case in the other sense. Similarly, we can have unweighted graphs where we have a connection or not, like in a social network (we do not say a person A is 50% friend of a person B). Graphs can also be weighted, where we represent how strong is the connection between nodes, like in *Google Maps* where we know how far is some place A to another place B. Unless explicitly stated otherwise, we consider undirected, connected, and weighted graphs in this thesis.

The fundamentals and applications of graphs have gained significant attention in recent years. Notably, Graph Neural Networks (GNNs) and Graph Signal Processing (GSP) have been extensively studied [28–35]. GNNs extend the concepts of Convolutional Neural Networks (CNNs) [36] to non-Euclidean data modeled as graphs. Similarly, GSP extends the concepts of classical digital signal processing to signals supported on graphs [28]. GNNs and GSP have numerous applications such as semi-

supervised learning [33, 34], point cloud semantic segmentation [37, 38], prediction of individual relations in social networks [39], and modeling of proteins for drug discovery [40, 41]. Similarly, other graph techniques have been recently applied to image, video, and medical image processing applications [1, 42–44].

Even though graph-based algorithms have had a lot of success in recent years, there are still several open questions. These topics range from fundamental aspects of the algorithms, to specific issues related to the application domains. In this thesis, we use graphs to model the complex relationships that exist between instances in some data. Thus, we exploit this intrinsic information to solve problems in: 1) learning with minimal supervision in computer vision and machine learning, and 2) reconstruction of time-varying signals that live on graphs.

1.2 Scope and Research Questions

This thesis is divided into three parts. Part 1 introduces novel concepts of graphs in some problems of computer vision like moving objects and semantic segmentation. Part 2 addresses problems of machine learning on graphs. In particular, we introduce several contributions in GNNs using concepts of GSP, where we study concepts like homophily [45], graph convolutions [33], over-smoothing [46], and over-squashing [47]. Finally, part 3 presents a new algorithm for the reconstruction of time-varying graph signals. This algorithm converges faster than other methods in the literature without requiring expensive eigenvalue decompositions or matrix inversions. This signal processing approach is evaluated for the reconstruction of COVID-19 and environmental data.

The contributions of this thesis have been driven by the following research questions:

1. *Can concepts of GSP be used in the problem of Moving Object Segmentation (MOS) under the paradigm of learning with minimal supervision?* We introduce Graph Moving Object Segmentation (GraphMOS) in Chapter 3. GraphMOS uses concepts of sampling and reconstruction in GSP to address MOS.

GraphMOS takes a minimal supervision approach leading to a new family of semi-supervised learning algorithms in MOS. GraphMOS is inspired in the reconstruction of graph signals to solve the semi-supervised learning problem. Therefore, we have to solve an optimization problem each time we want to evaluate our algorithm. GraphMOS is evaluated on several datasets for MOS.

2. *How can we overcome the limitations of GraphMOS using concepts of GNNs?* GraphMOS requires to solve an optimization problem if new instances are added to the dataset, which limits its practical applications. To address this problem, we propose to solve MOS with GCNs (GraphMOS-Net) in Chapter 4. GraphMOS-Net performs better than GraphMOS in several challenging conditions. Similarly, GraphMOS-Net is faster than GraphMOS in evaluation.
3. *Can we address semantic segmentation using the concepts developed in previous chapters?* We introduce HyperGraph Convolutional Networks for Weakly-supervised Semantic Segmentation (HyperGCN-WSS) in Chapter 5. Our algorithm uses hypergraphs to capture spatial information in the images, as well as structural information in the dataset. HyperGCN-WSS solves the problem of semantic segmentation with minimal supervision using weak signals like scribbles and clicks. HyperGCN-WSS is evaluated on the PASCAL VOC 2012 dataset.
4. *How can we improve the expressiveness of GNNs while keeping scalability?* We introduce Sparse Sobolev Graph Neural Networks (S-SobGNNs) in Chapter 6. Our algorithm uses a sparse Sobolev norm using concepts of GSP. Similarly, we add a linear combination mechanism to make graph convolutions more expressive, while keeping scalability. S-SobGNN is evaluated in several tasks including, semi-supervised learning, graph node classification, and MOS.
5. *What is the relationship between over-smoothing and over-squashing in GNNs?* We introduce a topological relationship between over-smoothing and over-squashing in Chapter 7. This underlying relationship is linked using the spectral gap of the Laplacian representation of the graph. We also introduce a new Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm to alleviate over-smoothing and over-squashing. SJLR is evaluated in several benchmarking datasets for

node classification.

6. *How can we increase the convergence rate of GSP algorithms for reconstruction?* We introduce Time-varying Graph signals Reconstruction via Sobolev Smoothness (GraphTRSS) in Chapter 8. GraphTRSS converges faster than previous methods for reconstruction of time-varying graph signals, without requiring expensive eigenvalue decompositions or matrix inversions. GraphTRSS is evaluated on several datasets including two COVID-19 datasets and two environmental datasets, outperforming many existing methods for time-varying graph signal reconstruction.

In addition to the contributions listed above, we include in Chapter 2 the mathematical notation and background of this thesis. After presenting the main body of this thesis in Parts 1, 2, and 3, we conclude and show potential directions for future work in Chapter 9.

Chapter 2

Background

In this chapter, we provide an introduction to the mathematical notation and some background topics that will be used in this thesis. More specifically, Section 2.1 presents the mathematical notation in this thesis. Section 2.2 introduces the basic concepts and definitions of graphs and graph signals. Section 2.3 explains the fundamental concepts of sampling and reconstruction of graph signals, and Section 2.4 explains the concept of smooth graph signals. Finally, Section 2.5 presents some concluding remarks of this chapter. Later chapters in this thesis introduce additional background concepts whenever required.

2.1 Mathematical Notation

In this thesis, calligraphic letters such as $\mathcal{V}$ denote sets and $|\mathcal{V}|$ represents its cardinality. Uppercase boldface letters such as $\mathbf{A}$ represent matrices, and lowercase boldface letters such as $\mathbf{x}$ denote vectors. $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is a vector of ones with appropriate dimensions. $(\cdot)^{\mathsf{T}}$ and $(\cdot)^{\mathsf{H}}$ represent transposition and Hermitian transpose. The vectorization of matrix $\mathbf{A}$ is denoted as $\mathrm{vec}\,(\mathbf{A})$, and $\mathrm{diag}(\mathbf{x})$ is the diagonal matrix with entries $\{x_1, x_2, \ldots, x_N\}$ as its diagonal elements. $\|\cdot\|_2$ is the ℓ_2-norm of a vector, $\|\cdot\|_F$ is the Frobenius norm, and $\|\cdot\|_{1,1}$ is the entry-wise norm-1 of a matrix. $\lambda_{\max}(\mathbf{A})$ and $\lambda_{\min}(\mathbf{A})$ represent the maximum and minimum eigenvalues of matrix $\mathbf{A}$, respectively. $\kappa(\cdot)$ is the condition number of a matrix, and $\triangleq$ is

the symbol for "equal by definition". $\nabla_{\mathbf{x}} f(\mathbf{x})$ is the gradient of certain vector-valued function $f(\mathbf{x})$, and $\nabla^2_{\mathbf{x}} f(\mathbf{x})$ is its Hessian. Finally, $\circ$ and $\otimes$ represent the Hadamard and Kronecker product between matrices, respectively.

2.2 Graph Signals

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected and weighted graph. $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes and $\mathcal{E} = \{(i, j)\}$ is the set of edges, where (i, j) is an edge between the vertices i and j. $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix of G such that $\mathbf{A}(i, j) = a_{ij} \in \mathbb{R}^+$ is the weight connecting vertices i and j. As a consequence, $\mathbf{A}$ is symmetric for undirected graphs. A graph signal is a function $y : \mathcal{V} \to \mathbb{R}$ defined on the nodes of G, and it can be represented as $\mathbf{y} \in \mathbb{R}^N$ where $\mathbf{y}(i)$ is the function evaluated on the ith node. Moreover, $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix of G such that $\mathbf{D}(i, i) = \sum_{j=1}^{N} \mathbf{A}(i, j)\ \forall\ i = 1, \ldots, N$, and $d_i = \mathbf{D}(i, i)$. There are several definitions in the literature for the Laplacian matrix. For example, $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the combinatorial Laplacian matrix, and $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the symmetric normalized Laplacian [28]. The Laplacian matrix is a positive semi-definite matrix for undirected graphs with eigenvalues[1] $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$ and corresponding eigenvectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$.

The graph Fourier basis of G is defined by the spectral decomposition of $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathsf{T}$ [28], where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]$ and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$, where λ_i is the frequency associated to the ith eigenvalue [28]. Therefore, the Graph Fourier Transform (GFT) $\hat{\mathbf{y}}$ of the signal $\mathbf{y}$ is defined as $\hat{\mathbf{y}} = \mathbf{U}^\mathsf{T}\mathbf{y}$, and the inverse GFT is given by $\mathbf{y} = \mathbf{U}\hat{\mathbf{y}}$ [28].

Definition 2.2.1. *A graph signal* $\mathbf{y}$ *is called bandlimited if* $\exists\ \rho \in \{1, 2, \ldots, N-1\}$ *such that its GFT satisfies* $\hat{\mathbf{y}}(i) = 0\ \forall\ i > \rho$.

The smallest ρ that holds Definition 2.2.1 is called the bandwidth of $\mathbf{y}$. Using these notions of frequency, Pesenson [48] defined the space of all ω-*bandlimited* signals as $PW_\omega(G) = \text{span}(\mathbf{U}_\rho : \lambda_\rho \leq \omega)$, where $\mathbf{U}_\rho$ represents the first ρ eigenvectors of $\mathbf{L}$, and

[1] $\lambda_N \leq 2$ in the case of the symmetric normalized Laplacian $\mathbf{\Delta}$.

$PW_\omega(G)$ is known as the Paley-Wiener space of G. As a consequence, a graph signal $\mathbf{y}$ has cutoff frequency ω, and bandwidth ρ if $\mathbf{y} \in PW_\omega(G)$.

2.3 Sampling and Reconstruction of Graph Signals

Given the definition of graph signals and the notions of bandlimitedness in terms of $PW_\omega(G)$, the next logical step is to find a bound for the minimum number of samples required to reach perfect recovery of $\mathbf{y} \in PW_\omega(G)$. The answer is that one needs at least ρ (bandwidth) labeled nodes to achieve perfect reconstruction. Intuitively, a graph signal $\mathbf{y}$ is smooth in G when $\mathbf{y} \in PW_\omega(G)$. For example, suppose a sensor network of temperatures in a specific region. One would expect that the temperature of two or more nearby localities should be similar, *i.e.*, the value of the graph signal evaluated in two or more strongly connected nodes should not be very different. As a consequence, probably one just needs the temperature of some of these nodes to reconstruct the whole graph signal in the other vertices.

To add mathematical precision to the notion of perfect reconstruction, the sampling of a graph signal is defined in terms of a subset of nodes $\mathcal{S} \subset \mathcal{V}$ with $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, where $m = |\mathcal{S}| \leq N$ is the number of sampled nodes. The sampled graph signal is defined as $\mathbf{y}(\mathcal{S}) = \mathbf{M}\mathbf{y}$, where $\mathbf{M}$ is a binary decimation matrix whose entries are given by $\mathbf{M} = [\boldsymbol{\delta}_{s_1}, \ldots, \boldsymbol{\delta}_{s_m}]^\mathsf{T}$ and $\boldsymbol{\delta}_v$ is the $N-$dimensional Kronecker column-vector centered at v. The recovery of a graph signal from its samples $\mathbf{y}(\mathcal{S})$ can be represented as $\tilde{\mathbf{y}} = \mathbf{\Phi}\mathbf{M}\mathbf{y}$, where $\mathbf{\Phi} \in \mathbb{R}^{N \times m}$ is an interpolation matrix. Perfect recovery is achievable if $\mathbf{\Phi}\mathbf{M} = \mathbf{I}$, *i.e.*, $\tilde{\mathbf{y}} = \mathbf{I}\mathbf{y} = \mathbf{y}$. Since $\text{rank}(\mathbf{\Phi}\mathbf{M}) \leq m \leq N$, perfect reconstruction is not possible in general. However, perfect reconstruction from a sampled graph signal $\mathbf{y}(\mathcal{S})$ is possible when the sampling size $|\mathcal{S}| \geq \rho$ [49].

Theorem 2.3.1 (Chen's theorem [49])**.** *Let* $\mathbf{M}$ *satisfy* $\text{rank}(\mathbf{M}\mathbf{U}_\rho) = \rho$*. For all* $\mathbf{y} \in PW_\omega(G)$*, perfect recovery, i.e.,* $\mathbf{y} = \mathbf{\Phi}\mathbf{M}\mathbf{y}$*, is achieved by choosing:*

$$\mathbf{\Phi} = \mathbf{U}_\rho \mathbf{V}, \tag{2.1}$$

with $\mathbf{VMU}_\rho$ *a* $\rho \times \rho$ *identity matrix.*

Proof: see [49].

Theorem 2.3.1 states that perfect reconstruction of graph signal from its samples is possible when $\mathbf{y}$ lies in $PW_\omega(G)$, and the number of samples is at least ρ. Then, perfect reconstruction is achieved by choosing the interpolation operator as in (2.1). A common approach to obtain a reconstructed version of $\mathbf{y}$ is given by:

$$\underset{\mathbf{z}\in\text{span}(\mathbf{U}_\rho)}{\arg\min} \; ||\mathbf{Mz} - \mathbf{y}(\mathcal{S})||_2^2 = \mathbf{U}_\rho(\mathbf{MU}_\rho)^\dagger \mathbf{y}(\mathcal{S}), \tag{2.2}$$

where $\mathbf{U}_\rho = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_\rho]$ is the matrix formed of the first ρ graph's eigenvectors, and $(\mathbf{MU}_\rho)^\dagger$ is the pseudo-inverse of $(\mathbf{MU}_\rho)$. In other words, the interpolation operator is such that $\mathbf{\Phi} = \mathbf{U}_\rho(\mathbf{MU}_\rho)^\dagger$. The computation of the Laplacian eigenvectors in (2.1) and (2.2) is computationally prohibitive for large graphs (as the ones treated in this work). In this thesis, the computation of $\mathbf{U}$ is avoided, and we resort to concepts of smooth graph signals to solve the problem of reconstruction of graph signals.

2.4 Smooth Graph Signals

The notion of bandlimitedness in Definition 2.2.1 is related to the smoothness of $\mathbf{y}$. When $\mathbf{y} \in PW_\omega(G)$, the variation of $\mathbf{y}$ is smooth in the vertex domain. From Definition 2.2.1, one knows that the GFT is required to check if $\mathbf{y} \in PW_\omega(G)$. However, the computation of the GFT requires the calculation of the eigenbasis, which is computationally prohibitive for large graphs. In this thesis, the computation of $\mathbf{U}$ is avoided by leveraging notions of global smoothness in G.

Formally, notions of smoothness in $\mathbf{y}$ were introduced through concepts of *local variation* and the discrete p-Dirichlet form [29]. The local variation of $\mathbf{y}$ at vertex i is defined as:

$$\|\nabla_i \mathbf{y}\|_2 \triangleq \left[\sum_{j\in\mathcal{N}_i} \mathbf{A}(i,j)[\mathbf{y}(j) - \mathbf{y}(i)]^2\right]^{\frac{1}{2}}, \tag{2.3}$$

where $\mathcal{N}_i$ is the set of vertices connected to the node i by one edge. Thus, the discrete

p-Dirichlet form is defined as $S_p(\mathbf{y}) \triangleq \frac{1}{p}\sum_{i\in\mathcal{V}} \|\nabla_i \mathbf{y}\|_2^p$, then:

$$S_p(\mathbf{y}) = \frac{1}{p}\sum_{i\in\mathcal{V}} \left[\sum_{j\in\mathcal{N}_i} \mathbf{A}(i,j)[\mathbf{y}(j)-\mathbf{y}(i)]^2\right]^{\frac{p}{2}}. \tag{2.4}$$

For example, when $p = 2$:

$$S_2(\mathbf{y}) = \sum_{(i,j)\in\mathcal{E}} \mathbf{A}(i,j)[\mathbf{y}(j)-\mathbf{y}(i)]^2 = \mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y}. \tag{2.5}$$

$S_2(\mathbf{y})$ is known as the graph Laplacian quadratic form [29]. Notice that $S_2(\mathbf{y}) = 0 \iff \mathbf{y} = \tau\mathbf{1}$, where τ is a constant; and more generally, $S_2(\mathbf{y})$ is small when the graph signal $\mathbf{y}$ has similar values at neighboring nodes connected by an edge, *i.e.*, when the signal is smooth.

The Laplacian quadratic form in (2.5) has been used as regularizer in reconstruction of graph signals, and semi-supervised learning problems [50], where this regularizer looks for smooth graph signals. Intuitively, there is a relationship between the smoothness of a graph signal and its bandwidth. For example, the variational problem [51] leads to the same solution of Eqn. (2.2) when $\mathbf{y}$ holds Definition 2.2.1 [52] (for further details see Section 3.3.6). Therefore, the minimization of the p-Dirichlet form is aligned with the prior assumption of bandlimitedness, without the explicit computation of the eigenbasis $\mathbf{U}$ and the bandwidth ρ of $\mathbf{y}$. Formally, since $\mathbf{L}$ is positive semi-definite for undirected graphs, all the eigenvalues are non-negative and real, and a full set of orthogonal eigenvectors can be obtained as explained in Section 2.2. The matrix of eigenvectors $\mathbf{U}$ is known as the GFT matrix of the graph. The eigenvalue-eigenvector pairs can be viewed as successive optimizers of the Rayleigh quotient, where the ith pair $\lambda_i, \mathbf{u}_i$ solves:

$$\mathbf{u}_i = \underset{\mathbf{y}^\mathsf{T}\mathbf{u}_{i'}=0, i'=0,\ldots,i-1}{\arg\min} \frac{\mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y}}{\mathbf{y}^\mathsf{T}\mathbf{y}}, \tag{2.6}$$

with $\lambda_i = \mathbf{u}_i^\mathsf{T}\mathbf{L}\mathbf{u}_i$ if $\mathbf{u}_i^\mathsf{T}\mathbf{u}_i = 1$. The term $\mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y}$ is precisely the Laplacian quadratic form of the graph signal $\mathbf{y}$ [29], *i.e.*, the GFT provides an orthogonal basis with in-

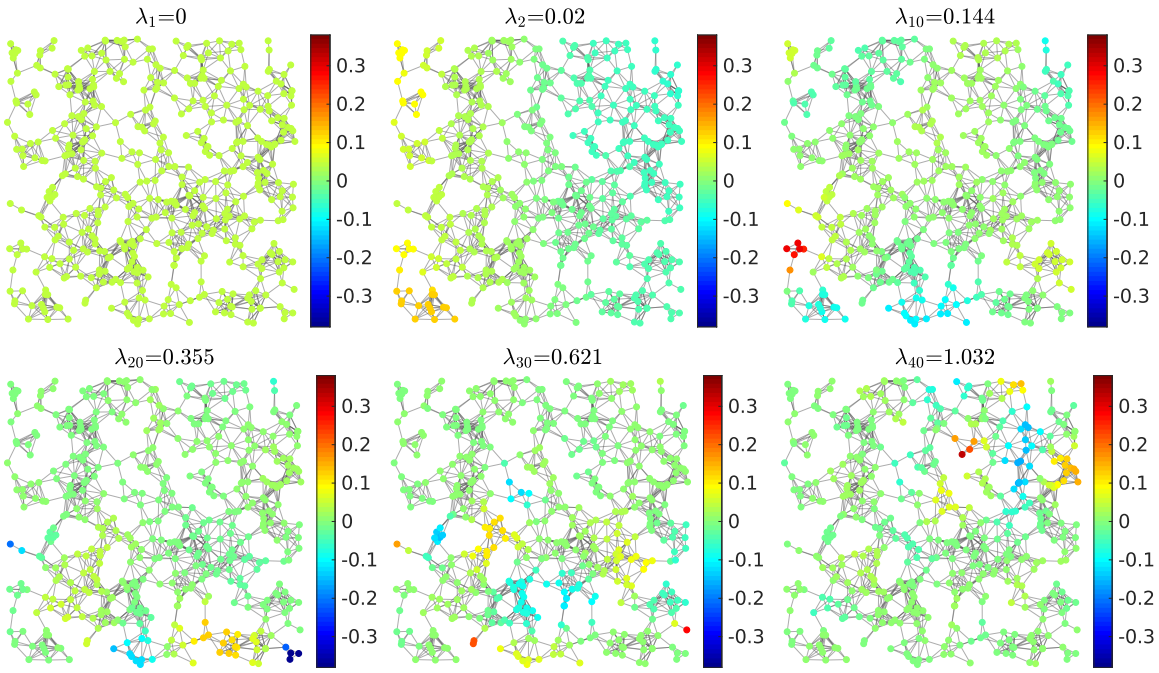

Figure 2-1: Example of elementary frequencies obtained from the Laplacian matrix on a sensor network of $N = 500$. Each graph shows a frequency λ_i with its corresponding eigenvector. The lowest frequency is $\lambda_1 = 0$, corresponding to a constant graph signal, *i.e.*, the Laplacian quadratic form of eigenvector $\mathbf{u}_1$ is given such that $\mathbf{u}_1^\mathsf{T}\mathbf{L}\mathbf{u}_1 = \lambda_1 = 0$.

creased variation [28]. Figure 2-1 shows an example of the eigenvectors of a weighted undirected sensor network with 500 nodes. One can notice that the eigenvector $\mathbf{u}_i$ has more variations as the value of λ_i increases, where the Laplacian quadratic form of $\mathbf{u}_1$, given as $\mathbf{u}_1^\mathsf{T}\mathbf{L}\mathbf{u}_1 = \lambda_1 = 0$, corresponds to a constant-valued eigenvector. Since the GFT of a graph signal is given such that $\hat{\mathbf{y}} = \mathbf{U}^\mathsf{T}\mathbf{y}$, one can precisely represent a bandlimited graph signal in the Fourier domain as $\hat{\mathbf{y}} = [\mathbf{u}_1, \ldots, \mathbf{u}_\rho, \mathbf{0}, \ldots, \mathbf{0}]^\mathsf{T}\mathbf{y}$ according to Definition 2.2.1, *i.e.*, a mapping of the first ρ eigenvectors. As a consequence, a bandlimited graph signal is smooth on undirected graphs.

In semi-supervised learning, the number of samples required to get perfect classification increases when the bandwidth ρ increases. In practice, graph signals are only approximately bandlimited [53]. As a consequence, the classification error is bounded by a value ϕ [52].

2.5 Conclusions

Graph representation learning and its applications have gained significant attention in recent years. GNNs and GSP have been extensively studied [28, 30, 33]. GNNs and GSP have numerous applications in machine learning [33] and computer vision [1, 38]. Despite the success of GNNs and GSP in many applications, there are still several open problems. For instance, GNNs suffer from the over-smoothing (and maybe the over-squashing) issue when several graph-convolutional layers are stacked to create deep models [46]. This problem limits the performance of deep GNN architectures. The following chapters of this thesis deal with some fundamental problems in graph representation learning and its applications in several fields of computer science.

Part I

Computer Vision

Chapter 3

Graph Moving Object Segmentation

3.1 Introduction

MOS is a relevant topic in computer vision and video analysis. MOS is a pre-processing task in many applications such as robotics system, intelligent transportation, and intelligent visual surveillance systems of human activities in public spaces, among others [54]. The main objective of MOS is to separate the moving objects called foreground, from the static component called background [55–58]. In the literature, MOS has been considered as a classification problem where each pixel is classified for either background or moving object in a sequence taken from a static or moving camera, and therefore this problem is also known as background subtraction and moving object detection [59]. Many efforts have been reported in the literature to improve the classical methods progressively in applications where challenges are becoming more complex. However, none of the methods can simultaneously address all the key challenges that are present in videos during long sequences as in the real cases [10]. In fact, several studies are focused on designing methods for specific challenges in MOS such as turbulence [60], dynamic backgrounds [61], and camouflaged moving objects [62]. Furthermore, many studies are limited to deal with shadows and the sequences taken from Pan-Tilt-Zoom (PTZ) cameras because of their challenging nature [54, 63–65].

MOS methods can broadly be categorized into unsupervised and supervised learn-

ing schemes. Many unsupervised learning methods have been proposed in the literature, and they rely on background models to predict the foreground objects [58,63,66]. However, these methods show performance degradation in the presence of dynamic background scenes. Supervised learning methods are based on CNNs [64]. CNN-based methods have demonstrated better performance than the unsupervised methods, however, most of these models fail to get optimal performance when employed on unseen videos (poor generalization). For example, the FgSegNet method uses 200 images from the test video for training and the remaining frames from the same video for evaluation [67]. The performance of FgSegNet dramatically decreases when applied to unseen videos [14]. Fig. 3-1 shows some visual results of the state-of-the-art methods for MOS under challenging background scenarios. Despite significant efforts and competitive performance in particular challenges, there are still several open issues for the MOS task [54, 64]. 1) None of the methods can effectively handle all MOS challenges in the presence of static and moving camera sequences. 2) Some CNNs-based models do not have a good performance on unseen videos, or their generalizations to other videos are hardly predictable [14]. 3) Deep learning methods lack theoretical guarantees and explanations about the sample complexity required during the end-to-end learning process. The classical fundamental theorem of statistical learning, involving the Vapnik–Chervonenkis dimension, bounds the sample complexity in machine learning [68]. However, this bound does not guarantee the performance in deep learning regimen because of the huge amount of parameters in common deep neural networks.

In recent years, a growing number of graph-based methods have been proposed for many computer vision and machine learning applications such as object tracking [43, 69], MOS [70–72], and tissue community detection [44, 73]. Graphs can model complex relationships on data [28]. These structures capture the intrinsic geometric structure in data and can model data points and complex interactions among them. Each node or vertex on the graph represents one data point to which a label can be associated, and a graph can be formed by connecting vertices with edge weights that are assigned based on distance values among the data points in the feature space.

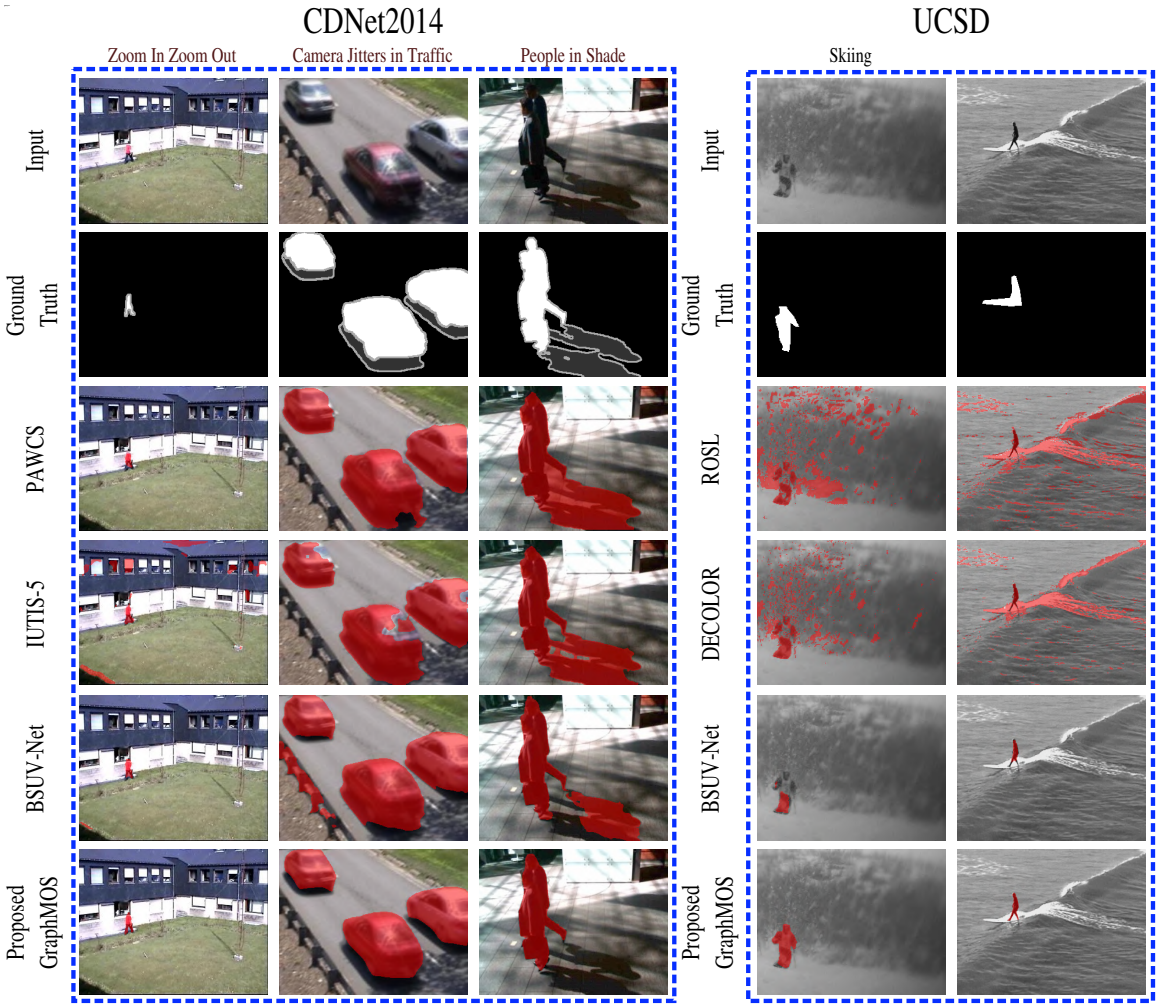

Figure 3-1: Comparisons of the visual results of the proposed Graph Moving Object Segmentation (GraphMOS) algorithm with existing state-of-the-art methods on five MOS challenging video sequences taken from CDNet2014 [10] and UCSD [11] datasets. The compared methods are: PAWCS [12], IUTIS-5 [13], BSUV-Net [14], ROSL [15], and DECOLOR [16]. Our proposed algorithm performs significantly better than the compared methods in these challenging sequences.

Social, financial, ecological networks, and the human brain are few examples of such data structure that can be modeled on graphs [28,29]. GSP enables different types of learning and filtering operations on values associated with graph nodes [49,53,74,75]. For inference, these graph models are used to classify graph signals. Therefore, semi-supervised learning can be modeled as the reconstruction of a graph signal from its samples [53]. When data labels are presented as signals on a graph, graph signal regularization techniques can be used in the process of estimating the unknown labels for graph nodes classification [53, 74].

In this chapter, we pose the problem of MOS as a semi-supervised learning problem on graphs. The nodes in the graph represent the homogeneous regions (known as superpixels) of the video sequence, and the task is to classify each homogeneous region to either a background (static component) or a moving object (foreground component) node by using the concepts of sampling and reconstruction of graph signals. Our algorithm thus lies in between the unsupervised and supervised techniques, leading to a new branch of MOS algorithms. Our proposed algorithm explores a somewhat radical departure from prior work in MOS, inspired by the theory of GSP [28,29]. We name the proposed semi-supervised learning algorithm as Graph Moving Object Segmentation (GraphMOS), where grouped regions in the videos are modeled as nodes of a graph embedded in a high dimensional space, and a graph signal is related to the class static or moving object. Our proposed GraphMOS algorithm is composed of superpixel segmentation, background initialization, feature extraction for nodes representation, construction of a graph, sampling of graph signals, and finally, a recovery method is applied to reconstruct the graph signal from its samples. The task of the reconstruction algorithm is to classify the graph nodes. Moreover, the bandwidth of the graph signal associated with the problem shows an upper bound for the sample complexity required in semi-supervised learning [52], assuming bandlimitedness and no noise in the graph signal. Several configurations of the proposed algorithm are evaluated for the MOS task outperforming many state-of-the-art methods on the Change Detection 2014 (CDNet2014) dataset [10], I2R [76], Scene Background Initialization (SBI2015) [65], and UCSD background subtraction [11] datasets.

The advantages of GraphMOS are: 1) its good performance even when the background scene rapidly changes, which is difficult to handle using existing MOS methods (Fig. 3-1), and 2) its theoretical foundation, unlike other state-of-the-art methods. The main contributions of the current work are summarized as follows:

- The MOS problem is posed as a graph learning problem by using the concepts of GSP. To the best of our knowledge, this is the first work that exploits the sampling and reconstruction of graph signals for MOS.
- One theoretical development is introduced, showing the upper-bound for the sample complexity required in semi-supervised learning under some prior assumptions in Corollary 3.3.1.
- Extensive evaluations are performed on four publicly available MOS benchmark datasets, and we compared our algorithms with 20 existing state-of-the-art methods with rigorous analysis. Unlike previous methods in the literature, our proposed algorithm can be applied to MOS with static and moving camera sequences.

3.2 Related Works

This section presents brief reviews for 1) GSP and its application to computer vision, and 2) unsupervised and supervised MOS algorithms.

3.2.1 Graph Signal Processing

Even though the study of graphs is an ancient field, Sandryhaila and Moura were the first authors to introduce the term of discrete signal processing on graphs and later coined with the name of GSP [77]. Graph signal processing emerged with the idea of developing tools to analyze data living in irregular and complex structures [28]. From one point of view, the first developments of GSP come from the studies of low-dimensional representations for high-dimensional data through spectral graph theory, and the graph Laplacian [78]. From another perspective, several authors developed compression schemes, wavelet decomposition, filter banks on graphs, regression al-

gorithms, and denoising using the graph Laplacian motivated by the data collected from sensor networks [79–83].

GSP has also been widely used in image processing and computer graphics. For example, Shi and Malik represented images as graphs to treat segmentation as a graph partitioning problem [84]. In the same way, image filtering techniques can be interpreted from a graph point of view [85]. Similarly in computer graphics, models like meshes can be naturally modeled with graphs to apply graph-based filtering and multi-resolution operations [86, 87].

In video processing, GSP is useful to model the spatiotemporal relationships among frames. For instance, the graph Cartesian product could be useful to process videos taking into account the spatiotemporal relationships of the pixels [88]. Interested readers can explore more details about GSP and its machine learning applications in a recent survey [28]. In our proposed algorithm, graphs are used to model the relationship among the nodes on videos where the graph signal represents the class foreground or background of the set of nodes in a dataset. Finally, the reconstruction of graph signals is applied to classify if a certain node is a moving or static object for MOS.

3.2.2 Moving Object Segmentation

There are many unsupervised methods in the literature to address the problem of MOS. These methods can be grouped as statistical [89], fuzzy [90], subspace learning [91], robust principal component analysis [92, 93], neural networks [94], and filtering-based [95] models. Interested readers may explore a complete review of unsupervised methods in the survey papers [58], [63], and [66]. With the success of deep CNNs on a wide variety of computer vision applications [36], several studies have also been proposed for MOS [64, 96, 97] applications.

The MOS supervised methods can be classified as basic CNN [96], multiscale CNN [98, 99], fully CNN [100], 3D CNN [97], and Generative Adversarial Networks (GANs) [101]. Some studies are also contributed to improving the loss functions during the training and analysis of the CNNs for the problem of MOS [102]. A

complete review of deep learning-based MOS methods can be explored in a recent survey [64].

Although the results of many fully supervised deep learning-based methods show impressive performance for MOS task, the performance of the best FgSegNet method proposed by Lim *et al.* [67] shows that CNNs methods are not efficient for unseen videos because of the lack of generalization capabilities as proved by Tezcan *et al.* [14]. This lack could be due to the limited amount of data available to train these deep learning methods for MOS, the lack of theoretical developments about the sample complexity required in deep learning, and the high model complexities of common deep neural networks. In this chapter, we fill this lack by proposing a semi-supervised graph learning algorithm for MOS in unseen videos. The question of sample complexity is also solved by using the fundamental concepts of GSP under the assumption of bandlimitedness of the underlying graph signals in MOS. It is also evident that a single method, either supervised or unsupervised, cannot effectively handle all the MOS challenges of unseen videos [14]. Our proposed algorithm also addresses these challenges for sequences taken from both static and moving cameras.

3.3 Moving Object Segmentation and Graph Signal Processing

This section presents the basic concepts of our proposed algorithm. Figure 3-2 shows an overview of GraphMOS. GraphMOS consists of several components including (a) superpixel segmentation, (b) background model initialization, (c) features extraction, (d) graph construction, (e) graph signal representation, (f) sampling of graph signals with an unseen scheme, and (g) a semi-supervised algorithm inspired from the theory of GSP.

The novices in GSP are referred to the review papers [28, 29]. GraphMOS can be viewed as three-step algorithm as follows: 1) a superpixel segmentation method [17, 103] is used to segment homogeneous regions on each frame; 2) deep and handcrafted

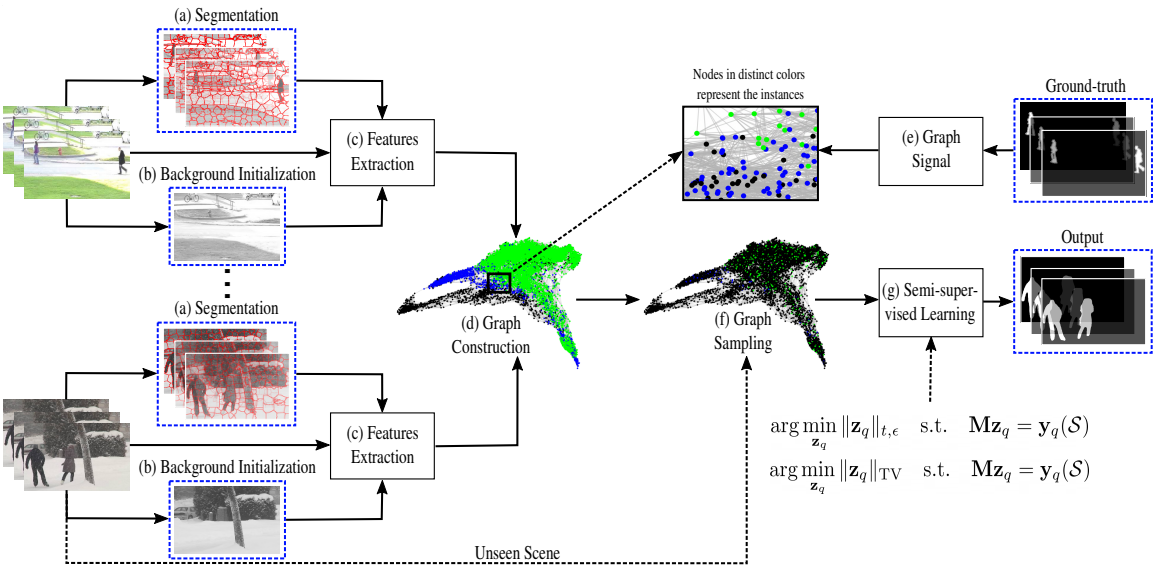

Figure 3-2: The pipeline of the MOS algorithm with the reconstruction of graph signals. The algorithm uses background initialization and superpixel segmentation [17, 18]. Each superpixel represents a node in a graph, and the representation of each node is obtained with motion, intensity, texture, and deep features. The ground-truth is used to decide if a node is a moving (green nodes) or a static object (blue nodes). Black nodes correspond to the non-labeled images in the dataset. Finally, some nodes are sampled and the semi-supervised algorithm reconstructs all the labels in the graph.

features representation from each superpixel, including optical flow and background initialization, are used to represent the spatiotemporal information of each superpixel node; and 3) a graph reasoning algorithm, including graph construction and semi-supervised learning, is used to classify between the static and moving objects with few labeled samples.

3.3.1 Graph Nodes Representation

In our proposed algorithm, there is a need for some mechanism to represent the graph nodes. The pixels of the video frames can represent the nodes of the graph. However, computational complexity issues arise within the proposed algorithm when using pixel-level nodes. Therefore, we use a group of pixels to define each node in a graph. To that end, one common way is to decompose the input sequence into a regular block structure or superpixels. However, other instance and semantic

segmentation methods can also be employed for node representation.

In this chapter, we employ several segmentation methods including, superpixel segmentation [17], semantic segmentation [19], instance segmentation [20], block-based decomposition, and background subtraction [104] to represent the nodes in a graph, as well as to compare the performance of each of the methods. However, the aforementioned segmentation methods present a fundamental limitation in the proposed algorithm. For instance, if the segmentation methods [20], [19], and [104] do not segment moving objects, the proposed algorithm will not be able to classify background or foreground objects effectively. On the other hand, superpixel segmentation and block-based decomposition methods can process all the regions in the frames, however, these approaches may contain more graph nodes than the segmentation methods. In the later subsections, we summarize each of these methods in more detail.

Superpixel Segmentation

In our proposed algorithm, we use a Simple Linear Iterative Clustering (SLIC) method for superpixel segmentation [17]. SLIC adapts a k-means approach to generate a set of superpixels. The desired number of approximately equally-sized superpixels per image ζ is an important parameter in SLIC for GraphMOS, because large values of ζ allow our algorithm to process more detailed regions, but may induce computational burdens on GraphMOS. Several ablation studies are performed in Section 3.5.3 to show the performance of the proposed algorithms by varying the number of superpixels.

Instance Segmentation

We also test several configurations of instance segmentation methods in GraphMOS. We employ Mask Region Convolutional Neural Network (Mask R-CNN) [20] and Cascade Mask R-CNN [105] with Residual Networks (ResNet) [106] and ResNeSt [107] for instance segmentation. Mask R-CNN builds upon Faster R-CNN by adding a branch for predicting an object mask in parallel with the already existing Faster R-CNN network for bounding box recognition [108]. Cascade Mask R-CNN builds upon

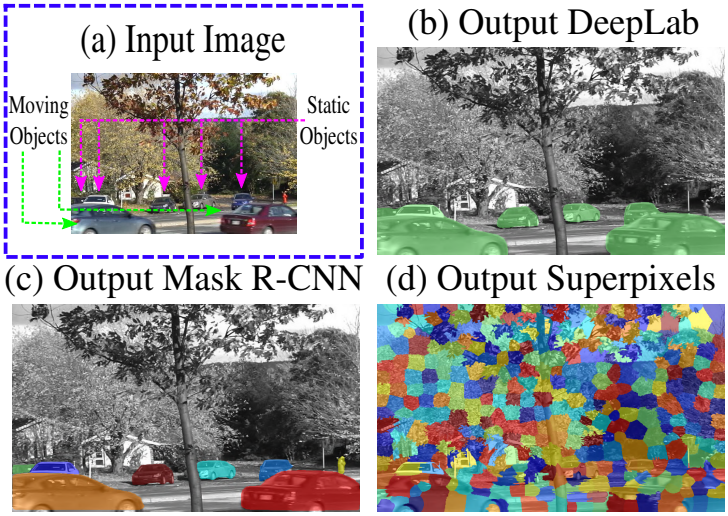

Figure 3-3: Results of the semantic, instance, and superpixel segmentation using DeepLab [19], Mask R-CNN [20], and SLIC [17] methods on the sequence *fall* taken from the CDNet2014 dataset. The green-colored cars in (b), instances in different colors in (c), and homogeneous regions in (d) represent the nodes of the graph.

Mask R-CNN by adding a sequence of detectors [105]. Mask R-CNN contains: 1) a CNN for image feature extraction, 2) a region proposal layer, 3) Region of Interest (ROI) alignment and 4) fully connected layers in parallel with convolutional layers to perform bounding box recognition and mask prediction, respectively. The readers are referred to Appendix A for further details about the instance segmentation methods.

Semantic Segmentation

We use the DeepLab method [19] for semantic segmentation to represent the nodes in the graph. However, our algorithm explicitly needs to know the exact location of each segmented instance in a frame to represent each node in a graph. Fig. 3-3 shows the segmentation results of the DeepLab, Mask R-CNN, and SLIC methods on a video frame selected from the dynamic backgrounds category of the CDNet2014 dataset. GraphMOS relies mainly on superpixel and instance segmentation because semantic segmentation methods do not give information about the location of each specific instance (for example, cars as shown in Fig. 3-3). As a consequence, GraphMOS is unable to differentiate between the parked cars in the background just behind the moving cars in the foreground when relying on semantic segmentation.

Other Segmentation Methods

We also decompose each video frame into non-overlapping blocks in GraphMOS. We use small blocks of size 8×8 to represent each node. We have also employed the background subtraction method SUBSENSE [104] to extract graph nodes. Several ablation studies are presented in Section 3.5.3 for comparing different segmentation methods.

3.3.2 Background Initialization and Feature Extraction

The MOS on unseen videos in static camera sequences can use the background of the scene as additional information. For the sake of simplicity, the temporal median filter is used as background initialization. The videos are processed in gray-scales in the current chapter.

The representation of the nodes is achieved with optical flow, intensity, texture, and deep features. The feature extraction module processes the ROI of the segmented regions[1] in the current frame for the current, previous, and background frames, as well as the absolute value of the difference between the current and background frames. Let $\mathbf{I}_v^t$ and $\mathbf{I}_v^{t-1}$ be the gray-scale crops corresponding to the node $v \in \mathcal{V}$ in the current (t) and previous frames $(t-1)$, respectively. Let $\mathbf{B}_v$ be the crop of the background image corresponding to the node v. Let $\mathcal{P}_v$ be the set of indices corresponding to the vth segmented region. Finally, let $\mathbf{v}_x^t(\mathcal{P}_v)$ and $\mathbf{v}_y^t(\mathcal{P}_v)$ be the optical flow vectors of the current frame with support in the set of indices $\mathcal{P}_v$ for the horizontal and vertical direction, respectively. We compute the optical flow by employing the Lucas-Kanade method [109]. Fig. 3-4 shows the procedure to extract the features of each segmented region when the segmentation algorithm is a Mask R-CNN. The texture representation is obtained by estimating the local binary patterns [110] in $\mathbf{I}_v^t$, $\mathbf{I}_v^{t-1}$, $\mathbf{B}_v$ and $|\mathbf{I}_v^t - \mathbf{B}_v|$. The intensity histograms are computed in $\mathbf{I}_v^t(\mathcal{P}_v)$, $\mathbf{I}_v^{t-1}(\mathcal{P}_v)$, $\mathbf{B}_v(\mathcal{P}_v)$ and $|\mathbf{I}_v^t(\mathcal{P}_v) - \mathbf{B}_v(\mathcal{P}_v)|$. The vectors of orientations and magnitudes obtained

[1]The segmented regions are: each superpixel for SLIC, each block of pixels for block-based method, each mask for instance segmentation, distinct regions of one category for semantic segmentation, and distinct foreground regions for SuBSENSE.

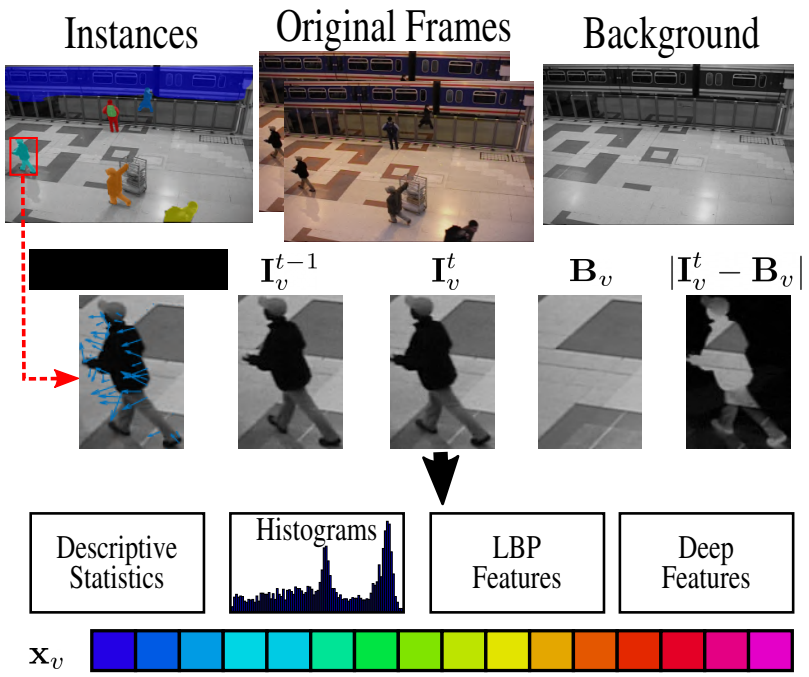

Figure 3-4: Procedure to represent the nodes of the graph with a Mask R-CNN as backbone. Each mask of the segmented image represents a node in the graph, and the representation of the node is achieved with intensity, optical flow, texture, and deep features.

from the optical flow vectors $\mathbf{v}_x^t(\mathcal{P}_v)$ and $\mathbf{v}_y^t(\mathcal{P}_v)$ are used to compute histograms and some descriptive statistics (the minimum, maximum, mean, standard deviation, mean absolute deviation, and range). Finally, the deep features of each segmented region are extracted. Inspired by the visual object tracking community [111, 112], we use a pre-trained VGG-m model [113] to extract the features from the 5th convolutional layer (Conv-5) and then a principal component analysis is applied to compress a high-dimensional feature vector into a low-dimensional vector. The representation of node v is obtained concatenating all the previous features, *i.e.*, optical flow, intensity, texture, and deep features. Each instance is represented by a M-dimensional vector $\mathbf{x}_v$. The readers are referred to Appendix B to see more details about $\mathbf{x}_v$.

3.3.3 Graph Construction

The construction of the graph aims to get geometrical information from the datasets, leading to a reduction in the number of labels required in the learning process. In

this chapter, the construction of the graph is achieved with a k-Nearest Neighborhood (k-NN) with a Gaussian kernel. Let $\mathbf{X} \in \mathbb{R}^{N \times M}$ be the matrix of N nodes, in which each node is an M-dimensional vector and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\mathsf{T}$. Firstly, the k-NN algorithm is used to connect the nodes in the graph. Afterward, vertices are connected to get an undirected and weighted graph. The weight between two connected vertices i, j is given such that $a_{ij} = \exp -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\sigma^2}$, where σ is the standard deviation of the Gaussian function given as $\sigma = \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$. As a result, the adjacency matrix $\mathbf{A}(i,j) = a_{ij} \; \forall \; (i,j) \in \mathcal{E}$ is obtained from this process.

A reasonable thought is to use a complete graph to avoid the optimization of parameter k. However, a complete graph requires a prohibitive amount of memory to store matrices $\mathbf{A}$ and $\mathbf{L}$ when dealing with huge graphs. For example, a complete graph with $N = 200000$ would require 320 gigabytes of memory to store $\mathbf{A}$, where each edge in $\mathbf{A}$ is represented with a variable of type double of 8 bytes. Other strategies for the representation of the graph can be used in GraphMOS. For example, Gangapure *et al.* [114] proposed a superpixel based causal multisensor video fusion method, where the key idea is to leverage temporal information for video and then construct spatiotemporal graph models.

3.3.4 Graph Signal

The graph signal is a matrix $\mathbf{Y} \in \mathbb{R}^{N \times Q}$, where Q is the number of classes of the problem. $\mathbf{y}_q = \mathbf{Y}_{:,q}$ is the graph signal associated with the qth class, where $\mathbf{Y}_{:,q}$ is the qth column vector of matrix $\mathbf{Y}$. Each row of $\mathbf{Y}$ represents if certain segmented region belongs to the qth class. In this chapter, the graph signal is given by the membership function $\mathbf{Y}^c$ of each class c, which takes a value of 1 on a node which belongs to the class and is 0 otherwise. For example, in MOS Q is equal to 2 that corresponds to the classes *static object* $[1, 0]$ and *moving object* $[0, 1]$. Given a segmented region, using some of the methods in Section 3.3.1, we can establish that node is a moving object if the statistical mode in the ground-truth is the class moving object and vice versa. Notice this can be easily extended to multiple classes, *e.g.*, in semantic segmentation with multiple classes.

3.3.5 Sampling of Graph signals and Sample Complexity for Semi-supervised Learning

Given the graph signals related to the problem of MOS and the notions of bandlimitedness in terms of $PW_\omega(G)$ from Section 2.2, we want to know what is the minimum amount of labeled nodes required to have a perfect classification in semi-supervised learning, given the prior assumption that the labels of the nodes are in the Paley-Wiener space of the graph? Similar to the sampling and reconstruction of graph signals in Section 2.3, the answer is that one needs at least ρ (bandwidth) labeled nodes to achieve perfect classification. This also holds for regression of graph signals, *i.e.*, given $\mathbf{y} \in PW_\omega(G)$, the number of sampled nodes required to get perfect reconstruction is ρ.

In Section 2.3, we provided Theorem 2.3.1 that states when perfect reconstruction is possible. One can relate Theorem 2.3.1 to the sample complexity in semi-supervised learning as follows:

Corollary 3.3.1. *Let $\mathbf{Y} \in \mathbb{R}^{N\times Q}$ be a graph signal associated with a semi-supervised learning problem, where Q is the number of classes; and let N_s be the sample complexity of the semi-supervised learning problem. $\mathbf{Y}_{i,:} = \boldsymbol{\delta}_q^\mathsf{T}$, where $\boldsymbol{\delta}_q$ is a Q-dimensional Kronecker column vector centered at q, and $\mathbf{Y}_{i,:}$ is the ith row of $\mathbf{Y}$. $\mathbf{Y}$ has a set of cutoff frequencies $\{\omega_1, \ldots, \omega_q\}$, with corresponding bandwidths $\{\rho_1, \ldots, \rho_q\}$ for each graph signal $\mathbf{Y}_{:,q}$ $\forall$ $1 \leq q \leq Q$. Then, N_s is bounded such that:*

$$N_s \leq \max\{\rho_1, \ldots, \rho_q\}. \tag{3.1}$$

Proof. From theorem 2.3.1, one can get a perfect reconstruction of each graph signal $\mathbf{Y}_{:,i} \in PW_{\omega_i}(G)$ $\forall$ $1 \leq i \leq q$ using Eqn. (2.2) with at least ρ_i samples. As a consequence, the worst-case scenario in the sample complexity N_s for perfect reconstruction of $\mathbf{Y}$ is $\max\{\rho_1, \ldots, \rho_q\}$. □

3.3.6 Minimization of the Sobolev Norm

One of the semi-supervised learning methods in this chapter is based on the variational splines of Pesenson [51].

Definition 3.3.1. *For a fixed $\epsilon \geq 0$ the Sobolev norm is introduced by the following formula:*

$$\|\mathbf{f}\|_{\alpha,\epsilon} = \|(\mathbf{L} + \epsilon\mathbf{I})^{\alpha/2}\mathbf{f}\|_2, \alpha \in \mathbb{R}. \tag{3.2}$$

Given a graph signal of sampled labels $\mathbf{y}_q(\mathcal{S}) = \mathbf{M}\mathbf{y}_q$, a positive $\alpha > 0$, and a non-negative $\epsilon \geq 0$, the variational problem for semi-supervised learning is stated as follows: find a vector $\mathbf{z}_q$ with the following properties: $\mathbf{z}_q(\mathcal{S}) = \mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S})$, and $\mathbf{z}_q$ minimizes functional $\mathbf{z}_q \to \|(\mathbf{L} + \epsilon\mathbf{I})^{\alpha/2}\mathbf{z}_q\|_2$. In other words, the variational problem solves the following optimization problem:

$$\begin{aligned} &\arg\min_{\mathbf{z}_q} \|\mathbf{z}_q\|^2_{\alpha,\epsilon} \quad \text{s.t.} \quad \mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S}) \to \\ &\arg\min_{\mathbf{z}_q} \mathbf{z}_q^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\alpha\mathbf{z}_q \quad \text{s.t.} \quad \mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S}) = 0. \end{aligned} \tag{3.3}$$

Equation (3.3) is a convex optimization problem since the term $\mathbf{z}_q^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\alpha\mathbf{z}_q$ is a quadratic convex function in $\mathbf{z}_q$; and the term $\mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S})$ is affine in $\mathbf{z}_q$. The semi-supervised learning problem is solved by determining the solution of (3.3) for $q = 1, \ldots, Q$.

The minimization of the Sobolev norm in (3.3) is closely related to the Laplacian quadratic form. The Laplacian quadratic form of $\mathbf{y}$ is given by $\mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y}$ as shown in (2.5). In the same way, $\mathbf{y}^\mathsf{T}\mathbf{L}^\alpha\mathbf{y}$ is known as the empirical iterated Laplacian regularizer or *higher-order regularization* [115] and has been used for regression and classification tasks in semi-supervised learning [50]. On the other hand, the Sobolev norm is such that $\mathbf{y}^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\alpha\mathbf{y}$. The Sobolev norm of $\mathbf{y}$ is the higher-order regularization with an addition of the semi-definite perturbation matrix $\epsilon\mathbf{I}$. For $\epsilon > 0$, $(\mathbf{L} + \epsilon\mathbf{I})$ is always invertible even though $\det(\mathbf{L}) = 0$ for undirected and connected graphs. Intuitively, the value ϵ is related to the stability of the inverse of $(\mathbf{L} + \epsilon\mathbf{I})$.

For the Laplacian matrix $\mathbf{L}$, we know that:

$$\kappa(\mathbf{L}) = \frac{|\lambda_{\max}(\mathbf{L})|}{|\lambda_{\min}(\mathbf{L})|} \approx \frac{\lambda_{\max}(\mathbf{L})}{0} \to \infty, \tag{3.4}$$

where $\kappa(\mathbf{L})$ is the condition number of $\mathbf{L}$. Since $\kappa(\mathbf{L}) \to \infty$, we have a bad-conditioned problem when relying on the Laplacian matrix alone. On the other hand, for the Sobolev term, we have that:

$$\mathbf{L} + \epsilon\mathbf{I} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}} + \epsilon\mathbf{I} = \mathbf{U}(\mathbf{\Lambda} + \epsilon\mathbf{I})\mathbf{U}^{\mathsf{T}}. \tag{3.5}$$

Therefore, $\lambda_{\min}(\mathbf{L} + \epsilon\mathbf{I}) = \epsilon$, *i.e.*, $\mathbf{L} + \epsilon\mathbf{I}$ is positive definite ($\mathbf{L} + \epsilon\mathbf{I} \succ 0$), and:

$$\kappa(\mathbf{L} + \epsilon\mathbf{I}) = \frac{|\lambda_{\max}(\mathbf{L} + \epsilon\mathbf{I})|}{|\lambda_{\min}(\mathbf{L} + \epsilon\mathbf{I})|} = \frac{\lambda_{\max}(\mathbf{L}) + \epsilon}{\epsilon} < \kappa(\mathbf{L}) \ \forall \ \epsilon > 0. \tag{3.6}$$

Namely, $\mathbf{L} + \epsilon\mathbf{I}$ has a better condition number than $\mathbf{L}$, where ϵ is fundamentally related to how well conditioned is the variational problem.

Since $(\mathbf{L} + \epsilon\mathbf{I})^{\alpha}$ is always invertible for $\alpha > 0$ and $\epsilon > 0$, one can show that the semi-supervised learning problem in (3.3) has a closed-form solution given by:

$$((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}}(\mathbf{M}((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}})^{-1}\mathbf{Y}(\mathcal{S}), \tag{3.7}$$

where $\mathbf{Y}(\mathcal{S})$ is the sub-matrix of $\mathbf{Y}$ with rows indexed by $\mathcal{S}$. The proof is shown in Appendix C. For small values of N, the minimization of the Sobolev norm can be solved with the closed-form solution in (3.7). For larger values of N, this minimization can be achieved with iterative methods such as the interior-point method.

3.3.7 Minimization of the Total Variation

Another reconstruction algorithm for semi-supervised learning in this chapter is based on the Total Variation (TV) of graph signals. TV of graph signals is defined as [116]:

$$\|\mathbf{y}\|_{\text{TV}} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \sqrt{\mathbf{A}(i,j)} \|\mathbf{y}(j) - \mathbf{y}(i)\|_1. \tag{3.8}$$

The minimization of the TV of $\mathbf{y}$ is related to the cluster assumption [117], and end ups in piecewise constant signals. The semi-supervised algorithm solves the following optimization problem:

$$\arg\min_{\mathbf{z}_q} \|\mathbf{z}_q\|_{\text{TV}} \quad \text{s.t.} \quad \mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S}), \tag{3.9}$$

for $q = 1, \ldots, Q$. Basically, Eqn. (3.9) is minimizing the TV of the reconstructed graph signal such that $\mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S})$. The minimization of the TV in (3.9) involves a non-differentiable objective function, which discards any gradient descent method. In the current chapter, the minimization of the TV is solved with a primal dual approach [118]. Let $\mathbf{P} \in \mathbb{R}^{|\mathcal{E}|\times N}$ be the *incidence matrix* of G defined as:

$$\mathbf{P}(e,i) = \begin{cases} \sqrt{\mathbf{A}(e)} & \text{if } i = \min\{i,j\} \\ -\sqrt{\mathbf{A}(e)} & \text{if } i = \max\{i,j\} \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

where $i \in \mathcal{V}$ and $e \in \mathcal{E}$ (in this case the edges are represented by the numbers $\{1, \ldots, |\mathcal{E}|\}$). The matrix $\mathbf{P}$ allows to represent the TV of the graph signal as $\|\mathbf{y}\|_{\text{TV}} = \|\mathbf{P}\mathbf{y}\|_1$ [117]. As a consequence, Eqn. (3.9) can be formulated as the primal equivalent unconstrained convex optimization problem:

$$\arg\min_{\mathbf{z}_q} g(\mathbf{P}\mathbf{z}_q) + h(\mathbf{z}_q), \tag{3.11}$$

$$g(\mathbf{x}) \triangleq \|\mathbf{x}\|_1, \text{ and } h(\mathbf{z}_q) \triangleq \begin{cases} \infty & \text{if } \mathbf{z}_q \notin \mathcal{Q} \\ 0 & \text{if } \mathbf{z}_q \in \mathcal{Q}, \end{cases} \tag{3.12}$$

where $h(\mathbf{z}_q)$ is the indicator function and $\mathcal{Q} = \{\mathbf{z}_q \in \mathbb{R}^N : \mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S})\}$.

Definition 3.3.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$. The function $f' : \mathbb{R}^n \to \mathbb{R}$ is the convex conjugate such that $f'(\mathbf{x}) \triangleq \sup_{\bar{\mathbf{x}}} \mathbf{x}^\mathsf{T}\bar{\mathbf{x}} - f(\bar{\mathbf{x}})$ [119].*

The dual problem associated with the minimization of the TV is given by $\arg\max_{\mathbf{x}} - h'(-\mathbf{P}^\mathsf{T}\mathbf{x}) - g'(\mathbf{x})$, where h' is the convex conjugate of the function h. The mini-

mization of the TV is solved with the primal-dual approach of first order [118] using the Unlocbox toolbox [120]. For further details, the readers are referred to the references [117, 120, 121].

3.3.8 GraphMOS in a Nutshell

GraphMOS is a transductive algorithm because they require that all nodes in the graph are present to solve the semi-supervised learning problem. Our algorithm is introduced with a batch method for superpixel segmentation with parameters batch size η, and ground-truth size χ, where η is the number of frames that will be processed on each iteration, and χ is the number of frames with ground-truth that will be randomly selected from other sequences (unseen scheme) to solve the semi-supervised learning problem. Algorithm 1 shows the pseudo-code for GraphMOS with batch processing. GraphMOS can also be solved using the whole dataset to construct the graph, but in this case, our algorithm will select a subset of nodes $\mathcal{S} \in \mathcal{V}$ with an unseen scheme to construct the graph signal $\mathbf{Y}$. Notice that in the case of processing the whole dataset the parameter η is not required, but one has to use instance or semantic segmentation methods to reduce the number of nodes. Also, notice that the parameters ϵ and α are not required when solving the semi-supervised learning problem with TV minimization.

3.4 Experimental Framework

This section introduces the datasets used in the current chapter, the evaluation metrics, the experiments, and the implementation details of GraphMOS.

3.4.1 Datasets

Our proposed algorithm is evaluated on a variety of datasets for MOS. More specifically, we evaluated the performance of GraphMOS for the sequences taken from static and moving cameras on several background modeling challenges.

Algorithm 1 Graph Moving Object Segmentation

Input: Video sequences in the dataset.
Initialization: Parameters ζ, k, ϵ, α, η, χ.

Select zth video to be segmented
while there are still frames to process in video z **do**
  Select η frames, in order, from z for batch processing
  Randomly select χ frames from other videos
  Compute ζ superpixels for the $\eta + \chi$ frames
  Compute $\mathbf{X}$ from the $\eta + \chi$ frames
  Compute $\mathbf{Y}$ from the χ ground-truth frames
  Find the set of k-NN of $\mathbf{x}_i \; \forall \; i \in \mathcal{N}$
  Compute $\sigma = \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$
  for (i, j) in the set of k-NN **do**
    $\mathbf{A}(i, j) = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/\sigma^2\right)$
  end for
  $\mathbf{D} = \text{diag}(\mathbf{A1})$, $\mathbf{L} = \mathbf{D} - \mathbf{A}$
  Solve either (3.3) or (3.9) for $q = 1, \ldots, Q$
  Get $\mathbf{Z}$ as the solution of the semi-supervised problem
  Use $\mathbf{Z}$ to compute the moving objects of the η frames
end while

MOS from Static Camera

For sequences taken from static camera, GraphMOS is evaluated on three datasets including CDNet2014 [10], I2R [76], and SBI2015 [65]. CDNet2014 is the reference dataset in the MOS community. This dataset is categorized into 11 main challenges including, bad weather, low frame rate, night videos, turbulence, baseline, dynamic backgrounds, PTZ, camera jitter, intermittent object motion, shadow, and thermal. PTZ category presents sequences taken from PTZ cameras, while the camera jitter category contains sequences of jittering effects. Each challenge contains from four up to six videos. Every video contains a certain amount of ground-truth frames, in which the ground truth shows the foreground and background. The average resolution of the sequences is 320×240.

I2R dataset contains nine challenging sequences with an average resolution of 240×192. Each sequence contains only 20 images of ground-truth. SBI2015 dataset contains 14 sequences with an average resolution of 328×246. The dataset presents the challenges of cluttered moving objects and slow object motion. This dataset was

originally designed to evaluate background initialization methods. However, Wang *et al.* provided the ground-truth for MOS for SBI2015 [99].

MOS for Moving Camera

GraphMOS is also evaluated on the UCSD dataset [11] containing 18 challenging moving camera sequences. This dataset also presents severe dynamic background variations. Each video of UCSD is partially or fully annotated with pixel-level ground-truth images of foreground and background. The average resolution of the dataset is 230×320.

3.4.2 Evaluation Metrics

The F-measure metric is used for the MOS task to compare the performance of our algorithm with state-of-the-art methods [10]. The F-measure metric is defined as follows:

$$\text{F-measure} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \text{ where} \tag{3.13}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \ \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3.14}$$

where TP, FP, and FN are the number of true positives, false positives, and false negatives pixels, respectively.

3.4.3 Experiments

A thorough comparison with state-of-the-art methods, along with a set of comprehensive ablation studies of GraphMOS are performed. The ablation studies analyze some specific elements of the pipeline in Fig. 3-2, like the segmentation method, the number of superpixels in the case of SLIC, the feature extraction procedure, the construction of the graph and the semi-supervised learning algorithm. The instance segmentation methods are performed with Mask R-CNN using ResNet-50 as the backbone, while Cascade Mask R-CNN is performed using ResNeSt-200.

Table 3.1: Summary of the parameters used in our experiments for each dataset.

Dataset	ζ	k	ϵ	α	η	χ
CDNet2014	200	30	0.2	1	100	350
I2R	200	30	0.2	1	100	150
SBI2015	300	30	0.2	1	100	60
UCSD	200	30	0.2	1	100	40

Finally, the last experiment is devoted to testing Corollary 3.3.1. Since the calculation of $\mathbf{U}$ is computationally prohibitive for the configurations involved in the ablation studies, in this case, a graph with $N = 7443$ nodes is constructed with the sequences *backdoor*, *bungalows*, and *busStation* of the challenge shadow of CDNet2014, using a k-NN with $k = 30$, and a Mask R-CNN. In practice, graph signals are only approximately bandlimited [53], *i.e.*, the GFT of $\mathbf{y}$ has an exponentially decaying shape. In this experiment, we compute the bandwidth as the ρ where we have at least 90% of the spectral energy of $\mathbf{y}$. The spectral energy is defined as $\sum_{i\in\mathcal{V}} \hat{\mathbf{y}}^2(i)$. As a consequence of this approximation, the classification error can be bounded by some value ϕ [52]. The last experiment computes the classification error in the non-sampled nodes for the sample sizes in the set $m = \{10, 20, 30, \ldots, 400\}$, using random sampling and the Sobolev minimization with $\epsilon = 0.2$ and $\alpha = 1$. This experiment is performed with a Monte Carlo cross-validation with 200 repetitions.

3.4.4 Parameters Settings

Our algorithms contain several parameters such as the number of superpixels ζ, the number of k neighbors for k-NN in the graph construction, the parameters ϵ and α for the Sobolev norm, the batch size η, and the number of selected ground-truth frames χ. Table 3.1 shows a summary of the best parameters for GraphMOS within each dataset in the current work. Table 3.9 shows an ablation study for the parameter ζ.

3.4.5 Implementation Details

The instance and semantic segmentation algorithms were implemented using Pytorch and Detectron2 [122]. The algorithm for the reconstruction of graph signals

was implemented using the graph signal processing toolbox [123] and the Matlab convex optimization toolbox [120]. The code has been made available[2]. For the comparison with our algorithm, most of the MOS methods were implemented with the BGSLibrary [124] and the LRSLibrary [125] with default parameters by reference. The experiments are executed on a laptop with a processor Intel Core i7 with 16 gigabytes of memory RAM. Using handcrafted features, GraphMOS algorithm with Mask R-CNN takes 3.73 FPS, with Cascaded Mask R-CNN it takes 1.76 FPS, with Deeplab method it takes 13.14 FPS, and with SuBSENSE method GraphMOS takes 15.11 FPS to segment the moving objects. In case of using deep features with block and superpixel-based segmentation methods, the experiments of GraphMOS are conducted on a powerful Nvidia DGX-2 server machine. GraphMOS takes 1.42 and 2.31 FPS for MOS using superpixel and block-based graph construction.

3.5 Results and Discussion

GraphMOS is compared with 20 state-of-the-art methods including SWCD [126], FTSG [127], SuBSENSE [104], WeSamBE [128], PAWCS [12], WisenetMD [129], IUTIS-5 [13], SemanticBGS [130], BSUV-net [14], MoG [89], DECOLOR [16], ViBe [131], 3WD [60], GRASTA [132], FgSegNet v2 [67], ROSL [15], ADMM [133], noncvxRPCA [134], and OR1MP [135] for the MOS task. Mask R-CNN can directly solve the MOS problem. However, Mask R-CNN does not include temporal information, leading to an ill-conditioned problem. In our experiments, we have also trained Mask R-CNN directly on CDNet2014 sequences with an unseen scheme. Each Mask R-CNN is trained with 1000 epochs using learning rate of 0.00025 and a batch size of 2.

3.5.1 Qualitative Evaluations

Fig. 3-5 shows some of the visual results of the proposed algorithms compared with several state-of-the-art methods on CDNet2014 and UCSD datasets. The CDNet2014

[2] `https://github.com/jhonygiraldo/GraphMOS`

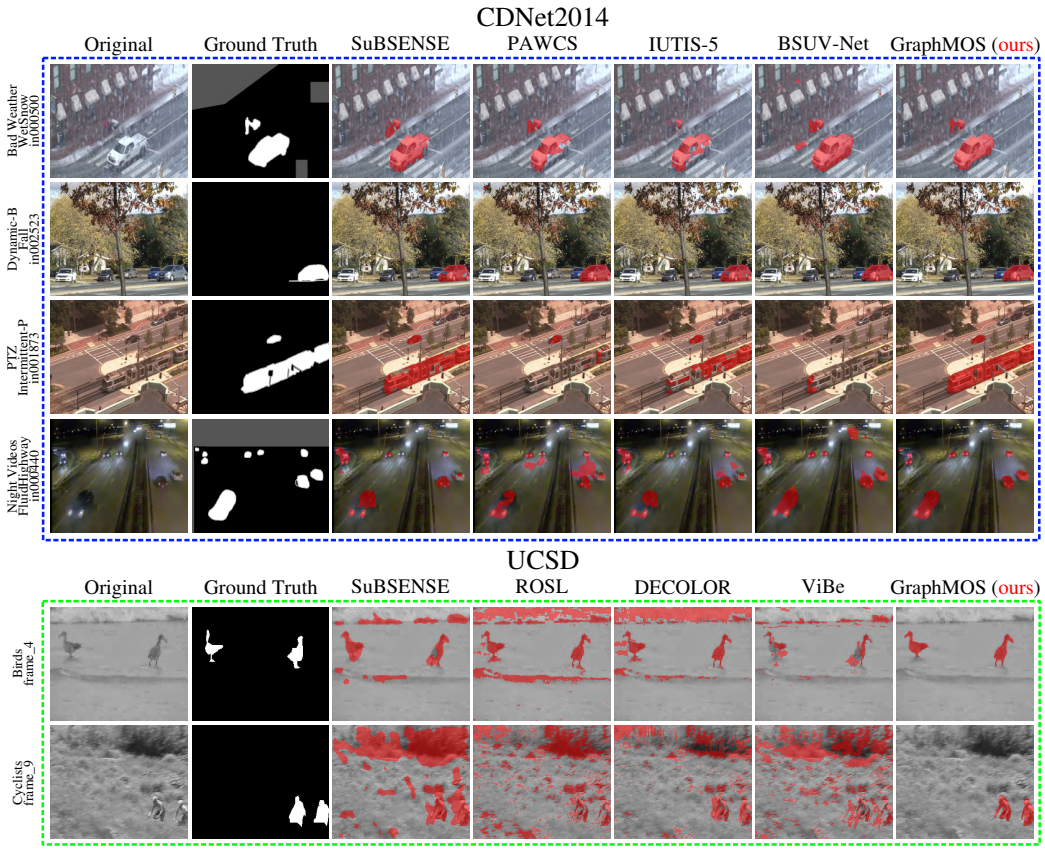

Figure 3-5: Comparison of the qualitative results of GraphMOS on CDNet2014 and UCSD datasets with existing state-of-the-art methods. Our algorithm performs better than the state-of-the-art methods in these challenging scenarios.

sequences *WetSnow*, *Fall*, *IntermittentPan*, and *FluidHighway* present bad weather conditions, dynamic background variations, panning of the scene, and nighttime lighting variations challenges. GraphMOS shows the best visual results, while BSUV-Net and SuBSENSE show competitive performance for these sequences. *Birds* and *Cyclists* sequences of UCSD present highly dynamic background variations challenges, including rippling water surface and swaying of bushes. The compared methods do not handle these sequences accurately, while only GraphMOS can handle these sequences successfully.

3.5.2 Quantitative Results

Tables 3.2-3.5 show the comparisons of the qualitative results of the GraphMOS algorithm on CDNet2014, I2R, SBI2015, and UCSD datasets for MOS. Within these

Table 3.2: Comparisons of average F-measure on CDNet2014 dataset.

Challenge	SWCD	FTSG	SuBSENSE	WeSamBE	PAWCS	WisenetMD	IUTIS-5	SemanticBGS	FgSegNet v2	Mask R-CNN	BSUV-Net	GraphMOS
Bad Weather	0.8233	0.8228	0.8619	0.8608	0.8152	0.8616	0.8248	0.8260	0.3277	0.7952	**0.8713**	**0.9411**
Baseline	0.9214	0.9330	0.9503	0.9413	0.9397	0.9487	0.9567	0.9604	0.6926	0.7608	**0.9693**	**0.9710**
Camera Jitter	0.7411	0.7513	0.8152	0.7976	0.8137	0.8228	0.8332	**0.8388**	0.4266	0.7021	0.7743	**0.9233**
Dynamic-B	0.8645	0.8792	0.8177	0.7440	**0.8938**	0.8376	0.8902	**0.9489**	0.3634	0.5880	0.7967	0.8922
I-O Motion	0.7092	**0.7891**	0.6569	0.7392	0.7764	0.7264	0.7296	**0.7878**	0.2002	0.4724	0.7499	0.6455
Low-F rate	0.7374	0.6259	0.6445	0.6602	0.6588	0.6404	**0.7743**	**0.7888**	0.2482	0.5776	0.6797	0.6910
Night Videos	0.5807	0.5130	0.5599	0.5929	0.4152	0.5701	0.5290	0.5014	0.2800	0.3898	**0.6987**	**0.8211**
PTZ	0.4545	0.3241	0.3476	0.3844	0.4615	0.3367	0.4282	0.5673	0.3503	0.6214	**0.6282**	**0.8511**
Shadow	0.8779	0.8832	0.8986	0.8999	0.8913	0.8984	0.9084	**0.9478**	0.5295	0.8781	0.9233	**0.9901**
Thermal	**0.8581**	0.7768	0.8171	0.7962	0.8324	0.8152	0.8303	0.8219	0.6038	0.4986	**0.8581**	**0.9010**
Turbulence	0.7735	0.7127	0.7792	0.7737	0.6450	**0.8304**	0.7836	0.6921	0.0643	0.1707	0.7051	**0.8233**
Overall	0.7583	0.7283	0.7408	0.7446	0.7403	0.7535	0.7717	**0.7892**	0.3715	0.5868	0.7868	**0.8592**

Table 3.3: Comparison of F-measure results over the sequences of I2R dataset.

Sequence	MoG	DECOLOR	ViBe	3WD	GRASTA	SuBSENSE	ROSL	ADMM	noncvxRPCA	OR1MP	BSUV-Net	BSUV-Net+	GraphMOS
Bootstrap	0.6328	0.6248	0.6134	0.5393	0.6017	0.6704	0.6449	0.5085	0.6142	0.6459	0.7133	**0.7636**	**0.9722**
Campus	0.2818	0.7652	0.4604	0.2258	0.2152	0.7498	0.1746	0.1913	0.1742	0.1714	0.8468	**0.8543**	**0.9711**
Curtain	0.6276	0.8342	0.6781	0.5127	0.7816	**0.9436**	0.7009	0.5211	0.4747	0.4624	0.9055	0.9373	**0.9899**
Escalator	0.5204	0.7183	0.5988	0.4706	0.4265	0.5756	0.3967	0.3716	0.4205	0.3673	0.6994	**0.7557**	**0.9081**
Fountain	0.7428	**0.8618**	0.5635	0.3500	0.6620	0.7937	0.2835	0.2185	0.2664	0.2646	**0.8644**	0.8306	0.8516
Hall	0.6152	0.5597	0.6377	0.5036	0.5355	0.7708	0.6751	0.3898	0.4819	0.5450	**0.8118**	0.7654	**0.8670**
Lobby	0.5760	0.5654	0.1488	0.5958	0.4059	0.2412	0.2329	0.1453	0.5157	0.2207	**0.7659**	0.7213	**0.9555**
Shopping.	0.6820	0.6800	0.5418	0.6832	0.6724	0.7594	0.6927	0.3764	0.6749	0.6198	**0.8070**	0.7619	**0.7712**
WaterSur.	0.6500	0.8873	0.8544	0.3008	0.7725	0.9365	0.6706	0.5479	0.4651	0.3607	0.9358	**0.9494**	**0.9899**
Overall	0.5920	0.7219	0.5663	0.4646	0.5635	0.7157	0.4969	0.3634	0.4542	0.4064	**0.8166**	0.8155	**0.9196**

tables, the best and second best performing methods are shown in **red** and **blue**, respectively. Our proposed algorithms show competitive performance as compared to state-of-the-art methods on all datasets.

For the CDNet2014 dataset (Table 3.2), our proposed algorithm GraphMOS obtained the best performance in terms of average F-measure score of 85.92%, which is 7.0% better than the second-best performing method SemanticBGS. Moreover, GraphMOS achieved the best results in seven out of eleven challenges including *Bad Weather* (94.11%), *Baseline* (97.10%), *Camera Jitter* (92.33%), *Night Videos* (82.11%), *PTZ* (85.11%), *Shadow* (99.01%), and *Thermal* (90.10%) while it obtained favorable performance for the remaining attributes including *Dynamic Background*, *Turbulence*, *Low Framerate*, and *Intermittent Object Motion*. In all these challenges, the sequences in *NightVideos* and *PTZ* are very challenging since the majority of the compared methods are not able to achieve more than 70.0% F-measure score, while GraphMOS achieved significantly high performance for these challenges.

In the I2R dataset (Table 3.3), GraphMOS has obtained the best performance of 91.96% overall, which is almost 10.0% better than the second-best performing

Table 3.4: Comparison of F-measure results over the sequences of SBI2015 dataset.

Sequence	MoG	DECOLOR	ViBe	3WD	GRASTA	SuBSENSE	ROSL	ADMM	noncvxRPCA	OR1MP	BSUV-Net	BSUV-Net+	GraphMOS
Board	0.7656	0.5033	0.7377	0.3959	0.5592	0.6588	0.6440	0.1420	0.5304	0.5259	0.9560	**0.9886**	**0.9931**
CAVIAR1	0.5974	0.9183	0.8051	0.4282	0.2104	0.8783	0.8533	0.5065	0.4204	0.4653	0.9260	**0.9358**	**0.9744**
CAVIAR2	0.4597	0.4324	0.7347	0.4470	0.0463	**0.8740**	0.2966	0.2452	0.1933	0.1813	0.8572	0.8649	**0.9210**
CaVignal	0.5057	0.4827	0.3497	0.3767	0.3812	0.4080	**0.6226**	0.4305	0.4720	0.3727	0.4628	0.4773	**0.7322**
Candela	0.4927	0.5025	0.5020	0.4702	0.1929	0.6959	0.4913	0.4410	0.4730	0.6941	**0.8997**	**0.8597**	0.7551
Hall&Mon.	0.5952	0.7826	0.6017	0.4479	0.1491	0.7559	0.6081	0.1521	0.4525	0.4827	0.8931	**0.9346**	**0.9122**
HighwayI	0.6272	0.6976	0.4150	0.4123	0.5522	0.5073	0.6836	0.5170	0.5733	0.5660	**0.8440**	0.8337	**0.9880**
HighwayII	0.8144	0.8925	0.5554	0.7426	0.4833	0.8779	0.7808	0.7429	0.7335	0.7287	**0.9690**	**0.9592**	0.9547
HumanBod.	0.7553	0.8265	0.4268	0.5074	0.5352	0.8560	0.7606	0.3115	0.5765	0.5954	0.9314	**0.9503**	**0.9522**
IBMtest2	0.7108	0.8823	0.7001	0.8162	0.3825	0.9281	0.8579	0.5076	0.6714	0.6460	**0.9722**	0.9643	**0.9856**
People&Fol.	0.5920	0.2601	0.6111	0.2911	0.3460	0.4251	0.4108	0.0569	0.3924	0.3754	**0.8837**	0.6930	**0.9059**
Toscana	0.6093	0.3669	0.7307	0.3132	0.4188	0.8256	0.7083	0.2533	0.6779	0.4320	0.9110	**0.9193**	**0.9411**
Foliage	**0.5786**	0.3178	0.5539	0.3376	0.4148	0.1962	0.4341	0.2105	0.4617	0.4481	0.4371	0.3450	**0.7792**
Snellen	**0.5498**	0.4023	0.3083	0.3213	0.4104	0.2467	0.4052	0.1099	0.4345	0.4083	0.3674	0.3786	**0.7380**
Overall	0.6181	0.5906	0.5737	0.4506	0.3630	0.6524	0.6112	0.3305	0.5045	0.4944	**0.8079**	0.7932	**0.9328**

Table 3.5: Comparison of F-measure results over the videos of UCSD background subtraction dataset.

Sequence	MoG	DECOLOR	ViBe	3WD	GRASTA	SuBSENSE	ROSL	ADMM	noncvxRPCA	OR1MP	BSUV-Net	BSUV-Net+	GraphMOS
Birds	0.1427	0.1457	0.3354	0.1308	0.1320	**0.4832**	0.1478	0.0227	0.1432	0.1394	0.3314	0.2625	**0.7897**
Boats	0.0881	0.2179	0.1854	0.1576	0.0678	0.4550	0.1637	0.1212	0.1380	0.1100	0.4586	**0.6621**	**0.8311**
Bottle	0.1856	0.4765	0.4512	0.1364	0.1159	0.6570	0.2069	0.6589	0.1974	0.1795	**0.8528**	0.5039	**0.9011**
Chopper	0.3237	0.6214	0.4930	0.3171	0.0842	**0.6723**	0.2920	0.1250	0.3103	0.2653	0.2805	0.3020	**0.8230**
Cyclists	0.0915	0.2224	0.1211	0.1003	0.1243	0.1445	0.1366	0.1093	0.1317	0.1242	0.0051	**0.4138**	**0.7911**
Flock	0.2706	0.2943	0.2306	0.2007	0.1612	0.2492	**0.3409**	0.1088	0.3220	0.2605	0.1160	0.0025	**0.6722**
Freeway	0.2622	**0.5229**	0.4002	0.5028	0.0814	**0.5518**	0.3875	0.0816	0.3126	0.1549	0.4780	0.1185	0.4831
Hockey	0.3867	0.3449	0.4195	0.2789	0.3149	0.3611	0.4106	0.2981	0.3411	0.4296	0.6460	**0.6908**	**0.8211**
Jump	0.2679	0.3135	0.2636	0.2481	0.4175	0.2295	0.4198	0.0609	0.3180	0.3073	0.5491	**0.8697**	**0.8233**
Landing	0.0335	0.0640	0.0433	0.0457	0.0414	0.0026	0.0506	**0.0826**	0.0480	0.0442	0.0021	0.0012	**0.4122**
Ocean	0.1113	0.1315	0.1648	0.2055	0.1144	0.2533	0.1422	0.1809	0.1274	0.1252	0.4117	**0.5335**	**0.9422**
Peds	0.3731	**0.7942**	0.5257	0.7536	0.4653	0.5154	0.7418	0.6667	0.4333	0.4297	0.6958	0.6738	**0.9411**
Skiing	0.2038	**0.3473**	0.1441	0.1981	0.0927	0.2482	0.1942	0.0519	0.1812	0.1791	0.0841	0.0602	**0.7622**
Surf	0.0489	0.0647	0.0462	0.0579	0.0523	0.0467	0.0453	0.0162	0.0325	0.0317	**0.0884**	0	**0.7322**
Surfers	0.0542	0.1959	0.1189	0.0962	0.0742	0.1393	0.1184	0.1950	0.1083	0.1044	0.2612	**0.4776**	**0.7591**
Trafic	0.2188	**0.2732**	0.1445	0.2032	0.0368	0.1165	0.1042	0.1044	0.0949	0.0882	0	0	**0.6670**
Overall	0.1914	0.3144	0.2555	0.2271	0.1485	0.3203	0.2439	0.1803	0.2025	0.1858	0.3288	**0.3483**	**0.7595**

BSUV-Net method. BSUV-Net+, SuBSENSE, and DECOLOR methods have obtained favorable performance, while the remaining compared methods are not able to achieve more than 60.0% F-measure score, which further shows the challenging nature of the sequences present in this dataset.

In the SBI2015 dataset (Table 3.4), the GraphMOS has also achieved the best performance in 11 out of 14 videos. Overall, the proposed algorithm obtained a 93.28% F-measure score, which is approximately 12.50% larger than BSUV-Net+ (80.79%) and 13.96% larger than BSUV-Net (79.32%) methods. The sequences in this dataset show clutter background scenes and slowly moving foreground objects. Therefore, the majority of the compared methods are not able to handle the overwhelming outliers of the moving objects in these sequences efficiently as compared to GraphMOS.

Similarly, in the case of the UCSD dataset (Table 3.5), GraphMOS also shows the best performance of 75.95%. The sequences in this dataset are taken from a moving camera. Therefore, it can be noticed in the performance comparison that none of the compared methods can perform favorably better since state-of-the-art methods such as MoG, DECOLOR, ViBe, GRASTA, SuBSENSE, and ADMM are designed to handle the static camera sequences. Our proposed GraphMOS algorithm shows better performance because of its generalization capabilities to tackle unseen videos on both static and moving cameras.

3.5.3 Ablation Studies

Several ablation studies are performed to analyze the performance of our proposed algorithm. These ablations include the analysis of different segmentation methods used in graph construction, feature extraction, the number of superpixels used in the SLIC method, the construction of the graph, and the semi-supervised learning method.

Segmentation Methods for Graph Nodes

Table 3.6 shows the performance comparison of GraphMOS for I2R, CDNet2014, SBI2015, and UCSD datasets using different segmentation methods for node representation during graph construction. Overall, the superpixel segmentation-based node representation for graph construction achieves the best performance as compared to other heavyweight semantic and instance segmentation methods. This is because the superpixel method segments all the homogeneous regions in the video frames, and then the graph is constructed for the semi-supervised learning task. In contrast, the instances of the videos such as moving objects, static objects, or other undesirable objects may or may not be segmented accurately by the heavyweight deep learning DeepLab and Mask R-CNN methods. Besides, the degradation in the performance of DeepLab with respect to the instance segmentation methods also suggests the unsuitability of semantic segmentation to solve the MOS problem.

Table 3.6: Performance comparisons in terms of average F-measure score for different segmentation methods used for graph construction. Only handcrafted features are used to report the performance. These ablations studies involves: graph construction using DeepLab with ResNet 101 (DeepLab), Mask R-CNN with ResNet 50 (Mask R-50), Cascade Mask R-CNN with ResNeSt 200 (Cascade RS-200), SuBSENSE, and Superpixel.

Dataset	SuBSENSE	DeepLab	Mask R-50	Cascade RS-200	Superpixel
I2R	0.7527	0.4145	**0.8397**	0.8196	**0.8510**
CDNet2014	**0.7396**	0.6119	0.6466	0.7283	**0.8138**
SBI2015	0.6488	0.7534	0.7538	**0.7832**	**0.8611**
UCSD	0.3388	0.4088	0.5553	**0.6751**	**0.7113**

Table 3.7: Performance comparison in terms of average F-measure score of superpixel and block-based segmentation for graph construction methods. The performance is reported by using both handcrafted and deep features representation of graph nodes.

Segmentation	I2R	CDNet2014	SBI2015	UCSD
Block-based	**0.8408**	**0.7494**	**0.8080**	**0.6607**
Superpixel	**0.9196**	**0.8592**	**0.9328**	**0.7595**

Table 3.7 shows the performance comparisons of superpixel and block-based node representation in graph construction. The average F-measure is reported by using handcrafted plus deep features representation of the node. Superpixel-based method clearly outperforms the block-based node representation method in all datasets.

Features Analysis

To analyze the effectiveness of the proposed algorithms, we also compare the MOS results with different features extracted from graph nodes on five different datasets. Table 3.8 shows the performance comparisons of the proposed algorithms using different features representation. The deep features are extracted from the $4th$ and $5th$ convolutional layers of the VGG-m model [113]. Overall, the deep features show competitive performance as compared to handcrafted features, while the incorporation of both deep features (Conv-5) and handcrafted features further improves the average F-measure score in all datasets.

Table 3.8: Performance comparisons in terms of average F-measure score on five datasets using distinct node features representations. Handcrafted, deep features, and the concatenation of handcrafted and deep features (Hand + Deep (Conv-5)) are used to represent graph nodes.

Features	I2R	CDNet2014	SBI2015	UCSD	DAVIS2016
Handcrafted features	0.8510	0.8210	0.8611	0.7113	0.8711
Deep Features (Conv-5)	**0.8701**	**0.8491**	**0.8722**	**0.7320**	**0.8993**
Deep Features (Conv-4)	0.8665	0.8410	0.8711	0.7222	0.8801
Hand + Deep (Conv-5)	**0.9196**	**0.8673**	**0.8952**	**0.7595**	**0.9121**

Table 3.9: Performance comparisons in terms of average F-measure score for the number of superpixels in the SLIC method.

Superpixels (ζ)	I2R	CDNet2014	SBI2015	UCSD	DAVIS2016
100	0.9054	0.8397	0.8632	0.7265	0.8810
200	**0.9196**	**0.8592**	0.8790	**0.7595**	0.9020
300	**0.9191**	**0.8555**	**0.8952**	**0.7496**	**0.9120**
400	0.9094	0.8473	**0.8810**	0.7406	**0.9030**
500	0.8983	0.8442	0.8734	0.7363	0.8821

Number of Superpixels

Table 3.9 shows the performance comparison of the proposed algorithms with varying number of superpixels. The best performance for each dataset is given by 200 or 300 superpixels per image.

Construction of the Graph

Table 3.10 summarizes the performance of GraphMOS with several configurations of the construction of the graph in the CDNet2014 dataset. The results suggest that: the challenges dynamic background, intermittent object motion, low frame rate, and thermal have the better results with $k = 10$; the challenges camera jitter and shadow have the better results with $k = 20$; and finally the results with $k = 30$ and $k = 40$ are very similar. These differences between the results can be partially explained because for $k = 10$ and $k = 20$ some sub-graphs are generated, *i.e.*, for the semi-supervised learning algorithm is executed independently for each sub-graph. On the other hand, G is connected for $k = 30$ and $k = 40$. As a consequence, for low values of k, GraphMOS is losing global information of the database. The good results in the challenge low frame rate motion can be explained in part because of the nature

Table 3.10: Average F-measure with variations in the construction of the graph. This ablation involves: k-NN with $k = 40$, $k = 30$, $k = 20$, and $k = 10$.

Challenge	$k = 40$	$k = 30$	$k = 20$	$k = 10$
Bad Weather	0.8294	**0.8372**	**0.8307**	0.8294
Baseline	**0.9398**	**0.9424**	0.9394	0.9307
Camera Jitter	**0.7005**	0.7001	**0.7016**	0.6848
Dynamic-B	0.7334	0.7431	**0.7799**	**0.7996**
I-O Motion	0.3607	0.4051	**0.4632**	**0.5539**
Low-F rate	0.5538	0.5581	**0.5584**	**0.5647**
PTZ	**0.7599**	0.7486	0.7499	**0.7595**
Shadow	0.9653	**0.9658**	**0.9659**	0.9647
Thermal	0.7292	0.7294	**0.7306**	**0.7351**
Overall	0.7302	0.7366	**0.7466**	**0.7580**

Table 3.11: Average F-measure with variations in the semi-supervised learning algorithm. This ablation involves: Sobolev minimization (Sob.) with $\epsilon = 50$, $\epsilon = 0.5$, $\epsilon = 0.2$, and Total Variation minimization (TV).

Challenge	Sob. $\epsilon = 50$	Sob. $\epsilon = 0.5$	Sob. $\epsilon = 0.2$	TV
Bad Weather	0.8096	**0.8337**	**0.8372**	0.8291
Baseline	0.8958	0.9301	**0.9424**	**0.9394**
Camera Jitter	0.6443	0.6952	**0.7001**	**0.7024**
Dynamic-B	**0.7706**	**0.7760**	0.7431	0.7486
I-O Motion	**0.5306**	**0.4447**	0.4051	0.3500
Low-F rate	0.5254	0.5567	**0.5581**	**0.5928**
PTZ	0.7420	**0.7495**	0.7486	**0.7899**
Shadow	0.9463	**0.9650**	**0.9658**	**0.9658**
Thermal	0.7158	**0.7286**	**0.7294**	0.7229
Overall	0.7312	**0.7422**	0.7366	**0.7379**

of these sequences since there is not a smooth change between frames, and perhaps the optical flow features do not work well in this challenge. As a consequence, the algorithm may take advantage of the local information of similar sequences in the same challenge. This is opposite to the idea to leverage global information to reduce the sample complexity in GraphMOS.

Semi-supervised Learning

Table 3.11 shows the performance of GraphMOS with several configurations of the semi-supervised learning algorithm. The results suggest that the minimization of the TV is better in the challenges: camera jitter, low frame rate, PTZ, and shadow; while the minimization of the Sobolev norm is better for the other challenges. Both Sobolev norm and TV minimization show complementary results. In a specific application one

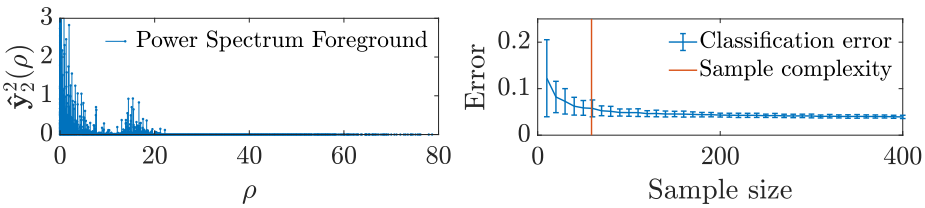

Figure 3-6: Results of the experiment related to the sample complexity. Left: power spectrum of the graph signal $\hat{\mathbf{y}}_2$ related the moving objects, right: classification error vs sample size of the semi-supervised learning algorithm.

can decide between Sobolev norm or TV minimization based on the specific challenges of the problem; or one can rely on the complexity of the algorithm, notice that the Sobolev norm minimization problem is a differentiable objective function, while TV minimization is non-differentiable.

3.5.4 Sample Complexity

Figure 3-6 shows the results of the experiment related to the sample complexity. The first plot in Fig. 3-6 shows the power spectrum of the graph signal associated with the moving objects, where the power spectrum is defined as the GFT square $\hat{\mathbf{y}}_i^2$. The second plot shows the classification error vs the sample size, and the bound of the sample complexity in the problem computed as in (3.1). The sample complexity computed in this experiment is $N_s \leq 59$ with corresponding cutoff frequency $\omega = 0.0089$. The average errors are 0.0576 and 0.0393 for the sampling sizes $m = 60$ and $m = 400$, respectively, *i.e.*, the change in the classification error is bounded by $\phi = 0.0576$ after the sample complexity. One can notice from Fig. 3-6 that the classification error just has a small change after a sample size of N_s.

3.6 Conclusions

In this chapter, we proposed new branch of algorithms for the tasks of MOS based on graph signal processing concepts. The pipeline of the algorithm involves segmentation, background initialization, features extraction for the representation of the nodes in a graph, construction of the graph, and finally semi-supervised learning algorithms

inspired by the theory of the graph signals reconstruction. In the same way, several theoretical insights about the sample complexity and the graph signals reconstruction are explored in this chapter. More specifically, Corollary 3.3.1 is introduced, showing a bound for the sample complexity given a smoothness prior assumption. The proposed algorithms are evaluated on four publicly available MOS datasets. Through an extensive series of experiments, the proposed algorithms have consistently outperformed existing state-of-the-art methods by a significant margin.

This chapter opens several future research directions in computer vision and machine learning. The first important direction is to further explore a generalized theory of graph signal processing in the field of MOS. The graph signals can be extended to fuzzy concepts leading to a richer representation of moving and static objects. The second direction is to explore the graph signal processing concepts applied in bounding boxes for applications such as multi-object tracking. Another important direction is to study an inductive learning framework, which aims to address the problems of real-time processing [136] for MOS. Further questions in these directions are: how can one use the structure of certain datasets to improve the generalization of state-of-the-art deep learning methods? How can one design an algorithm to train a neural network capable of learning from the labels and the structure of a dataset? What is the relationship between the sampling of graph signals and the problems in video analysis? Perhaps, the concepts of graph signal processing, such as active semi-supervised learning and graph convolutional networks, could lead to new developments in the field of computer vision and end-to-end architectures for video analysis with semi-supervised learning.

Chapter 4 addresses the problem of training a neural network capable of learning from the labels and the structure of a dataset for MOS. Similarly, Chapter 5 focuses on the problem of using GNNs in the problem of semantic segmentation.

Chapter 4

Graph Convolutional Networks for Moving Object Segmentation

4.1 Introduction

We introduced GraphMOS in Chapter 3 for the MOS problem. GraphMOS presents several advantages regarding previous supervised and unsupervised methods for MOS. For example, GraphMOS requires less labeled information than supervised models, and it performs better than unsupervised methods. However, the addition of new videos or nodes to the problem requires solving again the optimization problem, which makes GraphMOS unsuitable for practical applications.

In this chapter, we propose a novel semi-supervised algorithm dubbed GraphMOS Network (GraphMOS-Net) based on Graph Convolutional Networks (GCNs) [33] for the MOS problem. Similar to GraphMOS in Chapter 3, GraphMOS-Net models the instances in videos as nodes embedded in a graph. In this chapter, the instances are obtained with a Cascade Mask R-CNN [105]. The representation of the nodes is obtained with background initialization, optical flow, intensity, and texture features [109, 110]. Moreover, the nodes are associated either with the class moving or static object using ground-truth information. Finally, a GCN is trained with a small percentage of labeled nodes to perform a semi-supervised learning classification [33] with an unseen scheme [14]. After training, GraphMOS-Net can simply evaluate

the trained GCN architecture if new videos are added to the problem. Therefore, GraphMOS-Net overcomes the main limitation of GraphMOS that requires solving the optimization problem (Sobolev or TV minimization in (3.3) and (3.9)) again if new nodes are added to the dataset. GraphMOS-Net outperforms some state-of-the-art methods in several challenges of the CDNet2014 [10] and UCSD background subtraction [11] datasets. The main contributions of this chapter are summarized as follows:

- We pose the problem of MOS as a binary classification problem on the graph where each node is classified into two distinct components including background and foreground using GCNs.
- We perform rigorous experiments using the CDNet2014 dataset [10] for MOS. Our results demonstrate that GraphMOS-Net uses a limited amount of labeled data and outperforms some state-of-the-art classical and deep learning methods.

4.2 Moving Object Segmentation and Graph Convolutional Networks

GraphMOS-Net consists of several components including instance segmentation, background model initialization, features extraction, graph construction, and GCN training, as shown in Fig. 4-1. GraphMOS-Net is summarized as follows: a) we compute the instance segmentation mask of the input sequence using the Mask R-CNN method [20]; b) the instances and background model are used to derive the features to be used in the graph construction, we use texture features, motion features using optical flow estimation [109], and intensity features; c) we construct the graph where a node is assigned to each object instance represented by the group of the aforementioned features; and d) we train a GCN to classify the nodes into either background or foreground. The different components of GraphMOS-Net will be described next.

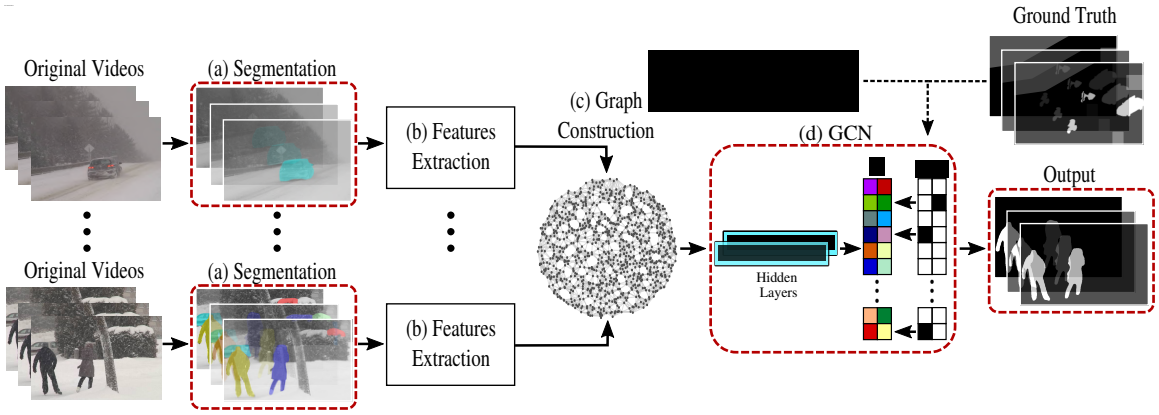

Figure 4-1: GraphMOS-Net uses background initialization and instance segmentation. Each instance represents a node in a graph using motion, intensity, and texture features. Finally, a GCN classifies if each node is a moving or static object with an unseen scheme.

4.2.1 Segmentation, Feature Extraction, and Graph Construction

The first stages of GraphMOS-Net are similar to those of GraphMOS in Chapter 3. For segmentation, we use a Cascade Mask R-CNN [105] with ResNeSt of 200 layers [107]. The feature extraction uses the handcrafted features of GraphMOS as explained in Section 3.3.2 and Appendix B, *i.e.*, no deep features are considered in this chapter. Finally, the construction of the graph is done with the k-NN method as in Section 3.3.3 with $k = 30$.

4.2.2 Graph Semi-supervised Learning Algorithm

GraphMOS algorithm in Chapter 3 relies on the sampling and reconstruction of graph signals [28,49,52,75]. As a consequence, we defined the sampled nodes (or training set in our case) in Chapter 3 as a subset of nodes $\mathcal{S} \subset \mathcal{V}$ with $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$, where $m = |\mathcal{S}| \leq N$ is the number of sampled nodes. GraphMOS applies a variational or TV method in graphs [51] to solve the semi-supervised learning problem. The application of these variational methods requires that all inferred nodes are present at the time of solving the optimization problem. GraphMOS-Net avoids this issue by training a GCN, and we refer to the set $\mathcal{S}$ as the training set in this chapter.

The layer-wise propagation rule of GraphMOS-Net, inspired from Kipf and Welling [33] method, is given as follows:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{4.1}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of G with added self-connections[1], $\mathbf{I}$ is the identity matrix, $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$, $\mathbf{W}^{(l)}$ is the matrix of trainable weights in layer l, $\sigma(\cdot)$ denotes an activation function, and $\mathbf{H}^{(l)}$ is the matrix of activations in layer l such that $\mathbf{H}^{(0)} = \mathbf{X}$ is the input matrix denoting the representation of the nodes. The propagation rule in (4.1) is motivated by the first-order approximation of localized spectral filters on graphs [31,80]. For further details, the readers are referred to [33].

GraphMOS-Net uses a GCN with one hidden layer to solve the semi-supervised learning problem. The forward of our model is computed using the propagation rule in (4.1) as follows:

$$\bar{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A}) = \text{softmax}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\text{ReLU}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}\right), \tag{4.2}$$

where $\bar{\mathbf{Y}}$ is the output of the GCN; $\mathbf{W}^{(0)} \in \mathbb{R}^{C\times H}$ is the weights of the first hidden layer such that C is the dimension of vector $\mathbf{x}_v$, *i.e.*, $C = 853$, and H is the number of feature maps; $\mathbf{W}^{(1)} \in \mathbb{R}^{H\times F}$ is the weights of the output layer such that F is the number of classes (*i.e.*, $F = 2$: background and foreground); $\text{ReLU}(\cdot) = \max(0, \cdot)$ is the rectified linear unit; and finally the softmax activation function is defined as $\text{softmax}(x_i) = \frac{1}{Q}\exp(x_i)$ where $Q = \sum_i \exp(x_i)$. Here the softmax function is applied row-wise. We set the number of feature maps H in the hidden layer to 16 in our model. The loss function of this GCN is defined as the cross-entropy error over all labeled nodes:

$$\mathcal{L} = -\sum_{i\in\mathcal{S}}\sum_{j=1}^{F}\mathbf{Y}(i,j)\ln\bar{\mathbf{Y}}(i,j). \tag{4.3}$$

[1]The self-connections given by $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ were introduced by Kipf and Welling [33] to avoid numerical instabilities and exploding/vanishing gradients.

4.3 Experimental Framework

4.3.1 Databases

In this chapter, we use the CDNet2014 [10] and the UCSD background subtraction [11] datasets previously introduced in Section 3.4.1. CDNet2014 contains 11 challenges including, bad weather, low frame rate, night videos, PTZ, turbulence, baseline, dynamic background, camera jitter, intermittent object motion, shadow, and thermal. On the other hand, UCSD contains 18 videos mainly composed of moving camera sequences, with 30 up to 246 frames each sequence.

4.3.2 Training, Validation, and Test Nodes

For CDNet2014, we evaluate the performance of GraphMOS-Net with a 4-fold cross-validation strategy using a video agnostic scheme (unseen videos), *i.e.*, the testing nodes should come from different videos regarding the training nodes. We grouped the videos in four folds as evenly as possible, using the same partition as in [137]. We use the 4-fold cross-validation strategy in this chapter to have a more direct and fair comparison against [137]. However, other strategies such as Monte Carlo cross-validation could be used. For each fold, we randomly take a small percentage of the nodes in the set $\mathcal{M} = \{0.1\%, 0.5\%, 5\%, 10\%\}$ for training, and 1% for validation, *i.e.*, we use a small portion of the data to train our model. For UCSD, we train our GraphMOS-Net in a subset of nodes of CDNet2014 and we test in UCSD dataset. For the training and validation sets we randomly choose nodes from the challenges baseline, dynamic background, shadow, and PTZ of CDNet2014. We use the F-measure metric as defined in Section 3.4.2 to evaluate the performance of GraphMOS-Net.

4.3.3 Experiments

For CDNet2014, GraphMOS-Net constructs a graph for the whole dataset, resulting in an undirected graph of 310289 nodes. For each partition of the dataset a percentage

of the total amount of nodes in the set $\mathcal{M}$ is used for training. For example, $\mathcal{S}$ contains 310 nodes for $0.1\% \in \mathcal{M}$ (keep in mind that usually each image contains several nodes as shown in Fig. 4-1). The training of our GCN is validated with a Monte Carlo cross-validation with 3 repetitions for each training density and split of the database. As a consequence, we train $4 \times 3 \times |\mathcal{M}| = 48$ GCNs for the validation of GraphMOS-Net.

For UCSD, GraphMOS-Net constructs a graph with the whole UCSD dataset plus the challenges baseline, dynamic background, shadow, and PTZ of CDNet2014; resulting in a graph of 104261 nodes. In this case, the test set is the whole UCSD dataset, and a small number of nodes in the set $\mathcal{M}$ is used for training. A Monte Carlo cross-validation with 5 repetitions is performed for each training density, *i.e.*, we train $5 \times |\mathcal{M}| = 20$ GCNs for UCSD dataset. For the sake of clarification, the metrics in the experiments are computed on the unseen set of nodes in each experiment. The CDNet2014 and UCSD experiments follow the guidelines for the training, validation, and testing sets of Section 4.3.2.

4.3.4 Implementation Details

The Cascade Mask R-CNN and the GCN were implemented using Pytorch and Detectron2 [107, 122]. We train the GCN with: 1) Adam optimizer [138], 2) a learning rate of 0.01, 3) weight decay of 5e−4, 4) a dropout rate of 0.5 in the first graph-convolutional layer, and 5) a maximum of 600 epochs. Similarly, we train the GCN using a learning scheduler that reduces the learning rate when the loss function has stopped improving in the validation set. Our scheduler has a reducing factor of 0.5, patience of 5 epochs, and a minimum learning rate of 1e−6 (the GCN stops the learning procedure either if we reach the maximum number of epochs or if we reach the minimum learning rate). We set the training batch to the full training set in each iteration since the whole database fits in memory.

For the comparison with GraphMOS-Net, the background subtraction algorithms MoG [89], SuBSENSE [104], and ViBe [131] for USCD were implemented with the BGSLibrary [124]; while GRASTA [132] and DECOLOR [16] were implemented using

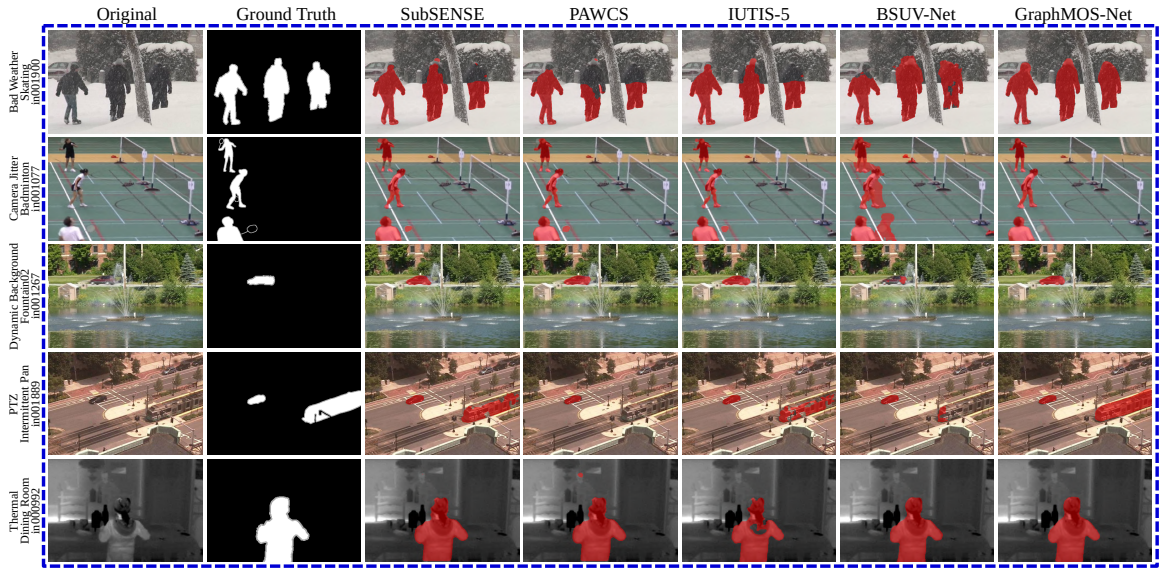

Figure 4-2: Visual results in CDNet2014

the LRSLibrary [125].

4.4 Results and Discussion

GraphMOS-Net is compared, either in CDNet2014 or UCSD, with the following state-of-the-art algorithms for MOS: SWCD [126], FTSG [127], SuBSENSE [104], PAWCS [12], WisenetMD [129], IUTIS-5 [13], SemanticBGS [130], FgSegNet v2 [67], BSUV-Net [14], GraphBGS-TV[2] [8], GraphBGS[3] [3], MoG [89], DECOLOR [16], ViBe [131], 3WD [60], GRASTA [132], ROSL [15], ADMM [133], noncvxRPCA [134], and OR1MP [135]. The results of FgSegNet v2 [67] for CDNet2014 are reported using unseen videos for the evaluation of the network, the performance comes from Tezcan *et al.* paper [14].

Fig. 4-2 shows some visual results of SubSENSE, PAWCS, IUTIS-5, BSUV-Net, and GraphMOS-Net in the CDNet2014 dataset. The sequences present challenging scenarios such as bad weather conditions, dynamic backgrounds, and PTZ cameras, among others. GraphMOS-Net presents high-quality visual results in all cases. In

[2]GraphBGS-TV is a particular instance of GraphMOS where the semi-supervised learning problem is solved with TV minimization as in (3.9).

[3]GraphBGS is a particular instance of GraphMOS where the segmentation algorithm is a Cascade Mask R-CNN.

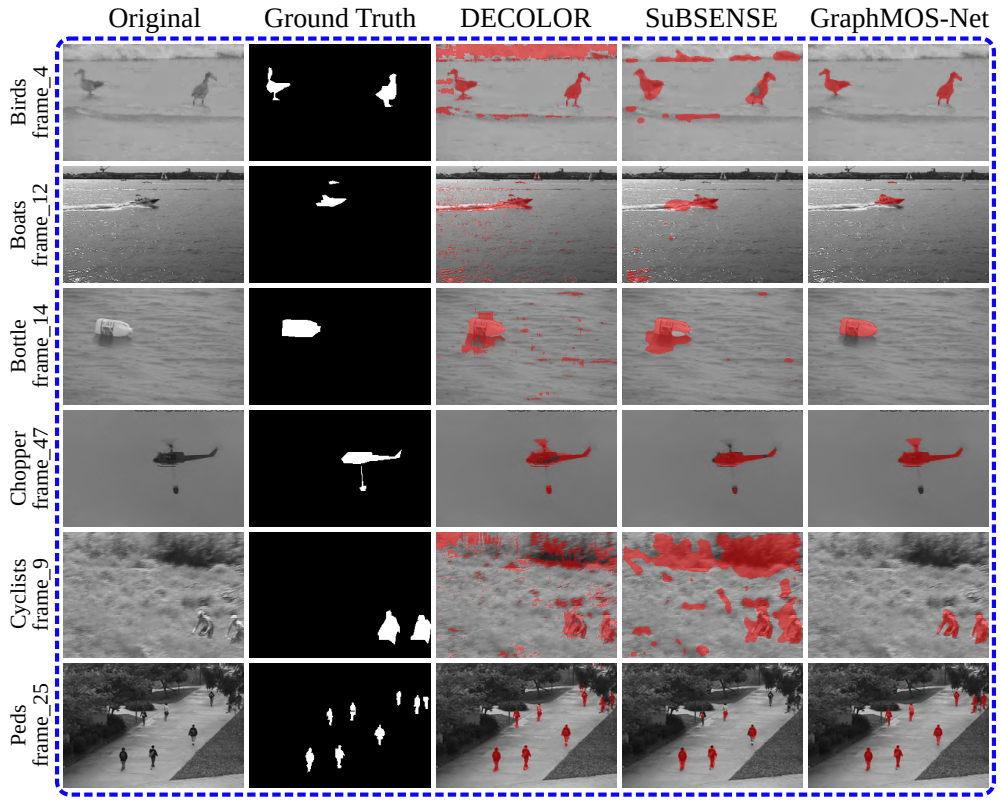

Figure 4-3: Visual results in UCSD

the same way, Fig. 4-3 shows visual results of GraphMOS-Net on UCSD compared with DECOLOR and SuBSENSE. The visual results of GraphMOS-Net indicate that GCNs are better than previous unsupervised and supervised algorithms used for MOS in some challenges. The improvement of GraphMOS-Net with respect to GraphBGS is because of the superior modeling capacity of GCNs compared with previous semi-supervised learning methods.

Tables 4.1 and 4.2 show the comparison of GraphMOS-Net with several state-of-the-art methods in CDNet2014 and UCSD, respectively. The results of GraphMOS-Net and GraphBGS are computed with the best F-measures in each sequence from the experiments (both algorithms have the same instance segmentation method). For CDNet2014 in Table 4.1, GraphMOS-Net shows the best results in the challenges baseline, dynamic background, PTZ, and shadow. The results also show that our algorithm has an overall increment in performance with respect to GraphBGS. For UCSD in Table 4.2, GraphMOS-Net shows either the best or the second-best results

Table 4.1: Comparisons of average F-measure over nine challenges of CDNet2014. GraphMOS-Net is compared with unsupervised and supervised algorithms in MOS.

Challenge	SWCD	FTSG	SuBSENSE	PAWCS	WisenetMD	IUTIS-5	SemanticBGS	FgSegNet v2	BSUV-Net	GraphBGS-TV	GraphBGS	GraphMOS-Net
Bad Weather	0.8233	0.8228	0.8619	0.8152	0.8616	0.8248	0.8260	0.7952	0.8713	0.8072	**0.9085**	**0.9142**
Baseline	0.9214	0.9330	0.9503	0.9397	0.9487	0.9567	0.9604	0.6926	**0.9693**	0.9432	0.9535	**0.9627**
Camera Jitter	0.7411	0.7513	0.8152	0.8137	0.8228	0.8332	0.8388	0.4266	0.7743	0.7191	**0.8826**	**0.8980**
Dynamic-B	0.8645	0.8792	0.8177	**0.8938**	0.8376	0.8902	**0.9489**	0.3634	0.7967	0.7365	0.8353	0.8849
I-O Motion	0.7092	**0.7891**	0.6569	0.7764	0.7264	0.7296	**0.7878**	0.2002	0.7499	0.4199	0.5036	0.7028
Low-F rate	0.7374	0.6259	0.6445	0.6588	0.6404	**0.7743**	**0.7888**	0.2482	0.6797	0.5103	0.6022	0.6945
PTZ	0.4545	0.3241	0.3476	0.4615	0.3367	0.4282	0.5673	0.3503	0.6282	**0.8014**	0.7993	**0.8329**
Shadow	0.8779	0.8832	0.8986	0.8913	0.8984	0.9084	0.9478	0.5295	0.9233	0.9658	**0.9712**	**0.9753**
Thermal	0.8581	0.7768	0.8171	0.8324	0.8152	0.8303	0.8219	0.6038	0.8581	0.7305	**0.8594**	**0.8647**
Overall	0.7764	0.7539	0.7566	0.7870	0.7653	0.7923	**0.8320**	0.4158	0.8056	0.7371	0.8128	**0.8589**

across almost all the videos of UCSD. In the overall performance, GraphMOS-Net comes second, lagging slightly behind GraphBGS. We believe it is a consequence of increasing the representational power when solving the semi-supervised learning problem with GCNs leading to a slightly worse generalization. Similarly, GraphMOS-Net outperforms subspace learning methods such as DECOLOR and GRASTA. Subspace learning methods generally have problems when dealing with moving camera sequences.

We can also notice the degradation of the deep learning method FgSegNet v2 compared to the original results [67] when tested on unseen videos in Table 4.1. Similarly, the performance of BSUV-Net drops when applied on UCSD, these results could suggest a weakness of deep learning methods in generalization for MOS (keep in mind that BSUV-Net uses the output of a fully CNN trained in ADE20K dataset [139] as input of the network). In Table 4.2, we evaluated the BSUV-Net model provided by the authors of [14] that was trained in the CDNet2014. In our case, we trained GraphMOS-Net in a small subset of nodes of CDNet2014 including the challenges baseline, dynamic background, shadow, and PTZ. This suggests that GraphMOS-Net generalizes better than BSUV-Net when applied on a different dataset from which it was trained. For the sake of clarification, we did not use the ground truth of UCSD in the training procedure. GraphMOS-Net benefits from the higher modeling capacity of GCNs by improving upon the GraphBGS method as shown in Tables 4.1 and 4.2.

Table 4.2: Comparison of F-measure results over the videos of UCSD background subtraction dataset.

Sequence	MoG	DECOLOR	ViBe	3WD	GRASTA	SuBSENSE	ROSL	ADMM	noncvxRPCA	OR1MP	BSUV-Net	BSUV-Net+	GraphBGS	GraphMOS-Net
Birds	0.1427	0.1457	0.3354	0.1308	0.1320	0.4832	0.1478	0.0227	0.1432	0.1394	0.3314	0.2625	**0.7495**	**0.7391**
Boats	0.0881	0.2179	0.1854	0.1576	0.0678	0.4550	0.1637	0.1212	0.1380	0.1100	0.4586	0.6621	**0.7765**	**0.8007**
Bottle	0.1856	0.4765	0.4512	0.1364	0.1159	0.6570	0.2069	0.6589	0.1974	0.1795	**0.8528**	0.5039	**0.8741**	**0.8741**
Chopper	0.3237	0.6214	0.4930	0.3171	0.0842	0.6723	0.2920	0.1250	0.3103	0.2653	0.2805	0.3020	**0.7766**	**0.7844**
Cyclists	0.0915	0.2224	0.1211	0.1003	0.1243	0.1445	0.1366	0.1093	0.1317	0.1242	0.0051	**0.4138**	**0.7417**	**0.7479**
Flock	0.2706	0.2943	0.2306	0.2007	0.1612	0.2492	**0.3409**	0.1088	0.3220	0.2605	0.1160	0.0025	**0.5903**	**0.5871**
Freeway	0.2622	**0.5229**	0.4002	0.5028	0.0814	**0.5518**	0.3875	0.0816	0.3126	0.1549	0.4780	0.1185	0.3719	0.3516
Hockey	0.3867	0.3449	0.4195	0.2789	0.3149	0.3611	0.4106	0.2981	0.3411	0.4296	0.6460	**0.6908**	**0.7664**	**0.7804**
Jump	0.2679	0.3135	0.2636	0.2481	0.4175	0.2295	0.4198	0.0609	0.3180	0.3073	0.5491	**0.8697**	**0.7734**	0.7602
Landing	0.0335	0.0640	0.0433	0.0457	0.0414	0.0026	0.0506	**0.0826**	0.0480	0.0442	0.0021	0.0012	**0.2452**	**0.1840**
Ocean	0.1113	0.1315	0.1648	0.2055	0.1144	0.2533	0.1422	0.1809	0.1274	0.1252	0.4117	**0.5335**	**0.8593**	**0.8593**
Peds	0.3731	**0.7942**	0.5257	0.7536	0.4653	0.5154	0.7418	0.6667	0.4333	0.4297	0.6958	0.6738	**0.8518**	**0.8512**
Skiing	0.2038	**0.3473**	0.1441	0.1981	0.0927	0.2482	0.1942	0.0519	0.1812	0.1791	0.0841	0.0602	**0.5963**	**0.5953**
Surf	0.0489	0.0647	0.0462	0.0579	0.0523	0.0467	0.0453	0.0162	0.0325	0.0317	**0.0884**	0	**0.5851**	**0.6139**
Surfers	0.0542	0.1959	0.1189	0.0962	0.0742	0.1393	0.1184	0.1950	0.1083	0.1044	0.2612	**0.4776**	**0.6719**	**0.6611**
Trafic	0.2188	**0.2732**	0.1445	0.2032	0.0368	0.1165	0.1042	0.1044	0.0949	0.0882	0	0	**0.5722**	**0.5722**
Overall	0.1914	0.3144	0.2555	0.2271	0.1485	0.3203	0.2439	0.1803	0.2025	0.1858	0.3288	**0.3483**	**0.6751**	**0.6727**

4.5 Conclusions

This chapter introduces a new algorithm for MOS based on GCN. The pipeline of the method involves a Cascade Mask R-CNN to get the object instances from the videos; a graph construction with k-nearest neighbors, where nodes are associated to object instances, and represented with feature vector encompassing optical flow, intensity and texture descriptors. We use a compact GCN model trained in a semi-supervised setting. Our algorithm outperforms previous state-of-the-art unsupervised, semi-supervised, and supervised learning algorithms for MOS in several challenges of the CDNet2014 and UCSD datasets.

This chapter opens some avenues for future research. One interesting topic is the study of an inductive framework [32] for GraphMOS-Net in the context of MOS. Similarly, other GNN architectures could be explored like Graph Attention Networks (GAT) [34] and GraphSAGE [32], among others. Chapter 5 further studies the topic of GNNs in computer vision, but in this case, we will explore the problem of weakly supervised semantic segmentation using HyperGraph Convolutional Networks (HyperGCNs).

Chapter 5

Hypergraph Convolutional Networks for Semantic Segmentation

5.1 Introduction

We introduced GraphMOS-Net in Chapter 4. GraphMOS-Net deals with the limitation of the former GraphMOS from Chapter 3, where the semi-supervised learning problem should be solved again when new nodes are added to the optimization function. In this chapter, we propose a new graph-based learning algorithm for the task of Weakly-supervised Semantic Segmentation (WSS).

Semantic segmentation is an important task in computer vision with multiple applications in image, 3D, and video processing [1, 38, 140]. The main objective of semantic segmentation is to classify all the pixels in the images into some predefined classes. Deep learning models have dominated the study of semantic segmentation in recent years [19, 141, 142]. However, these deep learning methods are usually very complex models containing millions of learnable parameters, and thus they require a lot of densely annotated images to perform well [1, 2].

Currently, there is an increasing interest in weakly supervised learning [143], where the predictions are obtained with a limited amount of labels. As a result, several studies have proposed WSS methods [144–147], where graphical models have played a central role. In particular, GCNs have been widely explored in WSS, reaching

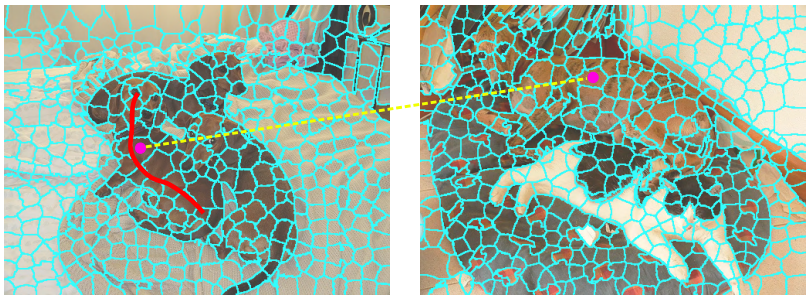

Figure 5-1: The idea of HyperGCN-WSS is to rely both on the spatial and structural information in the datasets.

state-of-the-art performances [146, 147]. However, these methods have focused on constructing graphs from individual images using spatial information. Therefore, these models waste crucial information that can be obtained from other images in the dataset.

In the current work, we propose a new algorithm named HyperGraph Convolutional Networks for Weakly-supervised Semantic Segmentation (HyperGCN-WSS), using scribbles and clicks as weak signals. The key idea of our algorithm is to rely on spatial information as in [146, 147], as well as on structural information that can be captured from other instances in the dataset. Our algorithm uses HyperGCNs [148] to capture such spatial-structural information. HyperGCN-WSS is composed of 1) SLIC superpixel segmentation [17] for node representation, 2) VGG-16 for feature extraction [149], 3) spatial and k-NN graph construction, 4) HyperGCN [148] to generate pseudo-labels, and 5) DeepLabV3+ [150] for semantic segmentation using the pseudo-labels as the target image. Fig. 5-1 shows the motivation of HyperGCN-WSS, where one labeled superpixel (with a scribble) is connected to another non-labeled superpixel in the dataset, allowing the propagation of information between different instances in the dataset. For clicks, we have some pixels with the ground-truth information. HyperGCN-WSS is evaluated in the PASCAL VOC 2012 dataset [151] for semantic segmentation using scribbles and clicks as weak signals. Our algorithm shows competitive performance against previous methods.

The main contributions of this chapter are presented as follows: 1) we propose a new algorithm for WSS, 2) we show that using HyperGCNs is better than using

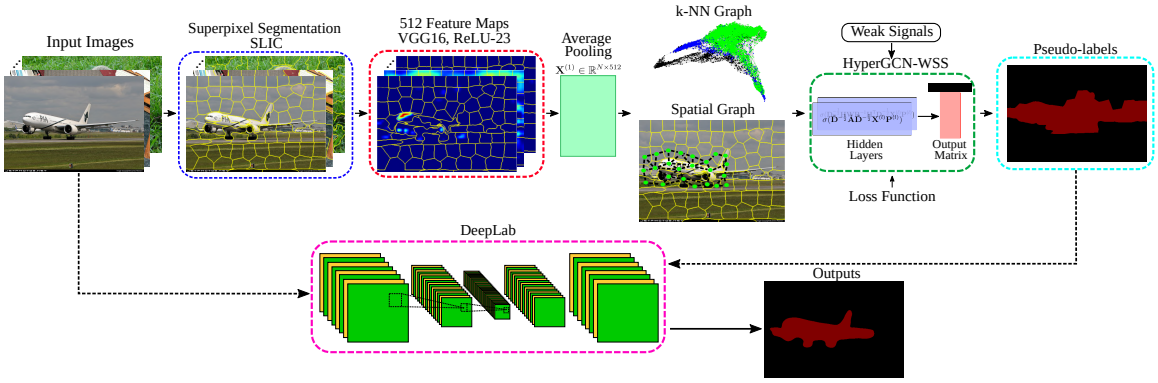

Figure 5-2: The pipeline of HyperGCN-WSS. Our algorithm uses SLIC superpixel segmentation, VGG16 feature extraction, average pooling, spatial and k-NN graph construction, a specialized HyperGCN architecture, and a DeepLabV3+ model.

GCNs alone for WSS, and 3) we evaluate our algorithm with two types of weak signals: scribbles, and clicks, showing competitive performance against some previous methods.

The rest of this chapter is organized as follows. Section 5.2 explains HyperGCN-WSS. Section 5.3 introduces the experiments and results. Finally, Section 5.4 presents the conclusions.

5.2 Proposed Method

Fig. 5-2 shows the pipeline of HyperGCN-WSS, where we have superpixel segmentation, feature extraction, hypergraph construction, HyperGCN, and a DeepLab model for semantic segmentation.

5.2.1 Preliminaries

A simple graph $G = (\mathcal{V}, \mathcal{E})$ is a mathematical entity where we have a set of nodes $\mathcal{V} \in \{1, \ldots, N\}$ and a set of edges $\mathcal{E} = \{(i, j)\}$ as explained in Section 2. Notice that $\mathbf{A}$ (the adjacency matrix) can only represent edges that connect two nodes. A hypergraph $G_h = (\mathcal{V}, \mathcal{E})$ is a generalization of simple graphs G, where the edges can connect multiple nodes. Let $\mathbf{W}_e \in \mathbb{R}^{M \times M}$ be the diagonal matrix of hyperedge weights, where $\mathbf{W}_e(e, e)$ is the weight of the eth hyperedge and $M = |\mathcal{E}|$. Let $\mathbf{H}_i \in$

$\{0,1\}^{N\times M}$ be the incidence matrix of G_h such that $\mathbf{H}_i(i,e) = 1$ if the ith node is incident to the edge e, and 0 otherwise, *i.e.*, $\mathbf{H}_i(i,e) = 1$ if the node i is connected by the edge e. Let $\mathbf{D}_h \in \mathbb{R}^{N\times N}$ be the diagonal matrix of node degree, where $\mathbf{D}_h(i,i) = \sum_{e=1}^{M} \mathbf{W}_e(e,e)\mathbf{H}_i(i,e)$. Finally, let $\mathbf{B} \in \mathbb{R}^{M\times M}$ be the diagonal matrix of hyperedge degree, where $\mathbf{B}(e,e) = \sum_{i=1}^{N} \mathbf{H}_i(i,e)$. In the current chapter, we use the hypergraphs to represent two kinds of relationships in the nodes: 1) the spatial relationships of the nodes on each image and 2) the relationship of nodes from different images in the dataset.

5.2.2 Nodes Representation and Graph Construction

We use the SLIC superpixel segmentation [17] method to represent the nodes in the graph G (or G_h). Furthermore, the input feature description of each node is obtained with some pre-trained CNN. In the current chapter, we use the outputs of the 10th ReLU layer of the VGG16 [149] (the 23th layer of the network), which contains 512 features maps. Additionally, an average pooling is performed on the superpixel regions of each feature map to obtain the feature representation, *i.e.*, each node is represented with a 512-dimensional vector.

In the current chapter, we construct two types of graphs: 1) spatial graphs in the superpixels of each image, and 2) k-NN graphs with k = 10 on some embedding space. Let α be the number of images in the dataset, let ξ be the number of superpixels for SLIC, and let μ be the maximum number of nodes we allow for each graph ($\mu = 40000$ in the experiments). Therefore, we construct $\tau = \lfloor \frac{\alpha\times\xi}{\mu} \rceil$ graphs, where we have $\gamma = \lfloor \frac{\alpha}{\tau} \rceil$ images per graph. For the spatial graphs, we connect all the nodes that are in the neighborhood of each superpixel as shown in Fig. 5-2. Therefore, we create a block diagonal matrix with the γ adjacency matrices of each image, *i.e.*, we have γ unconnected subgraphs for each spatial graph. For the k-NN graph, we use an embedding representation to compute the Euclidean distances. For example, these embeddings can be intermediate outputs of a GCN or a HyperGCN. The weights of the edges for the spatial and k-NN graphs are given by the Gaussian function $\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/\sigma^2\right)$, where $\mathbf{x}_i$ is the embedding (or feature representation) of the

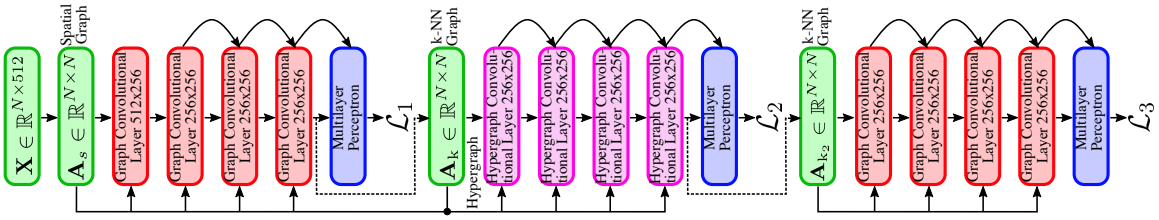

Figure 5-3: HyperGCN-WSS architecture with skip connections, as well as several graph and hypergraph convolutional layers. $\mathbf{X}$ is the matrix of features from VGG16. HyperGCN-WSS is trained in three stages, where we have three loss functions $\mathcal{L}_i$.

ith node, and σ is the standard deviation given by $\sigma = \frac{1}{|\mathcal{E}|}\sum_{(i,j)\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\|_2$.

5.2.3 Graph and Hypergraph Convolutional Networks

In this chapter, we use graph convolutions and hypergraph convolutions in our HyperGCN architecture. For the graph convolutions, we use the model of Kipf and Welling [33]. For the hypergraph convolutions, we use the model of Bai *et al.* [148].

The graph convolution in [33] was previously introduced in (4.1). Furthermore, the hypergraph convolution in [148] is given as follows:

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{D}_h^{-\frac{1}{2}}\mathbf{H}_i\mathbf{W}_e\mathbf{B}^{-1}\mathbf{H}_i^{\mathsf{T}}\mathbf{D}_h^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}), \tag{5.1}$$

where $\mathbf{H}^{(l)}$ is the matrix of activations in layer l (matrix of features or embeddings), $\mathbf{W}^{(l)}$ is the matrix of trainable weights in layer l, and $\sigma(\cdot)$ is an activation function.

5.2.4 HyperGCN Architecture

Fig. 5-3 shows the architecture of our HyperGCN-WSS. The input $\mathbf{X} \in \mathbb{R}^{N\times 512}$ is the matrix of features from the VGG16 network. Each Graph Convolutional Layer (GCL) contains batch normalization [152], Exponential Linear Unit (ELU) [153] as activation function, and it could have a residual connection [106] as shown in Fig. 5-3. The GCLs implement the propagation rule in (4.1). The Hypergraph Convolutional Layers (HCLs) are similar to the GCLs, but instead of using (4.1), they implement (5.1). Our architecture also contains Multi-layer Perceptrons that classify the embedding of

the GCLs or HCLs. We use intermediate embeddings as shown by the dotted lines in Fig. 5-3 to construct k-NN graphs. The first k-NN graph $\mathbf{A}_{\mathrm{k}}$ is combined with the spatial graph to create a hypergraph and the second intermediate embeddings are used to construct the k-NN graph $\mathbf{A}_{\mathrm{k}_2}$.

5.3 Experiments and Results

5.3.1 Dataset and Evaluation Metrics

HyperGCN-WSS is evaluated on PASCAL VOC 2012 [151] dataset for semantic segmentation. We also use the dataset of scribbles [145] and random clicks as weak signals. PASCAL VOC 2012 has 20 semantic classes and one background category. We use the augmented version of the PASCAL dataset provided by [154], resulting in 10582 images in the training set, and 1449 images in the validation set. In this chapter, we use the training set in [154] for training and validation, and we leave the validation set as the test set. We use the mean Intersection over Union (mIoU) metric [151] for evaluation.

5.3.2 Implementation Details

HyperGCN-WSS is implemented using PyTorch with a learning rate of 0.01 and weight decay of 5e$-$4. Each GCL or HCL has 256 hidden units and a dropout rate of 50%. Each stage of HyperGCN-WSS is trained for a maximum of 1000 epochs using Adam [138]. For scribbles, we use 5% of the scribbles for validation. For clicks, we use 1% of the clicks for validations. We train HyperGCN-WSS using a learning scheduler that reduces the learning rate when the loss function has stopped improving in the validation set. Our scheduler has a reducing factor of 0.5, patience of 25 epochs, and a minimum learning rate of 1e$-$6 (HyperGCN-WSS stops the learning procedure either if we reach the maximum number of epochs or if we reach the minimum learning rate). The final activation of the Multi-layer Perceptrons are logarithmic softmax, and we use the negative log-likelihood as loss functions.

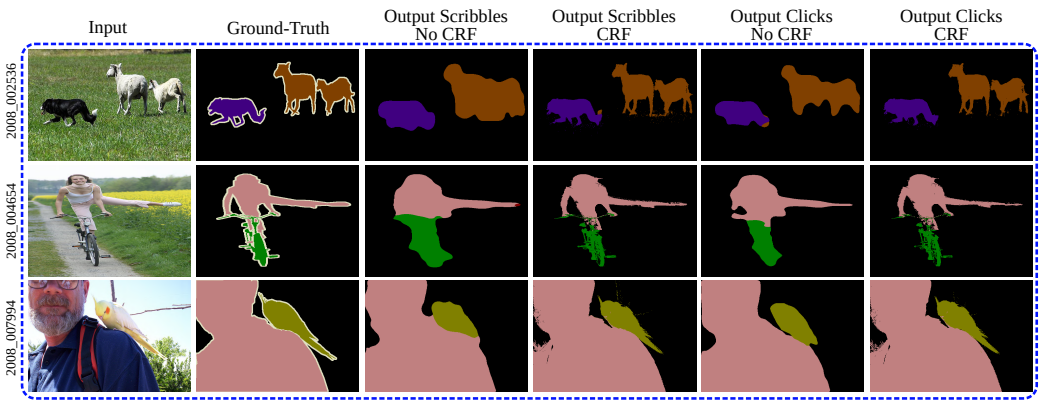

Figure 5-4: Some visual results on PASCAL VOC 2012 with our HyperGCN-WSS, using scribbles or clicks as weak signals.

5.3.3 Experiments

In this chapter, we perform experiments in 1) the dataset of scribbles [145] and 2) some random clicks that are given by a percentage of N. The percentage of clicks is given by the set $\mathcal{M} = \{\frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}\}$. For example, for $\frac{1}{32} \in \mathcal{M}$ and $\xi = 100$ number of superpixels, we have around 3.125 random clicks per image for training. Similarly, we analyze the impact of the number of superpixels ξ in the set $\mathcal{S} = \{50, 100, 200, 400, 800\}$, for both scribbles and clicks. We report the mIoU of the pseudo-labels in the training set for each loss function $\mathcal{L}_i \ \forall \ i \in \{1, 2, 3\}$ to assess the propagation of information of our algorithm after each stage. We also report the mIoU after the DeepLab training in the validation set. We do not perform an extensive search of semantic segmentation models like in [146, 155] because that is not the scope of this chapter.

5.3.4 Results and Discussions

Fig. 5-4 shows some visual results of HyperGCN-WSS before and after applying Conditional Random Field (CRF) [156]. CRFs are commonly applied as a post-processing step to improve the performance of image segmentation methods, as in [19, 145, 146]. We employ CRFs in this chapter for visualization purposes, aiming to have a visual reference regarding previous methods. Similarly, Table 5.1 shows the mIoU of the pseudo-labels after each loss function $\mathcal{L}_i \ \forall \ i \in \{1, 2, 3\}$ in the training

Table 5.1: Accuracy in mIoU (%) in the train set of PASCAL VOC after each loss function $\mathcal{L}_i \; \forall \; i \in \{1, 2, 3\}$ in our algorithm.

Weak	$\xi = 50$			$\xi = 100$			$\xi = 200$			$\xi = 400$			$\xi = 800$		
Signals	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$
Scribbles	51.85	54.49	54.23	50.04	54.31	**54.51**	46.03	49.72	50.72	39.86	44.65	45.51	32.42	37.99	39.51
Clicks $\frac{1}{32}$	41.35	42.13	42.05	41.43	42.61	42.30	40.46	43.06	42.97	39.36	43.76	**44.28**	32.56	37.91	39.87
Clicks $\frac{1}{16}$	45.59	46.78	46.70	46.06	47.98	47.72	44.81	49.17	**49.19**	39.84	45.84	46.98	33.60	38.77	41.48
Clicks $\frac{1}{8}$	50.04	51.71	51.48	51.08	**54.21**	54.15	47.94	52.98	53.18	41.33	46.48	47.98	33.16	38.76	41.24
Clicks $\frac{1}{4}$	53.44	55.76	55.52	53.63	**57.60**	57.51	49.74	54.51	55.26	41.42	46.92	48.82	34.19	37.49	40.07

Table 5.2: Accuracy of HyperGCN-WSS and previous methods in the validation set of PASCAL VOC. S: Scribbles. C: Clicks.

Method	Weak Signal	CRF	mIoU (%)
ScribbleSup [145]	S	✓	63.1
RAWKS [157]	S	✓	61.4
NormalizedCutLoss [155]	S	-	60.5
GraphNet [146]	S	-	63.3
HyperGCN-WSS (ours)	S	-	65.3
HyperGCN-WSS (ours)	C	-	**65.4**

set of PASCAL VOC with scribbles and clicks. Notice that we do not use the full labeled annotation of the training set of PASCAL VOC, so Table 5.1 shows how well the information is propagated to the other nodes in the graph. We notice that there is a gap in performance between using the GCN alone and the HyperGCN. For example, in scribbles and clicks, there is a gap of around 4% between $\mathcal{L}_1$ and $\mathcal{L}_2$. The best results for each weak signal in Table 5.1 (in **bold**) correspond to low values of superpixels ξ. The reason is that the dimensions of the features maps of the VGG16 in the 23 layer are around 8 times smaller than the original image. Therefore, having big values of ξ means smaller superpixels, which are hard to adequately represent with deep layers of CNNs. Finally, Table 5.2 shows the comparison of HyperGCN-WSS with previous methods. Our algorithm shows competitive performance against previous methods.

5.4 Conclusions

In this chapter, we introduced a new WSS algorithm called HyperGCN-WSS. Our algorithm is composed of SLIC superpixel segmentation, CNN feature extraction, hypergraph construction, a specialized HyperGCN architecture, and the DeepLabV3+

model. This new HyperGCN architecture combines graph and hypergraph convolutions. HyperGCN-WSS used spatial graphs constructed in the neighborhood of the superpixels, and k-NN graphs constructed in some embedding representation. We showed that using hypergraph convolutions is better than using graph convolutions alone. Similarly, our algorithm showed competitive performance against previous methods.

Chapters 6 and 7 delve into the machine learning part of this thesis. We focus in two problems of graph-based algorithms in machine learning: 1) Chapter 6 explores the learning of graphs for GNNs, and how to make graph convolutions more expressive without compromising scalability, and 2) Chapter 7 studies the well-known problem of over-smoothing and over-squashing in GNNs under a spectral graph theory and differential geometry approach.

Part II

Machine Learning

Chapter 6

Sparse Sobolev Graph Neural Networks

6.1 Introduction

In previous chapters we explored applications of graph-based algorithms in computer vision. In this chapter and in Chapter 7, we explore machine learning aspects of GNNs. In particular, this chapter explores topics on learning graphs from data, and making GNNs more expressive without compromising scalability.

Graph representation learning and its applications have gained significant attention in recent years. Notably, GNNs have been extensively studied [30–35]. GNNs have numerous applications such as semi-supervised learning [33, 34], point cloud semantic segmentation [37,38], prediction of individual relations in social networks [39], and modeling of proteins for drug discovery [40,41]. Similarly, other graph techniques have been recently applied to image, video, and medical image processing applications [1, 42–44].

GNNs perform well in problems of strong homophily [45]. The homophily of a graph is higher when neighbor nodes are in the same category or have similar features [45]. However, most of the state-of-the-art techniques on GNNs focus on problems where the graph topologies are already given by the problem [33, 34]. Therefore, we cannot control the homophily level of the datasets. Examples of applications

where the graphs are naturally given by the problem include citation, social, and quantum chemistry networks [158]. As a consequence, interesting tasks where the graph topologies are not readily available, remain mostly unexplored [159].

In this chapter, we study the inference of smooth graph topologies to promote homophily. We observe that smooth graphs increase the performance of baseline GNN architectures compared to the empirical k-NN method to construct graphs. Similarly, we propose a new GNN model that computes a cascade of higher-order filtering operations, allowing a more diverse set of functions on each layer. Our model is inspired by the Sobolev norm of GSP [9, 51]. This norm takes powers of the sparse adjacency matrix of a graph, converting it into a dense matrix quickly. The densification of the adjacency matrix results in memory and scalability problems in GNNs. Therefore, we modify the Sobolev norm using concepts of the Hadamard product between matrices (also known as the element-wise product) to maintain the sparsity of the adjacency matrix, and we dub our proposed architecture as Sparse Sobolev GNN (S-SobGNN). We rely on the spectral graph theory [78] and the Schur product theorem [160] to explain some properties of our filtering operation and show that this new sparse Sobolev norm satisfies the properties of vector norms. In our proposed architecture, we employ a linear combination layer at the end of each cascade of filters to select the best power functions. S-SobGNN shows good performance improvements in several applications in the presence of wide instead of deep networks. Therefore, we improve expressiveness by 1) learning smooth graphs that promote homophily and 2) computing a more diverse set of sparse graph-convolutional functions.

We evaluate the generalization capabilities of S-SobGNN in many applications including instance and node classification. First, we test S-SobGNN in semi-supervised learning tasks in several domains including, tissue phenotyping in colon cancer histology images [161], text classification of news [162], activity recognition with sensors [163], and recognition of spoken letters [164]. Finally, we test S-SobGNN in the benchmark Pattern and Cluster datasets for node classification [165]. Our algorithm outperforms many state-of-the-art methods in semi-supervised learning, node classification, and MOS in graphs.

The main contributions of this chapter are summarized as follows:

1. We demonstrate that the smooth graphs show significant performance improvement compared to k-NN graphs for GNNs by promoting homophily.
2. We propose a new GNN architecture that computes a cascade of higher-order filters inspired by the Sobolev norm of GSP.
3. Several mathematical insights of S-SobGNN are introduced using the spectral graph theory [78] and the Schur product theorem [160].
4. We perform extensive experimental evaluations on six publicly available benchmark datasets and compared S-SobGNN with 14 state-of-the-art GNN architectures with rigorous analysis. Our algorithm shows competitive results against several methods.

The rest of this chapter is organized as follows. Section 6.2 presents the related work, Section 6.3 introduces the basic concepts and the background on learning graphs from data. Section 6.4 explains the proposed GNN model. Section 6.5 presents the experimental framework and results. Finally, Section 6.6 shows the conclusions.

6.2 Related Work

6.2.1 Inference of Graph Topology

Many studies in GNNs assume that the graph topology is naturally given by the problem. For instance, nodes and edges in citation networks correspond to the documents and citation links [166]. Nevertheless, learning a graph topology from data is essential when a natural choice is not readily available. Furthermore, many interesting problems, where the graph topology is not available, have been little studied in GNNs [159].

Learning (or inferring) graphs from data is an emerging field of study in GSP [167]. The inference of a meaningful graph is crucial for the further processing steps. In the literature, the k-NN plus Gaussian kernel method is a typical procedure to construct a graph [29]. However, some authors have proposed more meaningful methods. For

example, Kalofolias showed in 2016 that the typical k-NN procedure is a particular case of a more general method of learning smooth graphs from data, where the smoothness is measured with a function of the Laplacian [168]. Later, other studies, including [169] and [170], have also used the intuition of learning smooth graphs using the Laplacian concepts with different optimization procedures. Interested readers can explore more details about learning graphs from data in a recent survey [167]. In our proposed algorithm, we explore the inference of smooth graphs in GNNs [168]. We show that the learned smooth graphs improve the performance of S-SobGNN and other state-of-the-art methods by promoting homophily.

6.2.2 Graph Neural Networks

The study of graphs is a well-established field in deep learning and mathematics [78, 171]. With its emergence in 2014 and motivated by the success of CNNs in regular-structured data like images [36], Bruna *et al.* proposed the first modern GNN by extending the convolutional operator of CNNs to graphs [30]. Afterward, Defferrard *et al.* used the concepts of GSP to propose localized spectral filtering [31] in 2016. Subsequently, Kipf and Welling approximated the filtering operation of spectral filtering to perform efficient convolution operations on graphs [33] in 2017. Other major GNNs works include the study of inductive representation learning on graphs [32] in 2017 and the development of GATs [34] in 2018. A complete review of GNNs can also be explored in the survey [35].

In this chapter, we introduce a new GNN architecture inspired by the Sobolev norm of GSP as introduced in Definition 3.3.1. This norm adds a matrix $\epsilon\mathbf{I}$ to the Laplacian, where $\epsilon \in \mathbb{R}_+$. Then, the Sobolev norm takes powers of the modified Laplacian matrix. The notion of taking powers of the matrix that represents the graph has been previously explored in GNNs [172,173]; however, the powers of the adjacency matrix (or Laplacian) quickly become a dense matrix, resulting in scalability and memory problems. As a result, we modify the Sobolev norm using the Hadamard product between matrices to keep the sparsity level of the adjacency matrix.

6.3 Learning Graphs from Data

6.3.1 Preliminaries

Given a set of classes for the vertices $\mathcal{V}$, the homophily captures the property of a node to have the same class as its neighbors. Pei *et al.* [174] proposed a metric to measure the level of homophily in a graph as follows:

$$H(G) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \frac{|\mathcal{N}_i^s|}{|\mathcal{N}_i|}, \tag{6.1}$$

where $\mathcal{N}_i$ is the set of vertices connected to the ith node by only one edge, and $\mathcal{N}_i^s$ is the set of nodes that share the same labels as the ith node. $H(G) \to 1$ when we have graphs with strong homophily, while $H(G) \to 0$ for graphs with strong heterophily. As a consequence, low homophily means high heterophily and vice versa.

6.3.2 Inferring Smooth Graphs

We use the smoothness as a prior assumption to infer the graphs in this chapter. However, we need to extend the definition of the Laplacian quadratic form in (2.5) to multiple-dimensional signals to learn graphs from data. Let $\mathbf{X} \in \mathbb{R}^{N \times M}$ be the data matrix of N instances, where each instance is an M-dimensional vector and $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\mathsf{T}$. $\mathbf{x}_v \in \mathbb{R}^M$ could be any meaningful M-dimensional feature representation of the vth instance in the problem we are approaching. For example, in image classification, this representation could be the features from some CNN. We can also think of $\mathbf{X}$ as a set of M one-dimensional graph signals such that $\mathbf{X} = [\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_M]$, where $\mathbf{x}'_i \in \mathbb{R}^N$ is the ith column vector of $\mathbf{X}$. The extension of the graph Laplacian quadratic form in Eq. (2.5) to multiple dimensions for $\mathbf{X}$ in G is given as follows [168]:

$$\frac{1}{2} \sum_{i,j} \mathbf{A}(i,j) \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \mathrm{tr}(\mathbf{X}^\mathsf{T} \mathbf{L} \mathbf{X}). \tag{6.2}$$

When we infer graphs based on the minimization of Eq. (6.2), we end up with the smoothest representation in the columns of $\mathbf{X}$. As a consequence, we refer to Eq. (6.2) as the smoothness of $\mathbf{X}$. Eq. (6.2) has been used in several studies to infer the topology of G [167–169]. Notice that $\text{tr}(\mathbf{X}^\intercal \mathbf{L}\mathbf{X}) = \sum_{i=1}^{M} \mathbf{x}_i'^\intercal \mathbf{L}\mathbf{x}_i'$ in Eq. (6.2), *i.e.*, the smoothness of the data matrix $\mathbf{X}$ is the sum of all graph Laplacian quadratic forms of each graph signal $\mathbf{x}_i' \; \forall \; i \in \{1, \ldots, M\}$. Furthermore, let $\mathbf{Z} \in \mathbb{R}_+^{N \times N}$ be the pairwise distances matrix of $\mathbf{X}$ such that $\mathbf{Z}(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \; \forall \; 1 \leq i,j \leq N$. We thus can rewrite (6.2) as follows [168]:

$$\text{tr}(\mathbf{X}^\intercal \mathbf{L}\mathbf{X}) = \frac{1}{2}\,\text{tr}(\mathbf{A}\mathbf{Z}) = \frac{1}{2}\|\mathbf{A} \circ \mathbf{Z}\|_{1,1}. \tag{6.3}$$

Kalofolias [168] proposed to infer a smooth topology of graphs using the following optimization function:

$$\underset{\mathbf{A} \in \mathcal{A}}{\arg\min} \, \|\mathbf{A} \circ \mathbf{Z}\|_{1,1} - \gamma \mathbf{1}^\intercal \log(\mathbf{A}\mathbf{1}) + \frac{\beta}{2}\|\mathbf{A}\|_F^2, \tag{6.4}$$

where $\mathcal{A}$ is the set of valid adjacency matrices (symmetric, with zero diagonal, and non-negative), and the parameters $\gamma > 0$ and $\beta \geq 0$ control the connectivity of the graph. The term $\|\mathbf{A} \circ \mathbf{Z}\|_{1,1}$ in Eq. (6.4) is associated with the smoothness of the data matrix, and therefore we are looking for the smoothest representation out of $\mathbf{X}$. Moreover, the second and third terms of Eq. (6.4) prevent the trivial solution and control sparsity. The optimization problem in Eq. (6.4) can be solved using primal-dual techniques [121].

6.3.3 Reducing Hyperparameters

One can relate the parameters γ and β of Eq. (6.4) with the number of k neighbors of each node as follows:

Proposition 1 (Kalofolias and Perraudin [170])**.** *Let* $\mathbf{A}^*(\mathbf{Z}, \gamma, \beta)$ *be the solution of Eq. (6.4) for the matrix of distances* $\mathbf{Z}$ *and parameters* $\gamma, \beta > 0$*. Therefore, the same solution can be achieved with parameters* $\gamma = 1$ *and* $\beta = 1$*, by multiplying* $\mathbf{Z}$ *by*

$\theta = 1/\sqrt{\gamma\beta}$ *and the resulting* $\mathbf{A}^*$ *by* $\delta = \sqrt{\gamma/\beta}$.
Proof: see [170].

In some applications, the multiplication of $\mathbf{A}^*$ by a constant δ does not change the functionality of the adjacency matrix, leading to a reduction in the number of hyperparameters. Similarly, we can find two bounds for the parameter θ to get approximately k edges per node as follows [170]:

$$\sum_{j=1}^{N} \frac{1}{N\sqrt{k\bar{\mathbf{Z}}^2(k+1,j) - \mathbf{C}(k,j)\bar{\mathbf{Z}}(k+1,i)}} < \theta_k,$$
$$\theta_k \leq \sum_{j=1}^{N} \frac{1}{N\sqrt{k\bar{\mathbf{Z}}^2(k,j) - \mathbf{C}(k,j)\bar{\mathbf{Z}}(k,j)}}, \tag{6.5}$$

where $\bar{\mathbf{Z}}$ is obtained by sorting each column of $\mathbf{Z}$ in increasing order, $\mathbf{C}(k,j) = \sum_{i=1}^{k} \bar{\mathbf{Z}}(i,j)\ \forall\ 1 \leq k,j \leq N$, and θ_k is the parameter to scale the matrix $\mathbf{Z}$ and get approximately k edges per node in $\mathbf{A}$.

In this chapter, we learn the smooth graph as follows:

$$\arg\min_{\mathbf{A}\in\mathcal{A}} \|\mathbf{A} \circ \theta_k\mathbf{Z}\|_{1,1} - \mathbf{1}^\mathsf{T}\log(\mathbf{A}\mathbf{1}) + \frac{1}{2}\|\mathbf{A}\|_F^2. \tag{6.6}$$

The solution of Eq. (6.6) is a smooth adjacency matrix $\mathbf{A}$ with approximately k edges per node, promoting homophily [45]. For example, the smooth graph in Fig. 6-1 has a clear cluster structure compared to the k-NN graph. In this chapter, we feed S-SobGNN and other state-of-the-art methods with smooth graphs learned from Eq. (6.6) and with k-NN graphs to assess their performance on graph representation learning tasks.

6.4 Sparse Sobolev Graph Neural Networks

Figure 6-1 shows an overview of our framework. Our algorithm consists of two steps 1) the inference or learning of a graph topology and 2) the proposed S-SobGNN architecture applied to semi-supervised learning or node classification.

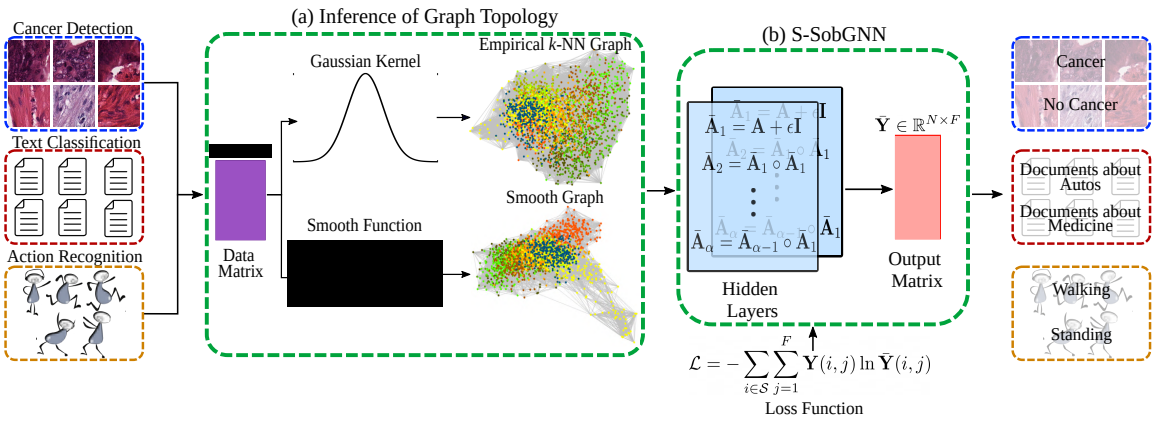

Figure 6-1: The pipeline of our S-SobGNN algorithm with k-NN or smooth-learned graphs. Our GNN can be used in a broad range of data such as images, and text, among others. However, the step of mapping the original dataset to the data matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ could be different in each case. Our framework is composed of (a) inference of the graph topology and (b) the S-SobGNN architecture. Finally, a loss function (such as the cross-entropy) is computed for the training procedure.

6.4.1 Sobolev Norm

The Sobolev norm in GSP, introduced in Definition 3.3.1, has been used as a regularization term to solve problems in 1) video processing [1, 3], 2) modeling of infectious diseases [8], and 3) interpolation of graph signals [9, 51].

When $\mathbf{L}$ is symmetric in the Sobolev norm in Definition 3.3.1, we have that $\|\mathbf{x}\|^2_{\rho,\epsilon}$ can be rewritten as follows:

$$\|\mathbf{x}\|^2_{\rho,\epsilon} = \mathbf{x}^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\rho \mathbf{x}. \tag{6.7}$$

The analysis of Eq. (6.7) is divided into two steps: 1) when $\epsilon = 0$, and 2) when $\rho = 1$. Let $\mathbf{U}$ be an orthonormal matrix $\mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}$, and for $\epsilon = 0$ in Eq. (6.7) we have:

$$\mathbf{x}^\mathsf{T}\mathbf{L}^\rho\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{U}\mathbf{\Lambda}^\rho\mathbf{U}^\mathsf{T}\mathbf{x} = \hat{\mathbf{x}}^\mathsf{T}\mathbf{\Lambda}^\rho\hat{\mathbf{x}} = \sum_{i=1}^{N} \hat{\mathbf{x}}^2(i)\lambda_i^\rho. \tag{6.8}$$

Notice that the spectral components $\hat{\mathbf{x}}(i)$ are penalized with powers of the eigenvalues λ_i^ρ of $\mathbf{L}$. Since the eigenvalues are ordered in increasing order, the higher frequencies of $\hat{\mathbf{x}}$ are penalized more than the lower frequencies when $\rho = 1$, leading to a smooth function in G. For $\rho > 1$, the GFT $\hat{\mathbf{x}}$ is penalized with a more diverse set of eigenval-

ues. We can have a similar analysis for the adjacency matrix $\mathbf{A}$ using the eigenvalue decomposition $\mathbf{A}^\rho = (\mathbf{V}\mathbf{\Sigma}\mathbf{V}^\mathsf{H})^\rho = \mathbf{V}\mathbf{\Sigma}^\rho\mathbf{V}^\mathsf{H}$, where $\mathbf{V}$ is the matrix of eigenvectors, and $\mathbf{\Sigma}$ is the matrix of eigenvalues of $\mathbf{A}$. In the case of $\mathbf{A}$, the GFT $\hat{\mathbf{x}} = \mathbf{V}^\mathsf{H}\mathbf{x}$.

For the second analysis of Eq. (6.7), for $\rho = 1$, we have:

$$\|\mathbf{x}\|^2_{\rho,\epsilon} = \mathbf{x}^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})\mathbf{x}. \tag{6.9}$$

The term $(\mathbf{L} + \epsilon\mathbf{I})$ in Eq. (6.9) is associated with a better condition number[1] than using $\mathbf{L}$ alone. Better condition numbers can be associated with faster convergence rates in gradient descent methods, as shown in [8]. For the Laplacian matrix $\mathbf{L}$, we know that $\kappa(\mathbf{L}) \to \infty$ from 3.4, *i.e.*, we have a bad-conditioned problem when relying on the Laplacian matrix alone. On the other hand, for the Sobolev term, we have that:

$$\mathbf{L} + \epsilon\mathbf{I} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathsf{T} + \epsilon\mathbf{I} = \mathbf{U}(\mathbf{\Lambda} + \epsilon\mathbf{I})\mathbf{U}^\mathsf{T}. \tag{6.10}$$

Therefore, $\lambda_{\min}(\mathbf{L} + \epsilon\mathbf{I}) = \epsilon$, *i.e.*, $\mathbf{L} + \epsilon\mathbf{I}$ is positive definite ($\mathbf{L} + \epsilon\mathbf{I} \succ 0$), and:

$$\kappa(\mathbf{L} + \epsilon\mathbf{I}) = \frac{|\lambda_{\max}(\mathbf{L} + \epsilon\mathbf{I})|}{|\lambda_{\min}(\mathbf{L} + \epsilon\mathbf{I})|} = \frac{\lambda_{\max}(\mathbf{L}) + \epsilon}{\epsilon} < \kappa(\mathbf{L}) \;\forall\; \epsilon > 0. \tag{6.11}$$

Namely, $\mathbf{L} + \epsilon\mathbf{I}$ has a better condition number than $\mathbf{L}$. It might not be evident why a better condition number could help in GNNs, where the inverses of the Laplacian or adjacency matrices are not required to perform the propagation rules. However, some authors have seen the adverse effects of bad-behaved matrices. For example, Kipf and Welling [33] used a renormalization trick such that $\mathbf{A} + \mathbf{I}$ is used in their filtering operation to avoid exploding/vanishing gradients. Similarly, Wu *et al.* [175] showed that adding the identity matrix to $\mathbf{A}$ shrinks the graph spectral domain, resulting in a low-pass-type filter.

The previous theoretical analysis shows the benefits of the Sobolev norm regarding 1) the more diverse frequency computation in Eq. (6.8), and 2) the better condition number in Eq. (6.11).

[1]The condition number $\kappa(\mathbf{L})$ associated with the square matrix $\mathbf{L}$ is a measure of how well or ill conditioned is the inversion of $\mathbf{L}$.

6.4.2 Sparse Sobolev Norm

The use of $\mathbf{L}$ or $\mathbf{A}$ in GNNs is computationally efficient because these matrices are usually sparse. Therefore, we can perform a small number of sparse matrix operations. For the Sobolev norm, the term $(\mathbf{L}+\epsilon\mathbf{I})^{\rho}$ can quickly become a dense matrix for large values of ρ, leading to scalability and memory problems. To mitigate this problem, we use the sparse Sobolev norm to keep the same sparsity level. Let us first define the theoretical aspects of the sparse Sobolev norm, and then we will show its application to GNNs.

Definition 6.4.1. *Let $\mathbf{L} \in \mathbb{R}^{N\times N}$ be the Laplacian matrix of a graph G. For fixed parameters $\epsilon \geq 0$ and $\rho \in \mathbb{N}$, the sparse Sobolev term for GNNs is introduced as the ρ Hadamard multiplications of $(\mathbf{L}+\epsilon\mathbf{I})$ (also known as the Hadamard powers) such that:*

$$(\mathbf{L}+\epsilon\mathbf{I})^{(\rho)} = (\mathbf{L}+\epsilon\mathbf{I}) \circ (\mathbf{L}+\epsilon\mathbf{I}) \circ \cdots \circ (\mathbf{L}+\epsilon\mathbf{I}). \tag{6.12}$$

For example, $(\mathbf{L}+\epsilon\mathbf{I})^{(2)} = (\mathbf{L}+\epsilon\mathbf{I}) \circ (\mathbf{L}+\epsilon\mathbf{I})$. As a result, the sparse Sobolev norm is given by:

$$\|\mathbf{x}\|_{(\rho),\epsilon} \triangleq \|(\mathbf{L}+\epsilon\mathbf{I})^{(\rho/2)}\mathbf{x}\|. \tag{6.13}$$

Let $\langle \mathbf{x}, \mathbf{y} \rangle_{(\rho),\epsilon} = \mathbf{x}^{\mathsf{T}}(\mathbf{L}+\epsilon\mathbf{I})^{(\rho)}\mathbf{y}$ be the inner product between two graph signals $\mathbf{x}$ and $\mathbf{y}$ that induces the associated sparse Sobolev norm.

Theorem 6.4.1. *The sparse Sobolev norm $\|\mathbf{x}\|_{(\rho),\epsilon} \triangleq \|(\mathbf{L}+\epsilon\mathbf{I})^{(\rho/2)}\mathbf{x}\|$ satisfies the basic properties of vector norms for $\epsilon > 0$ (for $\epsilon = 0$ we have a semi-norm).*
Proof: See Appendix D.

The sparse Sobolev term in (6.12) has the property of keeping the same sparsity level of $\mathbf{L}+\epsilon\mathbf{I}$ for any value of ρ. Notice that $(\mathbf{L}+\epsilon\mathbf{I})^{\rho}$ is equal to the sparse Sobolev term if 1) we restrict ρ to be in $\mathbb{N}$, and 2) we replace the common matrix multiplication with the Hadamard product. The theoretical properties of the Sobolev norm in (6.8) and (6.11) do not extend trivially to its sparse counterpart. However, we can develop some theoretical insights using concepts of Kronecker products and the Schur product theorem [160].

Lemma 6.4.2. *Let* $\mathbf{L}$ *be any Laplacian matrix of a graph with eigenvalue decomposition* $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}$*, we have that:*

$$\mathbf{L} \circ \mathbf{L} = \mathbf{L}^{(2)} = \mathbf{P}_N^{\mathsf{T}}(\mathbf{U} \otimes \mathbf{U})(\mathbf{\Lambda} \otimes \mathbf{\Lambda})(\mathbf{U}^{\mathsf{T}} \otimes \mathbf{U}^{\mathsf{T}})\mathbf{P}_N, \tag{6.14}$$

where $\mathbf{P}_N \in \{0,1\}^{N^2 \times N}$ *is a partial permutation matrix.*
Proof: See Appendix E.

Eq. (6.14) is a closed-form solution regarding the spectrum of the Hadamard power for $\rho = 2$. Thus, the spectrum of the Hadamard multiplication is a compressed form of the Kronecker product of the spectral components. The sparse Sobolev term we use in our S-SobGNN is given by $(\mathbf{L} + \epsilon\mathbf{I})^{(\rho)}$ so that the spectral components of the graph are changing for each value of ρ.

For the condition number of the Hadamard powers, we can use the Schur product theorem (in Appendix D, Theorem D.0.1). We know that $(\mathbf{L} + \epsilon\mathbf{I})^{(\rho)} \succ 0 \ \forall \ \epsilon > 0$ since $(\mathbf{L} + \epsilon\mathbf{I}) \succ 0 \ \forall \ \epsilon > 0$, and therefore $\kappa((\mathbf{L} + \epsilon\mathbf{I})^{(\rho)}) < \infty$. For the case of the adjacency matrix, the eigenvalues of $\mathbf{A}$ lie in the interval $[-d, d]$, where d is the maximal degree of the vertices in G (see Theorem 8.5 in [176]). Therefore, we can bound the eigenvalues of $\mathbf{A}$ into the interval $[-1, 1]$ by normalizing the adjacency matrix such that $\mathbf{A}_N = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. As a result, we know that $\mathbf{A}_N + \epsilon\mathbf{I} \succ 0 \ \forall \ \epsilon > 1$, and $(\mathbf{A}_N + \epsilon\mathbf{I})^{(\rho)} \succ 0 \ \forall \ \epsilon > 1$. Finally, we can say that the theoretical developments of the sparse Sobolev norm hold to some extent the same theoretical developments of Section 6.4.1, *i.e.*, a more diverse set of frequencies and a better condition number.

6.4.3 Graph Neural Network Architecture

Kipf and Welling [33] proposed one of the most successful yet simple GNN, called GCN. The propagation rule of GCN and the architectural details were introduced in Section 4.2.2.

In this chapter, we introduce a new filtering operation based on the sparse Sobolev

term such that our propagation rule is given as follows:

$$\mathbf{B}_{\rho}^{(l+1)} = \sigma(\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}\bar{\mathbf{A}}_{\rho}\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}\bar{\mathbf{H}}^{(l)}\mathbf{W}_{\rho}^{(l)}), \tag{6.15}$$

where $\bar{\mathbf{A}}_{\rho} = (\mathbf{A}+\epsilon\mathbf{I})^{(\rho)}$ is the ρth sparse Sobolev term of $\mathbf{A}$, $\bar{\mathbf{D}}_{\rho}$ is the degree matrix of $\bar{\mathbf{A}}_{\rho}$, and $\bar{\mathbf{H}}^{(1)} = \mathbf{X}$. Notice that $\bar{\mathbf{A}}_{\rho} = \tilde{\mathbf{A}}$ when $\epsilon = 1$, and $\rho = 1$, *i.e.*, our propagation rule is a generalization of Eq. (4.1). S-SobGNN computes a cascade of propagation rules as in Eq. (6.15) with several values of ρ in the set $\{1, 2, \ldots, \alpha\}$, and therefore a linear combination layer weights the outputs of each filter. The linear combination layer is implemented as follows:

$$\bar{\mathbf{H}}^{(l+1)} = \sum_{i=1}^{\alpha} \mu_{l,i}\mathbf{B}_{i}^{(l+1)}, \tag{6.16}$$

where $\mu_{l,i}$ is the learnable weight associated with $\mathbf{B}_{i}^{(l+1)}$. Figure 6-2 shows the basic configuration of S-SobGNN. However, we can construct more elaborate architectures based on our filtering operation. For example, we can add batch normalization, residual connections, among others. Notice that our graph convolution is efficiently computed since the term $\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}\bar{\mathbf{A}}_{\rho}\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}$ $\forall$ $\rho \in \{1, 2, \ldots, \alpha\}$ remains the same in all layers (so we can compute it offline), and also, these terms are sparse for any value of ρ (given that $\mathbf{A}$ is also sparse). S-SobGNN uses ReLU as the activation function for each filter, softmax at the end of the network, and the cross-entropy loss in Eq. (4.3) as loss function. The basic configuration of S-SobGNN is defined by the number of filters α in each layer, the parameter ϵ, the number of hidden units of each $\mathbf{W}_{\rho}^{(l)}$, and the number of layers n. When we construct weighted graphs with Gaussian kernels, the weights of the edges are in the interval $[0, 1]$. As a consequence, large values of ρ could make $\bar{\mathbf{A}}_{\rho} = \mathbf{0}$, and the diagonal elements of $\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}$ could become ∞. Similarly, large values of α make very wide architectures with a high parameter budget, so it is desirable to maintain a reasonable value for α. In the experiments, we test S-SobGNN architectures up to $\alpha = 6$.

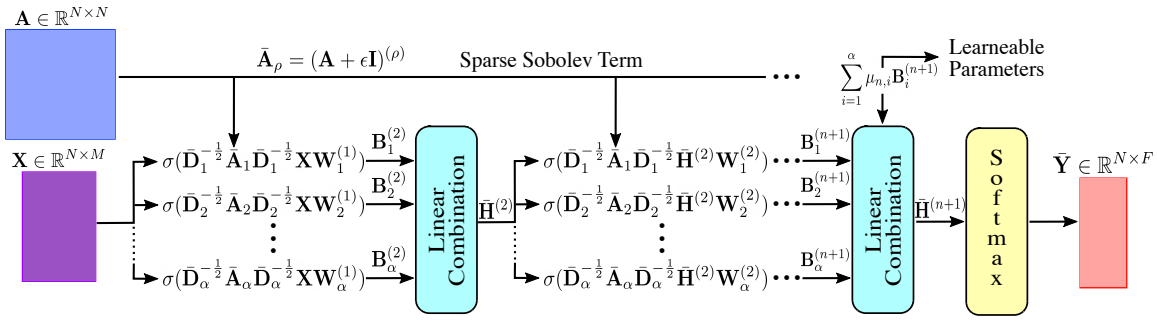

Figure 6-2: Basic configuration of our S-SobGNN architecture with n layers and α filters per layer.

6.5 Experiments and Results

The real performance of new convolutional operations in GNNs becomes illuded because authors can exhaustively fine-tune hyper-parameters and architectures in specific applications to improve accuracy. The aim to outperform other state-of-the-art methods may lead to very complex and computationally expensive models with millions of parameters [177]. However, our objective is to demonstrate a new useful filter by performance comparisons in similar data regimens while keeping a comparable amount of parameters. This section presents several experiments and results for semi-supervised learning, and node classification (the benchmarking datasets).

S-SobGNN is compared with 14 GNN architectures. S-SobGNN is compared with: Chebyshev filters (Cheby) [31], Topology Adaptive GCNs (TAGConv) [178], GCN [33], GraphSage [32], MoNet [179], GatedGCN [180], GAT [34], k-Graph Neural Network (k-GNN) [181], GIN [182], RingGNN [183], GCN via Initial residual and Identity mapping (GCNII) [184], GCN plus DropEdge (GCN+DE) [185], Graph Sampling based Inductive learning (GraphSAINT) [186], and SIGN [173].

6.5.1 Implementation Details

S-SobGNN is implemented using PyTorch. The k-NN and the learned graph methods are implemented using the GSP toolbox [123] and UNLocBox [120]. The values of k are searched in the set $\{10, 20, 30, 40\}$, and then the value θ_k in Eq. (6.6) is computed as the middle value of the interval in Eq. (6.5). The state-of-the-art GNNs

are implemented with PyTorch Geometric [187]. For the semi-supervised learning experiments, the standard implementation of S-SobGNN has 1) two layers, 2) 16 hidden units in the first convolutional layer (and the output layer contains F output units), 3) a learning rate of 0.01, 4) weight decay of 5e$-$4, and 5) a dropout rate of 0.5 in the first graph-convolutional layer. S-SobGNN and the other methods are trained with 500 epochs using Adam optimizer [138]. For the benchmarking, we add some additional features following the same experimental setting as in [165]: 1) S-SobGNN has four layers, 2) we restrict $\alpha = 2$ to keep a low parameter budget, 3) we add residual connections and batch normalization to each graph convolutional layer, and 4) we add a Multi-Layer Perceptron (MLP) at the end of our architecture.

6.5.2 Semi-supervised Learning

Datasets: We test S-SobGNN in several semi-supervised learning tasks including, tissue phenotyping in colon cancer histology images [161], text classification of news [162], activity recognition using sensors [163], and recognition of spoken letters [164]. These are semi-supervised problems because S-SobGNN uses both labeled and unlabeled data in the training procedure. We also test the smooth learned and the k-NN graph representations as inputs of the GNNs in these tasks.

Multi-class in Colorectal Cancer Histology: This cancer dataset comes from 10 anonymized tissue slides from the University Medical Center Mannheim [161]. The dataset has eight classes: tumor epithelium, simple stroma, complex stroma, immune cells, debris, normal mucosal glands, adipose tissue, and background (no tissue). This chapter does not explore the feature representation, and only uses Local Binary Patterns (LBP) [110] to represent each image. We explore two partitions of this dataset: 1) Cancer Binary (Cancer-B) considers only tumor epithelium and simple stroma, and 2) Cancer Multi-class (Cancer-M) tests all classes.

The 20 Newsgroups: The 20 Newsgroups dataset (20News) contains roughly 20000 news from 20 classes [162]. In this chapter, we use a subset of 10 categories: computer graphics, windows operative system, IBM hardware, MAC hardware, autos, motorcycles, cryptography, electronics, medicine, and space. Each news is represented with

the Term-Frequency Inverse-Document-Frequency (TFIDF) as in [188].

Human Activity Recognition (HAR): This dataset contains recordings of 30 people while they were doing activities of daily living [163]. These people were carrying waist-mounted smartphones with embedded sensors. HAR contains six classes: standing, sitting, laying down, walking, walking downstairs, and walking upstairs. Each recording is represented as in [163].

Isolated Letter Recognition: The Isolet dataset has recordings from 30 speakers of the 26 letters of the English alphabet. Isolet has 6238 samples represented as 617-dimensional feature vectors. The feature representation method is explained in [164].

Experiments

Several graphs are inferred for Cancer-B, Cancer-M, 20News, HAR, and Isolet, using a k-NN method and the algorithm presented in Section 6.3.3 (we refer to this algorithm as learned graphs in the results). For Cancer-B and Cancer-M, we compute two graphs with $N = 1250$ and $N = 5000$, respectively. For 20News, HAR, and Isolet, we infer ten graphs with $N = 3000$. The samples for each dataset are divided evenly across different classes. For each dataset, we vary the training percentage in the set $\{1\%, 2\%, \ldots, 19\%, 20\%\}$, and the rest of the samples are for testing. No validation set is used (we do not fine-tune hyperparameters in this experiment). We compare S-SobGNN against state-of-the-art GNNs using two strategies. In the first strategy, we train the state-of-the-art models with default parameters without any restriction. For the second strategy, 1) we train the state-of-the-art models with approximately the same amount of learnable parameters, 2) we do not consider any specialized layer, and 3) we use the same learning rate, number of epochs, dropout, weight decay, activation layers, and number of layers for each model. Moreover, several ablation studies analyze the values of ϵ, α, and k of S-SobGNN. The semi-supervised learning experiments are tested with a Monte Carlo cross-validation with ten repetitions for each training percentage. We also compute the sparsity percentages of our filter for $\rho \in \{1, 2, \ldots, 5\}$ for the sparse and non-sparse Sobolev norms. Therefore, we compute the average training time from ten repetitions on 20News with variations of

N in $\{100, 200, \ldots, 5500\}$, where the training percentage is set to 20%, and we use the default implementation of the state-of-the-art GNNs. Finally, we compute the homophily metric $H(G)$ for the k-NN and learned graphs in each dataset.

Results

Table 6.1 shows the summary of the comparison of state-of-the-art methods and S-SobGNN. The results are computed as the best mean in the set of training percentages for all datasets without restriction in the number of parameters. S-SobGNN presents competitive results in all datasets. Furthermore, we can see that smooth learned graphs present better results than k-NN graphs. Indeed, k-GNN and GCNII improve by a large margin their results with learned graphs compared to k-NN graphs. Therefore, learned graphs present advantages when we do not have a natural choice for the construction of the graph.

Table 6.2 presents the comparison of S-SobGNN against state-of-the-art methods when all GNNs have: 1) approximately the same amount of learnable parameters, and 2) the same hyperparameters such as learning rate, weigh decays, dropout, and others. S-SobGNN contains only one filtering operation $\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}\bar{\mathbf{A}}_{\rho}\bar{\mathbf{D}}_{\rho}^{-\frac{1}{2}}$ per layer in Table 6.2, and therefore no linear combination layer is required (in this case, we test S-SobGNN with values of ρ in the set $\{1, 2, \ldots, 6\}$). S-SobGNN shows competitive results on all datasets, and in some cases, our architecture presents an improvement of +2% accuracy regarding the second best method.

Table 6.3 shows the homophily metric for the datasets in the semi-supervised learning experiments. The learned graphs readily show stronger homophily in all cases. This explains the fact that all GNNs in Tables 6.1 and 6.2 in almost all cases performed better in learned graphs than in k-NN graphs.

Table 6.1: Accuracy (in %) for several state-of-the-art methods and our proposed S-SobGNN architecture in several datasets for semi-supervised learning, inferring the graphs with k-NN and the protocol to learn the graph as in Section 6.3.3. #Param is the number of learnable parameters, Acc. means accuracy, s.d. denotes standard deviation, T-Epoch is the average time per epoch. The best and second-best performing methods on each category are shown in **red** and **blue**, respectively.

Model	**Cancer-B**, $N = 1250$				**Cancer-M**, $N = 5000$				**20News**, $N = 3000$			
	#Param	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**T-Epoch**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**T-Epoch**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**T-Epoch**
Cheby [31]	2486	90.030 ± 1.441	90.970 ± 1.141	6.19ms	2486	64.882 ± 1.086	65.458 ± 0.957	7.31ms	130016	75.767 ± 0.981	76.796 ± 0.639	43.08ms
TAGConv [178]	2578	**90.745 ± 1.248**	90.930 ± 0.902	4.83ms	2968	64.848 ± 0.549	**65.993 ± 0.642**	7.29ms	128666	**75.938 ± 0.652**	75.913 ± 1.173	109.54ms
GCN [33]	658	89.170 ± 0.765	91.120 ± 1.037	3.45ms	760	62.812 ± 1.549	63.295 ± 1.082	3.40ms	32186	75.767 ± 0.801	77.213 ± 0.688	3.40ms
GAT [34]	2758	85.409 ± 0.955	86.223 ± 0.652	9.31ms	3160	42.761 ± 3.903	52.060 ± 2.692	15.31ms	128862	72.855 ± 1.213	73.688 ± 1.543	12.98ms
k-GNN [181]	5186	74.630 ± 17.36	87.599 ± 13.32	3.88ms	5960	45.334 ± 9.267	65.014 ± 5.089	5.38ms	257354	38.937 ± 20.89	67.040 ± 7.687	39.77ms
GCNII [184]	19010	76.403 ± 10.01	83.686 ± 0.989	7.52ms	19400	35.517 ± 7.996	41.998 ± 2.726	15.15ms	145098	56.958 ± 3.494	**77.313 ± 0.825**	11.65ms
GCN+DE [185]	658	90.150 ± 0.675	**92.140 ± 0.769**	3.65ms	760	64.571 ± 0.742	65.786 ± 0.676	3.93ms	32186	75.679 ± 1.013	76.863 ± 0.738	3.56ms
GraphSAINT [186]	283906	88.004 ± 1.356	89.360 ± 1.061	7.30ms	288520	**65.375 ± 0.734**	**66.563 ± 0.803**	25.21ms	1294602	70.864 ± 1.032	72.420 ± 0.925	44.89ms
S-SobGNN	3948	**92.430 ± 0.813**	**92.950 ± 1.183**	7.07ms	4560	**64.945 ± 1.090**	65.488 ± 1.255	16.29ms	128744	**76.938 ± 0.797**	**77.329 ± 0.719**	6.93ms

Model	**HAR**, $N = 3000$				**Isolet**, $N = 3000$			
	#Param	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**T-Epoch**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**T-Epoch**
Cheby [31]	36481	**94.193 ± 1.370**	**94.835 ± 1.222**	18.27ms	40121	**88.856 ± 0.462**	**88.015 ± 0.899**	12.52ms
TAGConv [178]	36310	90.898 ± 0.882	92.583 ± 1.450	42.66ms	41194	86.216 ± 1.887	86.054 ± 1.912	24.67ms
GCN [33]	9094	91.992 ± 0.589	93.679 ± 0.778	3.34ms	10330	86.490 ± 0.679	87.122 ± 0.711	3.32ms
GAT [34]	36498	90.028 ± 1.088	91.407 ± 0.804	14.15ms	41422	85.495 ± 1.291	86.533 ± 1.122	11.34ms
k-GNN [181]	72646	22.659 ± 17.88	42.138 ± 19.46	17.75ms	82394	43.199 ± 14.34	75.993 ± 9.593	10.76ms
GCNII [184]	52742	22.992 ± 6.596	85.305 ± 8.714	13.06ms	57626	16.746 ± 4.172	80.802 ± 2.624	7.61ms
GCN+DE [185]	9094	92.621 ± 0.651	94.325 ± 0.821	3.78ms	9094	87.087 ± 0.660	87.814 ± 0.724	3.56ms
GraphSAINT [186]	554758	88.433 ± 1.769	90.138 ± 1.025	29.87ms	598810	83.668 ± 0.670	84.840 ± 1.040	17.13ms
S-SobGNN	45470	**95.104 ± 0.475**	**95.912 ± 0.682**	8.64ms	41320	**89.279 ± 0.513**	**89.157 ± 0.555**	7.78ms

Table 6.2: Accuracy (in %) for several state-of-the-art methods and our proposed S-SobGNN architecture in several datasets for semi-supervised learning, inferring the graphs with k-NN and the protocol to learn the graph as in Section 6.3.3. All methods have approximately the same number of learnable parameters.

Model	**Cancer-B**, $N = 1250$			**Cancer-M**, $N = 5000$			**20News**, $N = 3000$		
	#Param	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**
Cheby [31]	654	72.015 ± 19.61	75.923 ± 14.48	654	44.868 ± 4.408	45.451 ± 4.675	34008	62.880 ± 2.511	63.102 ± 2.192
GCN [33]	658	89.220 ± 0.958	**91.350 ± 1.022**	760	**63.444 ± 0.866**	**64.017 ± 0.897**	32186	**75.671 ± 0.610**	**76.971 ± 0.460**
GAT [34]	694	85.725 ± 1.601	87.527 ± 1.627	808	52.477 ± 5.168	53.006 ± 4.232	32238	71.717 ± 1.389	70.911 ± 1.185
k-GNN [181]	650	86.782 ± 2.621	86.447 ± 12.69	752	48.577 ± 10.55	56.272 ± 5.047	32178	47.959 ± 13.77	58.147 ± 4.044
SIGN [173]	617	**89.870 ± 1.497**	90.395 ± 1.342	713	56.362 ± 3.799	56.109 ± 3.447	30175	62.708 ± 3.364	69.711 ± 2.117
S-SobGNN	658	**91.820 ± 1.251**	**92.010 ± 0.968**	760	**62.442 ± 1.174**	**63.101 ± 0.725**	32186	**77.008 ± 0.566**	**77.858 ± 0.452**

Model	**HAR**, $N = 3000$			**Isolet**, $N = 3000$		
	#Param	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**	**#Param**	**Test Acc.±s.d. k-NN**	**Test Acc.±s.d. Learned**
Cheby [31]	9545	89.238 ± 2.684	89.863 ± 4.649	10497	67.447 ± 4.341	68.450 ± 3.614
GCN [33]	9094	**91.610 ± 1.036**	**93.325 ± 1.065**	10330	**87.113 ± 1.154**	**86.930 ± 1.370**
GAT [34]	9138	83.563 ± 13.73	89.333 ± 2.664	10414	84.190 ± 1.100	84.851 ± 1.071
k-GNN [181]	9086	28.075 ± 16.51	59.382 ± 18.08	10322	5.386 ± 3.783	29.988 ± 15.44
SIGN [173]	8526	90.016 ± 3.680	90.275 ± 3.556	9686	70.749 ± 4.350	69.828 ± 10.93
S-SobGNN	9094	**93.976 ± 0.637**	**95.225 ± 0.931**	10330	**88.127 ± 0.800**	**88.570 ± 0.807**

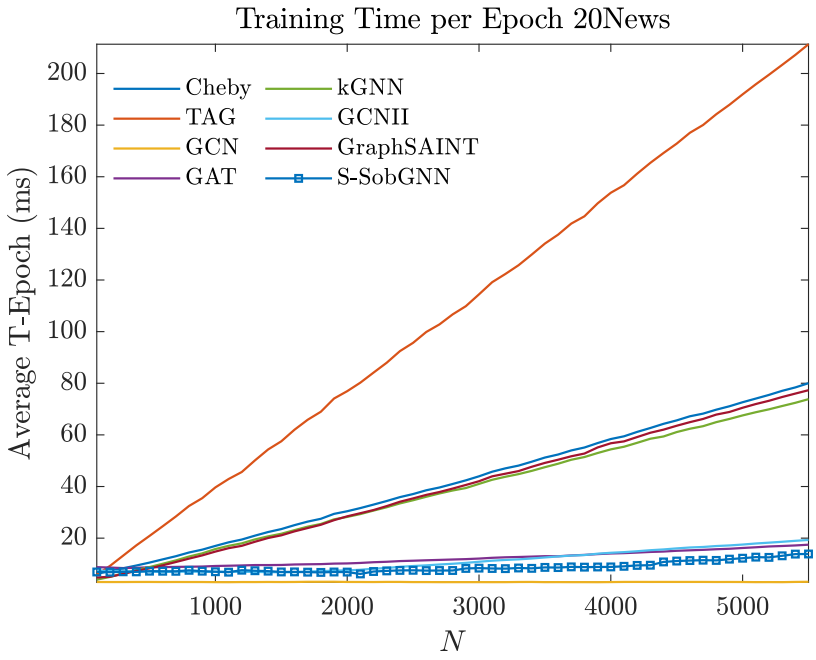

Figure 6-3: Average training time per epoch (T-Epoch) for several GNNs on 20News with variations in the number of nodes N.

Table 6.4 shows the ablation studies for α, ϵ, and k of S-SobGNN. Higher values of the parameter α show better performances because we have more diverse frequency components, *i.e.*, wider S-SobGNNs show better performance than narrow networks. Notice also that learned graphs outperform the k-NN graphs in almost all cases. Similarly, S-SobGNN shows stable results for the k values that we tested.

Table 6.5 shows the sparsity percentages of the sparse and non-sparse Sobolev norm for different values of ρ. We can notice how the non-sparse filter loses sparsity quickly for higher values of ρ. Finally, Fig. 6-3 shows the average training time per epoch (T-Epoch) on 20News for several GNNs with a GPU implementation using an Nvidia GTX 1070. The GNNs in Fig. 6-3 shows a linear complexity in the number of nodes for the given range of N. S-SobGNN has a low training time compared with several state-of-the-art GNNs due to the sparsity concepts developed in this chapter.

6.5.3 Benchmarking GNNs in Node Classification

Datasets: S-SobGNN is evaluated in the datasets *Pattern* and *Cluster*. These are the two artificial datasets for node classification of the benchmarking for graph neural net-

Table 6.3: Homophily index for the datasets for semi-supervised learning with different values of k.

Dataset	$k = 10$	$k = 20$	$k = 30$	$k = 40$
	k-NN|Lrnd.	k-NN|Lrnd.	k-NN|Lrnd.	k-NN|Lrnd.
Cancer-B	0.8625|0.8838	0.8447|0.8670	0.8334|0.8568	0.8230|0.8463
Cancer-M	0.6267|0.6534	0.5954|0.6242	0.5765|0.6054	0.5629|0.5921
20News	0.5557|0.6103	0.4968|0.5348	0.4615|0.4919	0.4354|0.4628
HAR	0.8849|0.9087	0.8478|0.8851	0.8211|0.8660	0.8007|0.8500
Isolet	0.7346|0.7619	0.6851|0.7255	0.6454|0.6943	0.6105|0.6664

works [165]. Pattern and Cluster are generated with stochastic block models for node classification, and they are used to model communities in social networks. For example, Pattern dataset tests the accuracy of some method to recognize predetermined subgraphs, while Cluster evaluates the identification of clusters in a semi-supervised scheme.

Experiments

For the benchmarking, we follow the same experimental setting as in [165], *i.e.*, we run our model four times with four different seeds.

Results

Table 6.6 shows the comparison of S-SobGNN with other methods in the benchmarking of graph neural networks for node classification [165]. S-SobGNN shows good results in Pattern and Cluster. Most importantly, S-SobGNN shows very competitive results in time per epoch (T-Epoch), where we trained our architecture in an Nvidia GTX 1070 GPU. The results of the other state-of-the-art methods are directly extracted from [165], where they trained these models with four Nvidia 2080Ti GPUs, which are more powerful than ours. The computational-time benefits of our architecture are due to the sparsity concepts developed in this chapter.

Table 6.4: Accuracy (in %) of S-SobGNN with variations of the parameters ϵ and α, as well as variations of the parameter k for the k-NN and the algorithm to learn graphs (Lrnd.) in all datasets for semi-supervised learning.

	Cancer-B, $N = 1250$					**Cancer-M**, $N = 5000$				
	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$
	k-NN|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*
$\epsilon = 0.5$	87.45|89.27	88.83|90.71	89.75|91.86	90.00|92.18	90.95|92.92	57.41|59.46	59.66|62.49	63.20|63.99	63.36|64.63	64.48|65.22
$\epsilon = 1$	87.83|89.56	89.35|91.44	90.72|92.35	91.26|92.39	92.00|92.95	56.77|57.97	61.35|62.26	62.27|63.63	63.98|64.78	64.69|64.59
$\epsilon = 1.5$	87.82|89.35	90.21|91.42	91.30|92.07	91.89|92.51	92.43|92.94	55.99|57.26	61.00|62.90	61.98|62.99	63.54|64.48	64.02|65.15
$\epsilon = 2$	88.34|89.83	90.21|90.91	91.03|91.85	92.12|92.72	92.36|92.73	56.66|56.47	60.56|61.36	62.89|63.62	63.25|63.59	64.95|65.49

	20News, $N = 3000$				
	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$
	k-NN|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*
$\epsilon = 0.5$	73.17|73.95	75.69|76.17	75.89|76.64	75.38|77.33	75.69|76.71
$\epsilon = 1$	72.73|75.95	75.52|75.24	76.94|76.30	76.49|75.76	76.67|75.63
$\epsilon = 1.5$	73.16|73.23	75.85|75.33	76.57|75.77	76.24|74.85	75.97|74.51
$\epsilon = 2$	73.46|72.84	76.19|75.31	76.53|75.12	76.29|74.55	76.35|75.73

	HAR, $N = 3000$				
	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$
	k-NN|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*
$\epsilon = 0.5$	91.80|94.68	92.53|95.35	92.97|95.82	93.10|95.91	93.02|95.59
$\epsilon = 1$	92.64|95.49	93.15|95.57	93.86|95.62	94.17|95.47	94.74|95.40
$\epsilon = 1.5$	93.13|95.33	94.08|95.43	94.10|95.07	94.96|95.08	95.10|95.03
$\epsilon = 2$	93.61|95.29	94.30|95.66	94.41|95.48	95.05|95.18	94.89|95.26

	Isolet, $N = 3000$				
	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$
	k-NN|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*
$\epsilon = 0.5$	84.92|87.66	87.01|88.41	87.03|88.49	86.96|88.06	87.02|88.23
$\epsilon = 1$	87.35|89.16	88.42|88.77	88.86|88.16	88.74|88.50	88.47|87.34
$\epsilon = 1.5$	88.06|88.19	88.87|87.79	89.28|88.26	88.68|87.97	88.85|87.98
$\epsilon = 2$	87.83|88.51	89.13|88.67	88.98|87.49	88.41|87.09	88.41|87.25

Dataset	$k = 10$	$k = 20$	$k = 30$	$k = 40$
	k-NN|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*	*k-NN*|*Lrnd.*
Cancer-B	92.83|93.14	92.82|92.84	92.20|92.91	92.07|92.62
Cancer-M	64.38|64.75	64.08|64.70	64.30|65.11	64.69|65.28
20News	76.11|76.16	75.92|76.50	76.40|76.63	76.31|76.36
HAR	95.52|96.08	95.65|96.30	95.53|95.99	94.92|96.00
Isolet	88.13|88.34	88.93|88.53	88.87|88.67	89.01|89.33

Table 6.5: Sparsity percentage for the sparse and non-sparse Sobolev filtering matrices.

Sobolev Filter	**Cancer-B**, $N = 1250$					**Cancer-M**, $N = 5000$					**20News**, $N = 3000$					**HAR**, $N = 3000$					**Isolet**, $N = 3000$				
	$\rho = 1$	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$	$\rho = 1$	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$	$\rho = 1$	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$	$\rho = 1$	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$	$\rho = 1$	$\rho = 2$	$\rho = 3$	$\rho = 4$	$\rho = 5$
Sparse	96.63	96.63	96.63	96.63	96.63	99.17	99.17	99.17	99.17	99.17	98.58	98.58	98.58	98.58	98.58	98.03	98.03	98.03	98.03	98.03	99.01	99.01	99.01	99.01	99.01
Non-Sparse	96.63	80.33	53.82	29.32	12.46	99.17	94.05	82.48	67.67	52.55	98.58	65.65	0.615	0	0	98.03	82.29	59.74	46.05	35.07	99.01	92.73	75.92	46.19	20.41

Table 6.6: Benchmarking results for several state-of-the-art methods and S-SobGNN, where L denotes the number of hidden layers. The results are averaged over four runs with four different seeds.

		Pattern					**Cluster**				
Model	L	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**T-Epoch**	**#Param**	**Test Acc.±s.d.**	**Train Acc.±s.d.**	**#Epoch**	**T-Epoch**
MLP	4	105263	50.519 ± 0.000	50.487 ± 0.014	42.25	8.95s	106015	20.973 ± 0.004	20.938 ± 0.002	42.25	5.83s
GCN [33]	4	100923	63.880 ± 0.074	65.126 ± 0.135	105.00	118.85s	101655	53.445 ± 2.029	54.041 ± 2.197	70.00	65.72s
GraphSage [32]	4	101739	50.516 ± 0.001	50.473 ± 0.014	43.75	93.41s	102187	50.454 ± 0.145	54.374 ± 0.203	64.00	53.56s
MoNet [179]	4	103775	85.482 ± 0.037	85.569 ± 0.044	89.75	35.71s	104227	58.064 ± 0.131	58.454 ± 0.183	76.25	24.29s
GAT [34]	4	109936	75.824 ± 1.823	77.883 ± 1.632	96.00	20.92s	110700	57.732 ± 0.323	58.331 ± 0.342	67.25	14.17s
GatedGCN [180]	4	104003	84.480 ± 0.122	84.474 ± 0.155	78.75	139.01s	104355	**60.404 ± 0.419**	61.618 ± 0.536	94.50	79.97s
GIN [182]	4	100884	85.590 ± 0.011	85.852 ± 0.030	93.00	15.24s	103544	58.384 ± 0.236	59.480 ± 0.337	74.75	10.71s
RingGNN [183]	2	105206	**86.245 ± 0.013**	86.118 ± 0.034	75.00	573.37s	104746	42.418 ± 20.06	42.520 ± 20.21	74.50	428.24s
S-SobGNN	4	107995	**85.616 ± 0.068**	85.613 ± 0.066	105.50	22.47s	107995	**61.791 ± 0.371**	62.322 ± 0.411	107.50	16.61s

6.6 Conclusion

In this chapter, we inferred smooth graphs and introduced a new GNN architecture with applications in semi-supervised learning, and node classification. We observed that smooth inferred graphs improve the performance of baseline GNN architectures by promoting homophily. Furthermore, our S-SobGNN model used the Sobolev norm of GSP to compute a more diverse set of functions for GNNs. Then, we extended the concept of Sobolev norms using the Hadamard product between matrices to keep the sparsity level of the graph representations. Several theoretical notions of our filtering operation were provided in Sections 6.4.1 and 6.4.2. Our architecture was designed with a cascade of filtering operations on each layer with an attention function to select the best operations. S-SobGNN was evaluated on publicly available data including, 1) four datasets for semi-supervised learning, and 2) two datasets for node classification. S-SobGNN outperformed several state-of-the-art methods in all applications.

This chapter opens several research directions. The first direction could be the application of S-SobGNN in the modeling of proteins for drug discovery [41]. Another important question is how we can further propose novel functions in the adjacency matrix to improve the expressiveness of GNNs maintaining sparsity?

Next Chapter 7 studies the relationship between the over-smoothing and over-squashing problems when stacking multiple graph convolutional layers in GNNs.

Chapter 7

On the Trade-off between Over-smoothing and Over-squashing in GNNs

7.1 Introduction

In the previous Chapter 6, we studied ideas to increase the expressive power of GNNs without compromising scalability. In this chapter, we study the relationship between over-smoothing and over-squashing in GNNs from a topological perspective.

Despite the successful developments and applications of GNNs, they can suffer from the typical limitations of neural networks like over-fitting, and vanishing gradients [38]. Similarly, GNNs have two known inherent limitations that are not fully understood yet: over-smoothing [46] and over-squashing [47]. Both problems, over-smoothing and over-squashing, occur when we stack several graph convolutional layers trying to capture long-range dependencies among samples in the data. The accepted explanation for over-smoothing is that node representations become indistinguishable when the number of convolutional layers increases [46, 189]. Similarly, we know that over-squashing consists of the distortion of an exponentially amount of information from distant nodes trying to pass through some edges, called "bottlenecks", in the

graph [47, 190]. The solution to these two problems is useful in situations where we have graphs with large diameters and long-range relationships between nodes, otherwise, shallow GNNs could be enough [47]. Several methods have been proposed to address over-smoothing [38, 185, 191–196] and over-squashing [47, 190]. However, both problems have been studied independently without a precise explanation concerning the connection between these issues.

In this chapter, we introduce a topological relationship between over-smoothing and over-squashing. From the side of over-smoothing, we use the random walk matrix to show how node representations exponentially converge to a stationary distribution [78]. We observe that the convergence rate of this exponential-decaying function depends on the first non-zero eigenvalue of the Laplacian matrix of the graph. From the side of over-squashing, we rely on the work of Topping *et al.* [190] to show how bottlenecks (and thus over-squashing) are also closely related to the first non-zero eigenvalue of the graph. Therefore, we can use the Cheeger inequality[1] [197] to show the underlying trade-off between over-smoothing and over-squashing, *i.e.*, we cannot arbitrarily improve one of them without worsening the other. Likewise, we propose a heuristic algorithm to find a good approximation for the trade-off between over-smoothing and over-squashing. Our algorithm uses a bound of the Ollivier's Ricci curvature [198] presented in [199]. We name this bound Jost and Liu Curvature (JLC) and our algorithm Stochastic Jost and Liu curvature Rewiring (SJLR). JLC is used to add edges in the graph as a pre-processing step, and then some edges are dropped in the training step based on the curvature metric and the node embeddings. We rewire the graph in two different stages to alleviate over-smoothing and over-squashing, so we try to address both problems simultaneously. The JLC metric is less computationally complex than the Balanced Forman Curvature (BFC) presented in [190] while keeping crucial theoretical properties. SJLR algorithm shows competitive performance in benchmarking homophilous and heterophyllous graph datasets for node classification.

The main contributions of this chapter are summarized as follows: 1) we present

[1]The Cheeger inequality bounds the first non-zero eigenvalue of the Laplacian with the Cheeger constant, which is at the same time related to the bottlenecks of the graph.

a topological relationship between over-smoothing and over-squashing, 2) we propose a new rewiring algorithm based on the JLC metric presented in [199] to address both over-smoothing and over-squashing, 3) we perform extensive experiments showing the properties of our SJLR algorithm, and 4) we release a codebase[2] based on PyG [187] for training, hyper-parameters tuning, and evaluation of methods aiming to alleviate over-smoothing or over-squashing (further details of the codebase are discussed in Appendix F). The rest of the chapter is organized as follows. Section 7.2 presents the related work. Section 7.3 introduces some preliminary concepts. Section 7.4 presents the relationship between over-smoothing and over-squashing. Section 7.5 explains the proposed SJLR algorithm. Finally, Sections 7.6 and 7.7 show the experiments and conclusions, respectively.

7.2 Related Work

Over-smoothing was first discussed in [46] to explain why node embeddings of distinct classes become indistinguishable when stacking multiple layers in GNNs. Later, several methods were proposed to ease over-smoothing [38,185,189,191–196,200,201]. We can briefly classify these approaches as:

1. **Graph rewiring methods:** Klicpera *et al.* [191,202] proposed an improved propagation scheme based on personalized PageRank for rewiring the graph as a preprocessing step. Rong *et al.* [185] and Huang *et al.* [201] introduced methods that drop edges from the graph when training the GNN. Finally, Chen *et al.* [189] presented an approach that adaptively changes the graph topology.
2. **Normalization techniques:** Zhao and Akoglu [193] and Zhou *et al.* [194] proposed node-embeddings normalization techniques to address over-smoothing directly, *i.e.*, they tried to avoid nodes becoming indistinguishable. Similarly, Oono and Suzuki [192] presented a procedure that normalizes the weights of the GNN architecture.
3. **Architectural changes:** Li *et al.* [38] proposed dilated convolutions and resid-

[2] `https://github.com/jhonygiraldo/Stochastic_Jost_Liu_Curvature_Rewiring`

ual/dense connections in GNNs to create deep architectures. Chen *et al.* [195] introduced an iteratively learning graph structure and graph embedding such that their method learns a better graph structure based on better node embeddings, and vice versa. Chien *et al.* [196] presented a new graph convolutional filter inspired by the graph signal processing literature [203] to avoid over-smoothing.

4. **Subgraphs approaches:** Zeng *et al.* [200] recently proposed a new idea to avoid over-smoothing, where localized subgraphs are extracted to train GNNs of arbitrary depths.

Alon and Yahav [47] introduced over-squashing to explain why GNNs struggle to propagate information between distant nodes in the graph. They also presented a rewiring method where a fully-adjacent matrix is added in the last GNN layer. Later, Topping *et al.* [190] proposed a theory explaining where the over-squashing comes from and how to alleviate it from a topological point of view. They presented a rewiring method using concepts of Ricci flow curvature from differential geometry [204]. These two studies [47,190] focused only on over-squashing, while in this chapter we introduce for the first time the trade-off between both issues, over-smoothing and over-squashing. Similarly, we try to address both problems simultaneously, using JLC and node-embedding metrics to rewire the graph.

7.3 Preliminaries

In this chapter we only consider unweighted graphs, *i.e.*, $\mathbf{A} \in \{0,1\}^{N\times N}$.

7.3.1 Cheeger Inequality and Cheeger Constant

Definition 7.3.1. *Let $G = (\mathcal{V}, \mathcal{E})$ be a graph, and let $\mathcal{S} \subset \mathcal{V}$ be a subset of nodes. Let $\partial\mathcal{S}$ be the set of edges going from a node in $\mathcal{S}$ to a node in $\mathcal{V} \setminus \mathcal{S}$, i.e., $\partial\mathcal{S} \triangleq \{\{u, v\} \in \mathcal{E} : u \in \mathcal{S}, v \in \mathcal{V} \setminus \mathcal{S}\}$. Therefore, we can define the Cheeger constant h_G of G as:*

$$h_G \triangleq \min_{\mathcal{S}} h_G(\mathcal{S}), \tag{7.1}$$

where $h_G(\mathcal{S}) = |\partial\mathcal{S}|/\min(\text{vol}(\mathcal{S}), \text{vol}(\mathcal{V} \setminus \mathcal{S}))$, $\text{vol}(\mathcal{S}) = \sum_{i \in \mathcal{S}} d_i$.

Intuitively, the Cheeger constant in Definition 7.3.1 is small when there exists a *bottleneck* in G, *i.e.*, when there are two sets of nodes with few edges between them. Similarly, we know that $h_G > 0$ if and only if G is a connected graph [78]. We can relate the Cheeger constant h_G with the first non-zero eigenvalue of $\mathcal{L}$ through the Cheeger inequality:

$$2h_G \geq \lambda_2 \geq \frac{h_G^2}{2}. \tag{7.2}$$

We notice from (7.2) that for having less "bottleneckness" in the graph, we need to promote big values of h_G, *i.e.*, having large values of λ_2 will increase the Cheeger constant since $h_G \geq \lambda_2/2$.

7.3.2 Over-squashing

The over-squashing problem is a more recent and less understood problem than over-smoothing. A graph learning problem has *long-range dependencies* when the outputs of GNNs depend on features of interacting distant nodes. In that scenario, information from non-adjacent nodes should be propagated through the network without distortion. Let $\mathcal{B}_r(i) \triangleq \{j \in \mathcal{V} : d_G(i,j) \leq r\}$ be the receptive field of an r-layer GNN for the node i, where $d_G(i,j)$ is the shortest-path distance between the nodes i and j, and $r \in \mathbb{N}$. In many graphs, $|\mathcal{B}_r(i)|$ grows exponentially with r and then representations of an exponential amount of neighboring nodes should be compressed into fixed-size vectors. This phenomenon is referred to as over-squashing[3] of information [47, 190].

[3]Further details about over-squashing can be found in Section 2.2 in [190].

7.4 Understanding the Over-smoothing vs. Over-squashing Trade-off

7.4.1 The Stationary Distribution on Graphs

Let $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ be the random walk transition matrix. For any initial distribution $f : \mathcal{V} \to \mathbb{R}$ with $\sum_{v\in\mathcal{V}} f(v) = 1$, the distribution after k steps is given by $\mathbf{f}^\top\mathbf{P}^k$, where $\mathbf{f} \in \mathbb{R}^{N\times 1}$ is the vector of initial distributions such that $\mathbf{f}(i)$ is the function evaluated on the ith node. The random walk is *ergodic* when there is a unique *stationary distribution* $\boldsymbol{\pi}$ satisfying that $\lim_{s\to\infty} \mathbf{f}^\top\mathbf{P}^s = \boldsymbol{\pi}$ [78].

Lemma 7.4.1 (Chung [78])**.** *Let* $\mathbf{P}$ *be the random walk transition matrix, let* $\boldsymbol{\pi}$ *be the stationary distribution of the corresponding walk, and let* $\mathbf{f}$ *be any initial distribution. For* $s \in \mathbb{N}^+$*, we know:*

$$\|\mathbf{f}^\top\mathbf{P}^s - \boldsymbol{\pi}\|_2 \leq e^{-s\lambda'} \frac{\max_i \sqrt{d_i}}{\min_j \sqrt{d_j}}, \text{ where } \lambda' = \begin{cases} \lambda_2 & \text{if } 1-\lambda_2 \geq \lambda_N - 1, \\ 2-\lambda_N & \text{otherwise.} \end{cases} \tag{7.3}$$

Therefore, we can compute the value of s *such that* $\|\mathbf{f}^\top\mathbf{P}^s - \boldsymbol{\pi}\|_2 \leq \epsilon$ *as follows:*

$$s \geq \frac{1}{\lambda'} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{7.4}$$

Proof: See [78].

Notice that λ' is either λ_2 or $2-\lambda_N$ in Lemma 7.4.1. However, we can show that only λ_2 is crucial. Suppose that $\lambda_N - 1 > 1 - \lambda_2$, so that $\lambda' = 2 - \lambda_N$. We can consider the lazy walk on the graph G' formed by $\mathbf{A} + \mathbf{I}$. Therefore, the new graph Laplacian has eigenvalues $\tilde{\lambda}_i = \lambda_i/2 \leq 1$ [78], and then $1 - \tilde{\lambda}_2 \geq 1 - \tilde{\lambda}_N \geq 0$.

The key message of Lemma 7.4.1 is a simplified version of the same observations of previous works [185] and [192], *i.e.*, GNNs converge exponentially to the stationary distribution when stacking several layers. However, we show that the convergence of this exponential function depends on the first non-zero eigenvalue of the graph

λ_2. We will use this result later to show the underlying relationship between over-smoothing and over-squashing. Similarly, we can have a simplified explanation of why sparsification methods in GNNs, like DropEdge [185], can alleviate over-smoothing.

Lemma 7.4.2 (Chung [78]). *Let G be a graph with diameter $Di \geq 4$, then:*

$$\lambda_2 \leq 1 - 2\frac{\sqrt{(\max_i d_i) - 1}}{\max_i d_i}\left(1 - \frac{2}{Di}\right) + \frac{2}{Di}. \tag{7.5}$$

Proof: See [205].

Lemma 7.4.2 shows an upper bound of λ_2 so that reducing the maximum degree of G promotes small values of λ_2, *i.e.*, having a sparser graph, like the DropEdge method does, promotes low values of λ_2 and thus high values of s in (7.4).

7.4.2 Over-smoothing and Over-squashing

We can establish a link between the Cheeger constant h_G and the parameter s in (7.4) as follows:

Lemma 7.4.3. *Let h_G be the Cheeger constant of a given graph G, and let s be the number of required steps such that the ℓ_2 distance between $\mathbf{f}^\top \mathbf{P}^{\mathbf{s}}$ and $\boldsymbol{\pi}$ is at most ϵ. Therefore, we have that:*

$$2h_G \geq \frac{1}{s}\log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{7.6}$$

Proof: see Appendix G.

From Lemma 7.4.3, we notice that if $s \to \infty$ then $h_G \to 0$, *i.e.*, if we want to avoid converging to the stationary distribution, we need to promote a bottleneck-kind structure in the graph. Similarly, if $h_G \to \infty$ then $s \to 0$, *i.e.*, if we want to promote less "bottleneckness" in the graph, we need to accelerate the convergence to the stationary distribution.

We can make the connection between over-smoothing and over-squashing using Lemma 7.4.3, the Simple Graph Convolution (SGC) model [175], and the theoretical

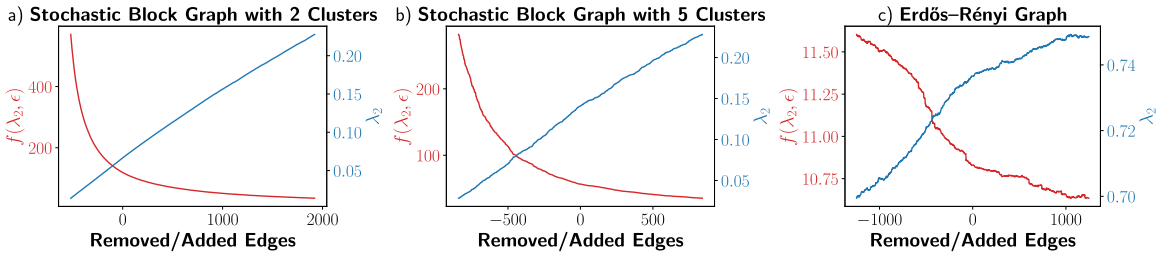

Figure 7-1: Mixing steps $f(\lambda_2, \epsilon)$ for $\epsilon = 5 \times 10^{-4}$ vs. number of removed or added edges for two stochastic block model graphs with a) two clusters, b) five clusters, and c) one Erdős-Rényi graph.

developments in [190]. In other words, we can use an SGC with a random walk kernel to show how the node embeddings converge to the stationary distribution when we stack several layers according to Lemma 7.4.3. Similarly, Topping *et al.* [190] explained how reducing bottlenecks in the graphs can alleviate over-squashing, *i.e.*, the receptive field of each node in a deep GNN will be polynomial in the hop-distance rather than exponential (see Corollary 3 in [190]). We leave for future work the analysis of the relationship between over-smoothing and over-squashing for more complex GNNs with non-linear activation functions.

Let $f(\lambda_2, \epsilon) = \frac{1}{\lambda_2} \log \left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}} \right)$ be the mixing steps of our graph, *i.e.*, the lower bound in the maximum number of layers of an SGC such that the difference between the initial and stationary distribution is at most ϵ. Figure 7-1 shows, for $\epsilon = 5 \times 10^{-4}$, how $f(\lambda_2, \epsilon)$ and λ_2 change when we add or remove edges in two artificial stochastic block model graphs and one Erdős-Rényi graph. We notice that we can increase the mixing steps by removing edges, *i.e.*, we can alleviate over-smoothing by making the graph more "bottleneckness". This partially explains why DropEdge [185] can alleviate over-smoothing. On the other hand, we increase λ_2 by adding edges as shown in Fig. 7-1, so we promote higher values of the Cheeger constant h_G, *i.e.*, we can alleviate over-squashing by making the graph less "bottleneckness". This can partially explain why the methodology by Topping *et al.* [190] can alleviate over-squashing. However, we notice that there is a trade-off between $f(\lambda_2, \epsilon)$ and λ_2 from a topological point of view, *i.e.*, we can increase $f(\lambda_2, \epsilon)$ by removing key edges but λ_2 will decrease, and vice versa. The algorithm to add and remove edges is explained in Section 7.5.

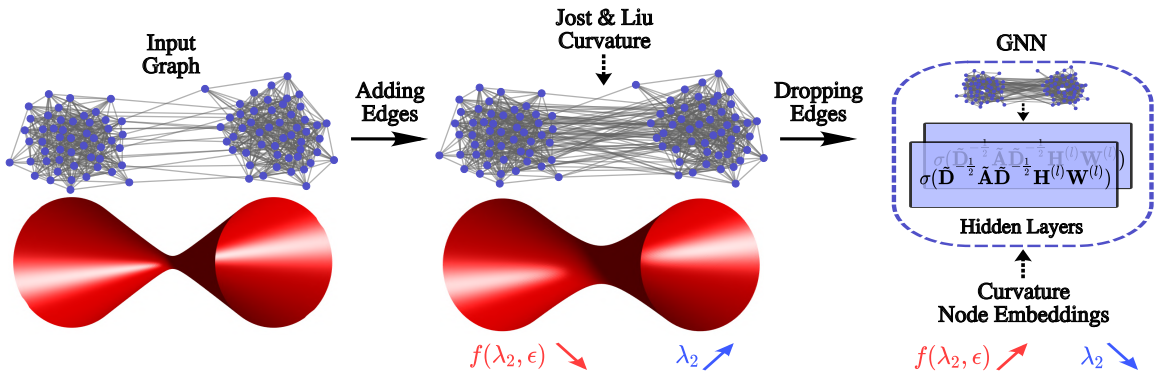

Figure 7-2: The pipeline of the proposed Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm.

7.5 Jost-Liu Curvature Rewiring

Figure 7-2 shows the pipeline of our Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm, where edges are added as a pre-processing step, and later edges are removed while training the GNN using JLC and node-embeddings information. We keep the addition and removal of edges in two different stages attempting to find a good solution in the over-smoothing vs. over-squashing trade-off.

Topping *et al.* [190] proposed a method to alleviate over-squashing using concepts of Ricci flow curvature. We can understand curvature-based methods using the following analysis:

Theorem 7.5.1 (Theorem 4.2 in [206])**.** *Let G be a finite graph, let λ_2 be its first non-zero Laplacian eigenvalue, and let $\kappa(i,j)$ be the Ricci curvature as defined in [206]. If for any edge (i,j), $\kappa(i,j) \geq \kappa > 0$, then $\lambda_2 \geq \kappa$.*
Proof: See [206].

Corollary 7.5.2. *If $\kappa(i,j) \geq \kappa > 0$ for any edge (i,j), then $2h_G \geq \kappa$.*
Proof: Using the Cheeger inequality and Theorem 7.5.1, we have that $2h_G \geq \lambda_2 \geq \kappa$, and then $2h_G \geq \kappa$.

From Theorem 7.5.1 and Corollary 7.5.2, we can conclude that if we have positive Ricci curvature everywhere, then $2h_G \geq \kappa$. Therefore, increasing curvature will make the graph less "bottleneckness".

Toppings *et al.* [190] defined a new metric, called Balanced Forman Curvature (BFC), to solve the problem of over-squashing. BFC requires counting triangles, 4-cycles, and the maximal number of 4-cycles based at edge (i, j) traversing a common node for each edge we are processing. This process makes BFC very computationally complex and unsuitable for practical applications. In this chapter, we use a bound of the Ollivier's Ricci curvature [198] presented in [199], namely JLC.

Definition 7.5.1 (Jost and Liu Curvature (JLC) [199])**.** *For any edge (i, j) in a finite graph:*

$$JLC(i,j) = -\left(1 - \frac{1}{d_i} - \frac{1}{d_j} - \frac{\#(i,j)}{d_i \wedge d_j}\right)_+ - \left(1 - \frac{1}{d_i} - \frac{1}{d_j} - \frac{\#(i,j)}{d_i \vee d_j}\right)_+ + \frac{\#(i,j)}{d_i \vee d_j}, \tag{7.7}$$

where $\#(i, j)$ is the number of triangles which include (i, j) as nodes, $c_+ \triangleq \max(c, 0)$, $c \vee t \triangleq \max(c, t)$, and $c \wedge t \triangleq \min(c, t)$.

Theorem 7.5.3 (Jost and Liu [199])**.** *On a locally finite graph we have that $\kappa(i, j) \geq JLC(i, j)$.*
Proof: See [199].

Corollary 7.5.4. *If $JLC(i, j) \geq \kappa > 0$ for any edge (i, j), then $\kappa(i, j) \geq \kappa > 0$.*
Proof: Using Theorems 7.5.1 and 7.5.3 we have that $\kappa(i, j) \geq JLC(i, j) \geq \kappa \geq 0$, then $\kappa(i, j) \geq \kappa \geq 0$.

From Corollary 7.5.4 we can conclude that if we have positive JLC everywhere in the graph, $\kappa(i, j)$ will also be positive everywhere. As a result, having positive JLC in the graph ensures that the receptive field of each node in a deep GNN will be polynomial in the hop-distance rather than exponential (see Corollary 3 in [190]). As a result, JLC is less computationally complex than the BFC in [190] while keeping the same theoretical properties about the polynomial receptive field growth.

7.5.1 Curvature Rewiring Algorithm

Our proposed algorithm uses JLC and features information to perform rewiring, *i.e.*, we remove and add edges based on their JLC metric and feature information of the

Algorithm 2 Stochastic Jost and Liu Curvature Rewiring

Input: Graph G, GNN architecture
Initialization: Hyper-parameters p_A, p_D, τ, α.

Compute $\text{JLC}(i,j)\ \forall\ (i,j) \in \mathcal{E}_1$, $A = p_A|\mathcal{E}_1|$, and $D = p_D|\mathcal{E}_1|$, where $\mathcal{E}_1 = \mathcal{E}$
for $m = 1$ until $m = A$ **do**
  $(i', j') = \arg\min_{(i,j)} \text{JLC}(i,j)$
  Compute the set of edges $\mathcal{A} = \{(\mathcal{N}_{i'} \setminus j') \times (\mathcal{N}_{j'} \setminus i') : (\mathcal{N}_{i'} \setminus j') \times (\mathcal{N}_{j'} \setminus i') \notin \mathcal{E}_m\}$
  Compute for all $0 < p \leq |\mathcal{A}|$, $\boldsymbol{\sigma}(p) = \text{JLC}'(r,s) - \text{JLC}(i',j')$, where $(r,s) = \mathcal{A}(p)$
  Add randomly an edge $(r', s') \in \mathcal{A}$ to $\mathcal{E}_m$ according to the distribution $\text{softmax}(\tau\boldsymbol{\sigma})$
  Update $\text{JLC}(i,j)\ \forall\ (i,j) \in \mathcal{E}_{m+1}$, where $\mathcal{E}_{m+1} = \mathcal{E}_m \cup (r', s')$
end for
Normalize $\text{JLC}(i,j)$ to be in $[0,1]$ for all $(i,j) \in \mathcal{E}_A$.
for each layer l in GNN while training **do**
  Compute $\mathbf{d}^{(l)}(p) = \|\mathbf{h}_i^{(l)} - \mathbf{h}_j^{(l)}\|_2\ \forall\ (i,j) \in \mathcal{E}_A$
  Drop D edges according to the probability distribution $\text{softmax}(\alpha\boldsymbol{\phi} + (1-\alpha)\mathbf{d}_n^{(l)})$
end for

nodes. Topping *et al.* [190] proposed a method where edges are added and removed as a pre-processing step. However, the reason for removing edges was not properly justified in [190]. We can argue that removing edges is also important to alleviate over-smoothing according to the developments in Section 7.4. Our SJLR algorithm also adds and removes edges but in a fundamentally different way. We keep both processes separate so that we can alleviate over-squashing and over-smoothing according to the dataset as shown in Fig. 7-2. We define a set of hyper-parameters for SJLR: 1) let p_A be the percentage of added edges, 2) let p_D be the percentage of dropped edges, 3) let τ be a variable controlling how stochastic SJLR is when adding edges, and 4) let $\alpha \in [0,1]$ be a variable controlling how important is the JLC metric against the embedding information while dropping edges. We optimize the hyper-parameters in the validation set so that SJLR can choose either if the specific dataset requires more adding or removal of edges, *i.e.*, SJLR tries to find the "sweet point" in the trade-off between over-squashing and over-smoothing.

Algorithm 2 presents our SJLR approach in detail. The SJLR algorithm has as input an initial graph G and a given GNN architecture. $\mathcal{N}_l$ is the set of neighbors of node l, and $\boldsymbol{\sigma} \in \mathbb{R}^{|\mathcal{A}|}$ is the vector of JLC improvements if adding an edge from

$\mathcal{A}$. Similarly, $\text{JLC}'(r,s)$ is the JLC metric computed in $G' = (\mathcal{V}, \mathcal{E}_m \cup (r,s))$. $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times F} = [\mathbf{h}_1^{(l)}, \dots \mathbf{h}_N^{(l)}]^\intercal$ is the F-dimensional embeddings of all nodes after l layers such that each $\mathbf{h}_i^{(l)} \in \mathbb{R}^F$. Finally, $\boldsymbol{\phi} \in \mathbb{R}^{|\mathcal{E}_A|}$ is the vector with the normalized values of $\text{JLC}(i,j)\ \forall\ (i,j) \in \mathcal{E}_A$, and $\mathbf{d}_n^{(l)} \in [0,1]^{|\mathcal{E}_A|}$ is the vector of normalized distances between node embeddings. SJLR algorithm is divided into two parts: 1) the addition of edges using the JLC metric, and 2) the removal of edges while training the GNN. Notice that SJLR is general to any GNN architecture. However, we only test our algorithm using GCNs [33] in Section 7.6 due to limited computational resources.

7.6 Experimental Framework and Results

7.6.1 Experiments

We perform a set of experiments to compare SJLR with several approaches in the literature, striving to understand the underlying relationship between over-smoothing and over-squashing. SJLR is compared with seven state-of-the-art methods to alleviate over-smoothing or over-squashing, including: Residual/Dense Connections (RDC) [38], Graph Diffusion Convolution (GDC) with personalized PageRank kernel [202], DropEdge (DE) [185], PairNorm (PN) [193], Differentiable Group Normalization (DGN) [194], Fully-Adjacent (FA) layers [47], and Stochastic Discrete Ricci Flow (SDRF) [190]. Our GNN base model is GCN [33] for all experiments. We evaluate all methods in nine datasets: Cornell, Texas, and Wisconsin from the WebKB project[4], Chameleon [207], Squirrel [207], Actor [208], Cora [209], Citeseer [166], and Pubmed [210]. These databases are related to data mining problems, where the objectives are to classify web pages or scientific papers into predefined classes, for example. We optimize the hyperparameters with a random search procedure by maximizing the average accuracy in the validation sets. The search space and the best hyperparameters for each experiment are provided in Appendix H. We consider the largest connected component of the graph for each dataset as in [190, 202]. We

[4] `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/`

split the data into train/validation/test sets, where we first divide the data into a development set and a test set. This is done once to avoid using the test set in the hyperparameter optimization procedure. We follow the same experimental framework as in [190, 202], *i.e.*, we optimize the hyperparameters for all dataset-preprocessing combinations separately by random search over 100 data splits. Furthermore, we report average accuracies on the test set with 95% confidence intervals calculated by bootstrapping with 1000 samples. The development set contains 1500 nodes and the rest of the nodes are used for testing in Cora, Citeseer, and Pubmed. The train set contains 20 nodes of each class while the rest of the nodes are used for validation. For the other datasets, we use a 60/20/20 split of the nodes, *i.e.*, 60% for training, 20% for validation, and 20% for testing. We use the same method and random seeds as in [190, 202], so we expect to have the same partitions and comparable results.

The search space for the number of layers of the GNN in the first experiment is $\{1, 2, 3\}$ as in [190] (for further details please see Appendix H). However, this search space does not provide insights about the problems we could face when stacking several layers. As a result, we perform a second experiment where we stack several layers in the set $\mathcal{F} = \{2, 8, 32\}$. We follow the same methodology as in the first experiment for each method and for each number of layers in $\mathcal{F}$, regarding data partitions and hyperparameters optimization. Finally, we calculate the average running time over ten repetitions to compute the metrics BFC and JLC for all edges in three artificial graphs with different amounts of nodes.

7.6.2 Implementation Details

All methods are implemented using PyTorch and PyG [187]. We use the same architectural components in all methods for a fair comparison. We use GCN [33] as graph convolutional layers, notice however that other GNNs like GAT [34] can be used as well. We use Rectified Linear Unit (ReLU) and log-softmax as activation functions in our GNN architectures. The number of GNN layers, hidden units, learning rate, weight decay, and dropout rate for each method are optimized in the validation set on each experiment (for further details please see Appendix H). For GDC, we apply

Table 7.1: Results of the first experiment regarding the comparison of our SJLR algorithm with several state-of-the-art methods to alleviate over-smoothing and over-squashing.

Method	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Citeseer	Pubmed
Baseline	66.18 ± 1.64	59.44 ± 1.22	52.11 ± 1.08	**40.82 ± 0.47**	41.77 ± 0.33	29.16 ± 0.30	**82.02 ± 0.28**	69.23 ± 0.30	78.44 ± 0.37
RDC [38]	66.77 ± 1.68	58.52 ± 1.11	51.85 ± 1.03	**41.50 ± 0.57**	**41.88 ± 0.35**	29.26 ± 0.33	81.79 ± 0.27	69.10 ± 0.34	78.49 ± 0.36
GDC [202]	67.01 ± 1.43	56.97 ± 1.07	59.97 ± 1.03	38.27 ± 0.50	36.64 ± 0.32	**32.68 ± 0.17**	**82.52 ± 0.26**	**69.28 ± 0.28**	**78.84 ± 0.38**
DE [185]	**80.41 ± 1.22**	**80.12 ± 1.07**	**84.17 ± 1.08**	40.48 ± 0.51	41.21 ± 0.31	**33.20 ± 0.23**	81.59 ± 0.25	69.04 ± 0.39	78.27 ± 0.38
PN [193]	65.26 ± 1.57	61.40 ± 1.00	52.77 ± 0.89	40.56 ± 0.43	40.92 ± 0.32	28.74 ± 0.22	79.62 ± 0.31	67.27 ± 0.40	77.29 ± 0.34
DGN [194]	67.49 ± 1.64	60.19 ± 0.93	52.33 ± 0.97	39.80 ± 0.57	**41.81 ± 0.32**	28.26 ± 0.31	81.55 ± 0.26	69.13 ± 0.36	78.39 ± 0.36
FA [47]	53.57 ± 0.00	59.26 ± 0.00	42.98 ± 0.49	27.28 ± 0.40	31.51 ± 0.00	$23.84 \pm 0.43^{*}$	29.85 ± 0.00	23.23 ± 0.00	**$79.48 \pm 0.12^{*}$**
SDRF [190]	58.26 ± 1.43	54.88 ± 1.45	55.62 ± 0.92	40.11 ± 0.47	41.78 ± 0.30	28.83 ± 0.36	81.05 ± 0.28	69.21 ± 0.33	78.40 ± 0.33
SJLR (ours)	**71.90 ± 1.97**	**66.18 ± 1.44**	**66.12 ± 1.27**	39.11 ± 0.51	40.64 ± 0.27	31.29 ± 0.24	81.27 ± 0.24	**70.51 ± 0.32**	78.03 ± 0.35

* Certain results were copied directly from [190] because of limited computational resources.
The best and second-best performing methods on each dataset are shown in **red** and **blue**, respectively.

weight decay regularization only in the first graph convolutional layer, otherwise we do not get comparable results as in [202]. All methods are trained for 1000 epochs using Adam optimizer [138]. We do not use early stopping or learning schedulers for any method. We make all graphs undirected, and we also remove all the self loops from the input graph. We evaluate all methods under the same configuration unless otherwise stated. We implemented SDRF [190] at our best understanding because there was not an official or available implementation of the method at the time of executing the experiments. However, we used JLC instead of BFC for our implementation of SDRF [190] because we do not have the computational resources to run the hyperparameter optimization using BFC. The hyperparameter optimizations were executed on several GPUs including: two Nvidia GeForce RTX 2080, two Nvidia GeForce GTX 1070, and two Nvidia Tesla V100.

7.6.3 Results

Table 7.1 shows the results of the first experiment related to the random search with few convolutional layers. We can notice two general trends: 1) rewiring methods like DE and SJLR dominate in heterophyllous datasets like Cornell, Texas, Wisconsin, Chameleon, Squirrel, and Actor; and 2) GDC leads in homophilous datasets like Cora, Citeseer, and Pubmed, while SJLR and FA offer competitive performances in some cases. Regarding the comparison between the curvature-based methods SJLR and SDRF [190], we notice that SJLR outperforms SDRF in six out of nine datasets. Both

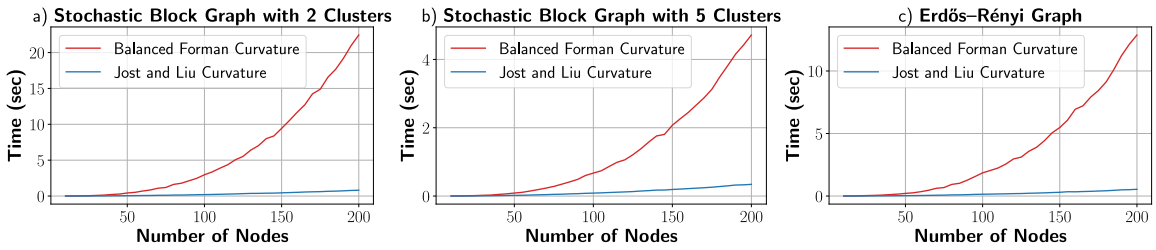

Figure 7-3: Average running time for balanced Forman curvature and Jost and Liu curvature for variations in the number of nodes in three artificial graphs.

SJLR and SDRF are executed using the same JLC metric in Table 7.1, and therefore we are assessing their performance based on the way the edges are added or removed. We can argue that SJLR is thus a critical improvement over SDRF regarding the practical adoption of curvature-based methods in GNNs. We can also notice that no single method can address all kinds of datasets, and perhaps an ensemble of different algorithms can be more suitable to outperform all previous approaches, which is not the scope of this chapter and we leave for future work. Finally, we remark that some methods like GDC [202] and FA [47] require specific architectural changes to work properly. For example, we achieve the specific results of GDC only when applying weight decay in the first graph convolutional layer as stated in Section 7.6.2.

Figure 7-3 shows the average running time over ten repetitions to compute the BFC and JLC for three artificial graphs with different amounts of nodes. We notice the large gap between the computation time of the JLC and BFC, which makes JLC more suitable for practical applications.

7.6.4 Limitations

Several studies in GNNs, like the ones reviewed in Section 7.2, have focused on solving the problems of over-smoothing or over-squashing without a clear idea of their practical importance. For example, the definitions of homophily in the literature usually consider only first-neighboring relationships. Similarly, the benchmarking graphs in Table 7.1 do not have large diameters. Therefore, we do not have mathematical or practical tools to assess how many "long-range dependencies" exist between samples in a specific dataset. Perhaps, the benchmarking datasets we have been testing so

far do not have many long-range dependencies, and carefully designed GNNs can be enough in these scenarios [211]. As a consequence, the results in Table 7.1 and also the results in previous works [38, 47, 185, 190, 193, 194, 202] might be only partially meaningful.

7.7 Conclusions

In this chapter, we presented a connection between the problems of over-smoothing and over-squashing in GNNs. We showed how both issues are intrinsically related to the eigenvalue gap of the normalized Laplacian matrix. As a result, we used the Cheeger inequality to show that we cannot arbitrarily improve over-smoothing and over-squashing simultaneously from a topological perspective, *i.e.*, there exists a trade-off between both problems. We also introduced a new Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm based on a bound of the Ollivier's Ricci curvature to alleviate over-smoothing and over-squashing. Our algorithm is less computationally complex than previous methods while guaranteeing important theoretical properties. SJLR outperforms some previous methods in homophilous and heterophyllous graph datasets for node classification.

The simple theoretical results presented in this chapter strive to improve our understanding of the problems we face when training deep GNNs. However, there are still many open problems in this domain. For example, two interesting questions for future work could be: 1) do we really need deep GNNs? and 2) when do we need deep GNNs? The answer to these two questions is practically meaningful and will prove the practical significance of this and previous works.

The following Chapter 8 explores the signal processing part of this thesis, where we are concerned with the problem of reconstruction of missing values in time-varying graph signals.

Part III

Signal Processing

Chapter 8

Reconstruction of Time-Varying Graph Signals via Sobolev Smoothness

8.1 Introduction

The previous Chapters 6 and 7 focused on the machine learning part of this thesis. We introduced new concepts on graphs trying to overcome some limitations of GNNs such as expresiveness, over-smoothing, and over-squashing. In this chapter, we explore the signal processing part of this thesis. Here we focus in the reconstruction of time-varying graph signals.

The sampling and reconstruction of graph signals are fundamental tasks in GSP that have recently received considerable attention. Naturally, the mathematics of sampling theory and spectral graph theory have been combined, leading to generalized Nyquist sampling principles for graphs [49, 53, 212, 213], where most of these previous works have focused on static graph signals. The reconstruction of time-varying graph signals is an important problem that has not been well explored[1]. The reconstruction

[1] One can think of the reconstruction of time-varying graph signals as a matrix completion problem where each row (or column) is associated with a node, and each column (or row) is associated with time.

of time-varying graph signals from discrete samples has several real world applications, such as the estimation of new cases of infectious diseases [8], or the recovery of the sea surface temperature for the study of the earth's climate dynamics [26].

Some recent methods for the reconstruction of time-varying graph signals have also used the assumption that temporal differences of graph signals are smooth. For example, Qiu *et al.* [26] extended the definition of smooth signals from static to time-varying graph signals, and solved an optimization problem for reconstruction. However, Qiu's method may have slow convergence rate because the eigenvalue spread of the Hessian associated with their optimization problem may be large. The techniques presented in [214–216] may be used to circumvent the problem of slow convergence. However, these techniques require eigenvalue decomposition or matrix inversion, which are computationally expensive for large graphs.

In the current chapter, we propose a new algorithm to reconstruct time-varying graph signals from samples. Our algorithm is based on the extension of the Sobolev norm defined in GSP for time-varying graph signals [3, 51]. Therefore, we dub our algorithm as Time-varying Graph signal Reconstruction via Sobolev Smoothness (GraphTRSS). Our algorithm uses a Sobolev smoothness function to formulate an optimization problem for the reconstruction of a time-varying graphs signal from its samples, where the samples are obtained with a random sampling strategy. The graph is constructed with a k-NN method. The optimization problem of GraphTRSS is solved with the conjugate gradient method. We analyze the convergence rate of our algorithm by studying the condition number of the Hessian associated with our optimization function, and we conclude that our algorithm converges faster than its closest competitor [26] under certain conditions. Moreover, GraphTRSS does not require expensive eigenvalue decomposition or matrix inversion. Finally, we evaluate our algorithm on synthetic data, two COVID-19, and two environmental datasets, where our algorithm outperforms several state-of-the-art methods.

The main contributions of this chapter are summarized as follows:

- We use the concept of Sobolev norms from static graph signals to define a smoothness function for time-varying graph signals, and then we use this new conception

to introduce an algorithm for reconstruction.

- We provide several mathematical insights of GraphTRSS. Specifically, we analyze the convergence rate of our algorithm by studying the condition number of the Hessian associated with our problem.
- GraphTRSS improves convergence speed without relying on expensive matrix inversion or eigenvalue decomposition.
- Concepts of reconstruction of graph signals from GSP are introduced in the mathematical modeling of infectious diseases for COVID-19, as well as environmental data.

The rest of this chapter is organized as follows. Section 8.2 shows related work in time-varying graph signals reconstruction. Section 8.3 presents the theoretical background of this chapter. Section 8.4 explains the details of GraphTRSS. Section 8.5 introduces the experimental framework. Finally, Sections 8.6 and 8.7 show the results and conclusions, respectively.

8.2 Related Work

The problems of sampling and reconstruction of graph signals have been widely explored in GSP [49, 53, 74, 75, 217–220]. Pesenson [48] introduced concepts of Paley-Wiener spaces in graphs, where a graph signal can be determined by its samples in a set of nodes called uniqueness set. One can say that a set of nodes is a uniqueness set of a certain graph if the fact that two graph signals in the Paley-Wiener space of the graph coincide in the uniqueness set implies that they coincide in the whole set of nodes. As a result, a bandlimited graph signal can exactly be reconstructed from its samples if the graph signal is sampled according to its uniqueness set. However, the bandlimitedness assumption is not realistic, *i.e.*, the graph signals in real-world datasets tend to be approximately bandlimited instead of strictly bandlimited. Therefore, several researchers have proposed reconstruction algorithms based on the smoothness assumption of graph signals [23,221,222], where the smoothness is measured with a Laplacian function. Similarly, other studies have used the

Total Variation of graph signals [223], or extensions of the concept of stationarity on graph signals [25, 224] for reconstruction.

For time-varying graph signals, scientists have developed notions of joint harmonic analysis linking together the time-domain signal processing techniques with GSP [225], while other researchers have proposed reconstruction algorithms assuming bandlimitedness of the signals at each instant [223, 226]. Most of these methods do not fully exploit the underlying temporal correlations of the time-varying graph signals. Qiu *et al.* [26] proposed an approach where the temporal correlations are captured with a temporal difference operator in the time-varying graph signal. However, Qiu's method may have slow convergence because the optimization problem depends on the Laplacian matrix. In particular, the eigenvalue spread of the Hessian associated with their problem may be large, leading to poor condition numbers. Other studies have reported the undesirable effects in convergence due to the large eigenvalue spread when dealing with shift operators derived from the Laplacian or the adjacency matrix. For example, Hua *et al.* [215] used a method of second-order moments to solve the problem of slow convergence speed, and this approach showed improvement in performance at the expense of additional computational cost by expensive matrix inversion operations. We can also circumvent the problem of slow convergence by using energy-preserving shift operators as in [214]. For example, Xia *et al.* [216] exploited an energy-preserving shift operator for distributed learning problems over graphs. However, the energy-preserving shift operators require the eigendecomposition of the adjacency (or Laplacian) matrix, which is computational prohibitive for large graphs.

In the current chapter, we assume smoothness in the temporal differences of graph signals as in [26]. However, our algorithm improves the condition number of the Hessian associated with our problem without requiring expensive eigenvalue decompositions or matrix inversions. We show that GraphTRSS converges faster than Qiu's method [26] under some conditions.

8.3 Reconstruction of Time-Varying Graph Signals

Several studies have used the smoothness assumption in graphs to solve problems of reconstruction and sampling of graph signals. Formally, notions of smoothness in $\mathbf{x}$ have been introduced with concepts of local variation [29] as in Section 2.4. Some researchers have used the graph Laplacian quadratic form as a regularization term to solve problems of reconstruction of graph signals [27]. For time-varying graph signals, an extension of the graph Laplacian quadratic form has also been used for reconstruction [26].

Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]$ be a time-varying graph signal, where $\mathbf{x}_i \in \mathbb{R}^N$ is a graph signal in G at time i. One can extend the concept of graph Laplacian quadratic form to time-varying graph signals by summing the Laplacian quadratic form of each graph signal $\mathbf{x}_i$. Then, we have:

$$S_2(\mathbf{X}) = \sum_{i=1}^{M} \mathbf{x}_i^\intercal \mathbf{L} \mathbf{x}_i = \text{tr}(\mathbf{X}^\intercal \mathbf{L} \mathbf{X}). \tag{8.1}$$

However, notice that (8.1) does not have temporal relationships between graph signals at different times i. To include time information, Qiu *et al.* [26] introduced the temporal difference operator $\mathbf{D}_h \in \mathbb{R}^{M \times (M-1)}$ defined as follows:

$$\mathbf{D}_h = \begin{bmatrix} -1 & & & \\ 1 & -1 & & \\ & 1 & \ddots & \\ & & \ddots & -1 \\ & & & 1 \end{bmatrix} \in \mathbb{R}^{M \times (M-1)}. \tag{8.2}$$

As a consequence, the temporal difference graph signal is such that:

$$\mathbf{X}\mathbf{D}_h = [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_2, \ldots, \mathbf{x}_M - \mathbf{x}_{M-1}]. \tag{8.3}$$

Qiu *et al.* [26] noted that $S_2(\mathbf{X}\mathbf{D}_h)$ exhibits better smoothness properties compared to

$S_2(\mathbf{X})$ in real-world datasets for time-varying graph signals, *i.e.*, the difference signal $\mathbf{x}_i - \mathbf{x}_{i-1}$ exhibits smoothness in the graph even if the signal $\mathbf{x}_i$ is not smooth across the graph. They defined the structure of time-varying graph signals as follows [26]:

Definition 8.3.1. *The α-structured set $\mathcal{B}_\alpha(G)$ composed of smoothly evolving graph signals is defined as:*

$$\mathcal{B}_\alpha(G) = \left\{\mathbf{X} : \operatorname{tr}\left((\mathbf{X}\mathbf{D}_h)^\mathsf{T}\mathbf{L}\mathbf{X}\mathbf{D}_h\right) \leq (M-1)\alpha\right\}, \tag{8.4}$$

where α indicates the smoothness level of the time-varying graph signal.

Qiu *et al.* [26] used Definition 8.3.1 to introduce a Time-varying Graph Signal Reconstruction (TGSR) method as follows:

$$\min_{\tilde{\mathbf{X}}} \frac{1}{2}\|\mathbf{J} \circ \tilde{\mathbf{X}} - \mathbf{Y}\|_F^2 + \frac{\upsilon}{2} \operatorname{tr}\left((\tilde{\mathbf{X}}\mathbf{D}_h)^\mathsf{T}\mathbf{L}\tilde{\mathbf{X}}\mathbf{D}_h\right), \tag{8.5}$$

where $\mathbf{J} \in \{0,1\}^{N\times M}$ is the sampling matrix of $\mathbf{X}$, υ is a regularization parameter, and $\mathbf{Y} \in \mathbb{R}^{N\times M}$ is the matrix of signals that we know (the observed values). The sampling matrix $\mathbf{J}$ is defined as follows:

$$\mathbf{J}(i,j) = \begin{cases} 1 & \text{if } i \in \mathcal{S}_j, \\ 0 & \text{if } i \notin \mathcal{S}_j, \end{cases} \tag{8.6}$$

where $\mathcal{S}_j$ is the set of sampled nodes at column j.

Theorem 8.3.1 (Qiu *et al.* [26])**.** *The solution of (8.5) is unique when the following conditions are satisfied by the sampling matrix $\mathbf{J}$:*

1. *For any $n \in \{1,\ldots,N\}$, $\exists\ m \in \{1,\ldots,M\}$ such that $\mathbf{J}(n,m) = 1$.*
2. *There is a fiducial time $m_0 \in \{1,\ldots,M\}$, such that for any $m \in \{1,\ldots,M\}$, with $m \neq m_0$, there exist a node $n_m \in \{1,\ldots,N\}$ satisfying that $\mathbf{J}(n_m,m_0) = \mathbf{J}(n_m,m) = 1$.*

Proof: see [26].

Theorem 8.3.1 provides some properties that $\mathbf{J}$ should satisfy to obtain a unique

solution for (8.5). Notice that deterministic sampling methods [49, 53, 227, 228] or sampling on product graphs [229] do not satisfy the first condition of Theorem 8.3.1, so it is not suitable for these problems. In contrast, in the current chapter we propose a random sampling strategy as in [26].

Eqn. (8.5) reconstructs a time-varying graph signal $\tilde{\mathbf{X}}$ with 1) a small error $\|\mathbf{J} \circ \tilde{\mathbf{X}} - \mathbf{Y}\|_F^2$, and 2) a small value of the temporal difference graph signal smoothness $\text{tr}((\tilde{\mathbf{X}}\mathbf{D}_h)^\intercal \mathbf{L}\tilde{\mathbf{X}}\mathbf{D}_h)$. Finally, the parameter υ in (8.5) weights the importance between the error and smoothness terms. This parameter υ is usually tuned experimentally.

In the current chapter, we explore two additional temporal difference operators, a two-time steps operator:

$$\mathbf{D}_{h2} = \begin{bmatrix} -1 & & & \\ 2 & -1 & & \\ -1 & 2 & \ddots & \\ & -1 & \ddots & -1 \\ & & \ddots & 2 \\ & & & -1 \end{bmatrix} \in \mathbb{R}^{M \times (M-2)}, \tag{8.7}$$

and a three-time steps operator:

$$\mathbf{D}_{h3} = \begin{bmatrix} -1 & & & \\ -1 & -1 & & \\ 4 & -1 & \ddots & \\ -1 & 4 & \ddots & -1 \\ -1 & -1 & \ddots & -1 \\ & -1 & \ddots & 4 \\ & & \ddots & -1 \\ & & & -1 \end{bmatrix} \in \mathbb{R}^{M \times (M-4)}. \tag{8.8}$$

The objective of introducing (8.7) and (8.8) is to capture different kinds of temporal relationships besides $\mathbf{X}\mathbf{D}_h$. For example, $\mathbf{X}\mathbf{D}_{h2} = [2\mathbf{x}_2 - \mathbf{x}_1 - \mathbf{x}_3, 2\mathbf{x}_3 - \mathbf{x}_2 -$

$\mathbf{x}_4, \ldots, 2\mathbf{x}_{M-1} - \mathbf{x}_{M-2} - \mathbf{x}_M]$.

8.4 Sobolev Smoothness of Time-Varying Graph Signals

In the current chapter, we extend the definition of Sobolev norms in GSP [3, 51] from static graph signals to a smoothness function for time-varying graph signals, and then we formulate a new reconstruction algorithm. The Sobolev norm was defined by Pesenson [51], and introduced in Chapter 3, Eqn. (3.2). When $\mathbf{L}$ is symmetric (as is the case in this chapter), we have that:

$$\|\mathbf{x}\|^2_{\beta,\epsilon} = \mathbf{x}^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\beta \mathbf{x}, \tag{8.9}$$

where $\epsilon \geq 0$, and $\alpha \in \mathbb{R}$. Notice the Sobolev norm in (8.9) is equal to the Laplacian quadratic form in (2.5) when $\epsilon = 0$ and $\beta = 1$. We use the Sobolev norm to define a new smoothness function for time-varying graph signals as follows:

Definition 8.4.1 (Sobolev smoothness of time-varying graph signals)**.** *Let* $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]$ *be a time-varying graph signal, let* $\mathbf{D}_h$ *be the temporal difference operator, and let* $\mathbf{L}$ *be the combinatorial Laplacian matrix of a graph* G*. For fixed parameters* $\epsilon \geq 0$ *and* $\beta \in \mathbb{R}^+$*, the Sobolev smoothness of* $\mathbf{X}$ *is given as follows:*

$$\begin{aligned} S_{\beta,\epsilon}(\mathbf{X}) &\triangleq \sum_{i=2}^{M} (\mathbf{x}_i - \mathbf{x}_{i-1})^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\beta(\mathbf{x}_i - \mathbf{x}_{i-1}) \\ &= \operatorname{tr}\left((\mathbf{X}\mathbf{D}_h)^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^\beta(\mathbf{X}\mathbf{D}_h)\right). \end{aligned} \tag{8.10}$$

8.4.1 Sobolev Reconstruction

We use the Sobolev smoothness in Definition 8.4.1 to formulate two new reconstruction algorithms. The first algorithm solves the problem for the noiseless case, while the second algorithm solves the noisy case.

In the noiseless case, we assume that the sampling mechanism does not add noise to the problem. As a consequence, the observed graph signal is given by $\mathbf{Y} = \mathbf{J} \circ \tilde{\mathbf{X}}$. One can formulate an optimization problem to get an approximate reconstruction of the time-varying graph signal as follows:

$$\min_{\tilde{\mathbf{X}}} \frac{1}{2} \operatorname{tr}\left((\tilde{\mathbf{X}}\mathbf{D}_h)^{\mathsf{T}} (\mathbf{L} + \epsilon \mathbf{I})^{\beta} \tilde{\mathbf{X}}\mathbf{D}_h \right) \text{ s.t. } \mathbf{J} \circ \tilde{\mathbf{X}} = \mathbf{Y}. \tag{8.11}$$

The optimization function in (8.11) reconstructs a smooth spatiotemporal graph signal $\tilde{\mathbf{X}}$ given the constraint $\mathbf{Y} = \mathbf{J} \circ \tilde{\mathbf{X}}$. The noiseless case can be solved by a gradient projection algorithm. The iterative update is such that:

$$\tilde{\mathbf{X}}^{t+1} = \left(\tilde{\mathbf{X}}^{t} - \xi \nabla_{\tilde{\mathbf{X}}} f_s(\tilde{\mathbf{X}}^{t}) \right)^{+}, \tag{8.12}$$

where $f_s(\tilde{\mathbf{X}}^t) = \frac{1}{2} \operatorname{tr}\left((\tilde{\mathbf{X}}^t\mathbf{D}_h)^{\mathsf{T}} (\mathbf{L} + \epsilon \mathbf{I})^{\beta} \tilde{\mathbf{X}}^t\mathbf{D}_h \right)$, ξ is the step size, $\nabla_{\tilde{\mathbf{X}}} f_s(\tilde{\mathbf{X}}^t)$ is the gradient of function $f_s(\tilde{\mathbf{X}}^t)$ given as:

$$\nabla_{\tilde{\mathbf{X}}} f_s(\tilde{\mathbf{X}}^{t}) = (\mathbf{L} + \epsilon \mathbf{I})^{\beta} \tilde{\mathbf{X}}^t \mathbf{D}_h \mathbf{D}_h^{\mathsf{T}}, \tag{8.13}$$

and $(\mathbf{V})^{+}$ is the projection of signal $\mathbf{V}$ to the signal space $\mathbf{Y} = \mathbf{J} \circ \tilde{\mathbf{X}}$ given as follows [26]:

$$(\mathbf{V})^{+} = \mathbf{Y} + \mathbf{V} - \mathbf{J} \circ \mathbf{V}. \tag{8.14}$$

The above formulation can be extended to the noisy case as follows. We take into account the noise by relaxing the constraint in (8.11):

$$\min_{\tilde{\mathbf{X}}} \frac{1}{2} \|\mathbf{J} \circ \tilde{\mathbf{X}} - \mathbf{Y}\|_F^2 + \frac{\upsilon}{2} \operatorname{tr}\left((\tilde{\mathbf{X}}\mathbf{D}_h)^{\mathsf{T}} (\mathbf{L} + \epsilon \mathbf{I})^{\beta} \tilde{\mathbf{X}}\mathbf{D}_h \right). \tag{8.15}$$

The optimization problem in (8.15) reconstructs a time-varying graph signal with a small error (given by the first term) and a small value for the Sobolev smoothness of its temporal difference signal. Here, we assume that the difference of graph signals $\mathbf{X}\mathbf{D}_h$ presents better smoothness properties than $\mathbf{X}$ alone.

In the current chapter, we solve (8.5) and (8.15) using the conjugate gradient

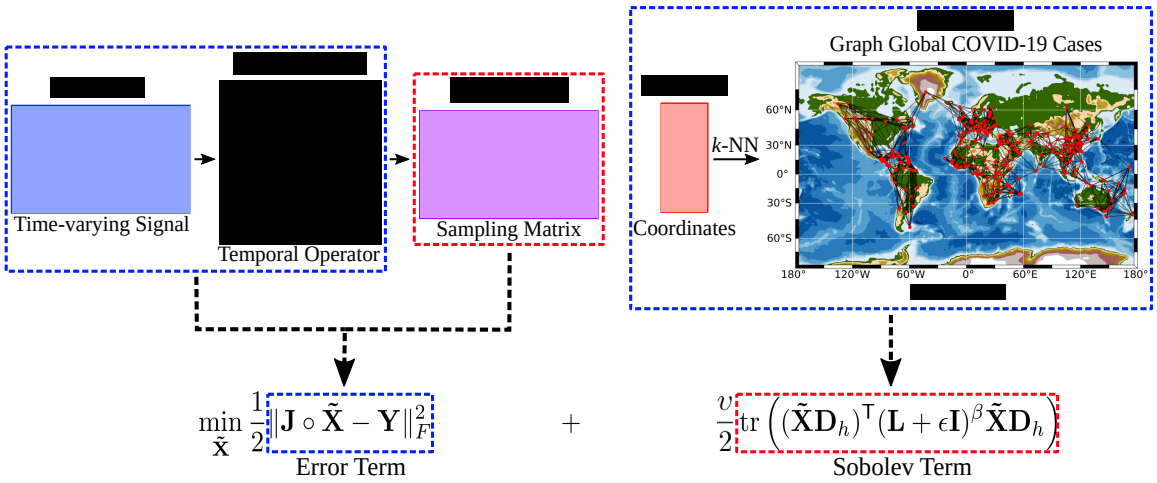

Figure 8-1: The framework of our algorithm (GraphTRSS) using a matrix of coordinates $\mathbf{M} \in \mathbb{R}^{N\times 2}$ to construct a graph with N regions in the world with confirmed cases of COVID-19 by November 18, 2020. The graph is constructed with a k-NN method. GraphTRSS uses the operator $\mathbf{D}_h \in \mathbb{R}^{M\times(M-1)}$ to capture temporal information in the time-varying signal $\mathbf{X} \in \mathbb{R}^{N\times M}$ with M temporal snapshots, and it also uses different sampling strategies $\mathbf{J} \in \{0,1\}^{N\times M}$ according to the desired output (reconstruction or forecasting). Finally, the optimization function, which includes the error and Sobolev terms, reconstructs or predicts the missing values, *i.e.*, the indexes of $\mathbf{X}$ where $\mathbf{J}$ has values zero.

method as in [26]. Therefore, we update the search direction $\Delta\tilde{\mathbf{X}}^t$ and the step size μ on each iteration t as follows:

$$\mu = -\frac{\langle \Delta\tilde{\mathbf{X}}^t, \nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}^t)\rangle}{\langle \Delta\tilde{\mathbf{X}}^t, \nabla_{\tilde{\mathbf{X}}} f_u(\Delta\tilde{\mathbf{X}}^t) + \mathbf{Y}\rangle}, \tag{8.16}$$

where $\nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}) = \mathbf{J} \circ \tilde{\mathbf{X}} - \mathbf{Y} + \upsilon\mathbf{L}\tilde{\mathbf{X}}\mathbf{D}_h\mathbf{D}_h^\mathsf{T}$ to solve (8.5), or $\nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}) = \mathbf{J} \circ \tilde{\mathbf{X}} - \mathbf{Y} + \upsilon(\mathbf{L}+\epsilon\mathbf{I})^\beta\tilde{\mathbf{X}}\mathbf{D}_h\mathbf{D}_h^\mathsf{T}$ to solve (8.15). Using μ, we have that $\tilde{\mathbf{X}}^{t+1} = \tilde{\mathbf{X}}^t + \mu\Delta\tilde{\mathbf{X}}^t$, where $\Delta\tilde{\mathbf{X}}^t = -\nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}^t) + \gamma\Delta\tilde{\mathbf{X}}^{t-1}$ and $\gamma = \frac{\|\nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}^t)\|_F^2}{\|\nabla_{\tilde{\mathbf{X}}} f_u(\tilde{\mathbf{X}}^{t-1})\|_F^2}$. The stopping condition is given either by achieving a maximum number of iterations or when:

$$\|\Delta\tilde{\mathbf{X}}^t\|_F \leq \delta, \tag{8.17}$$

where $\delta = 10^{-6}$ in the experiments for both (8.5) and (8.15).

Fig. 8-1 shows the pipeline of our algorithm applied to a graph with the regions in the world with confirmed cases of COVID-19 by November 18, 2020. The advantage

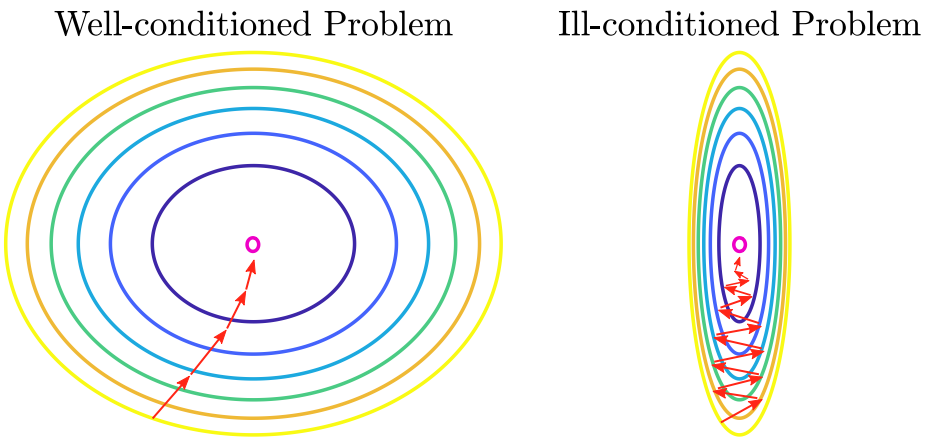

Figure 8-2: Contour plots of two error surfaces of well and ill-conditioned problems showing the evolution of a gradient descent method.

of the Sobolev minimization in (8.15) is its fast convergence with respect to the Laplacian minimization in (8.5). In the following section, we analyze some properties of the convergence rate of the optimization problems described in (8.5) and (8.15).

8.4.2 Convergence Rate

Intuitively, the convergence of a gradient descent method is faster when we have well-conditioned optimization problems. Fig. 8-2 shows a toy example of the convergence between well and ill-conditioned problems in the contour of a convex error surface. One can notice that the well-conditioned problem goes smoothly and faster to the global minimum, while the ill-conditioned problem has an erratic convergence. Similarly, the ill-conditioned problem can take longer to satisfy the stopping condition of certain optimization algorithms because of the erratic convergence.

Formally, the rate of convergence of a gradient descent method is at best linear. We can accelerate this rate if we reduce the condition number of the Hessian associated with the objective function of the problem [230].

Theorem 8.4.1. *Let* $\nabla^2_{\mathbf{z}} f_S(\mathbf{z}) = \mathbf{Q} + [\upsilon(\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes (\mathbf{L} + \epsilon\mathbf{I})^\beta]$ *and* $\nabla^2_{\mathbf{z}} f_L(\mathbf{z}) = \mathbf{Q} + [\upsilon(\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes \mathbf{L}]$ *be the Hessian associated with the Sobolev and Laplacian time-varying graph signal reconstruction problems in (8.15) and (8.5), respectively, where* $\mathbf{Q} = \operatorname{diag}(\operatorname{vec}(\mathbf{J})) \in \mathbb{R}^{MN \times MN}$ *and* $\mathbf{J} \neq \mathbf{0}$. *We have that:*

1. $0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \leq (\lambda_N + \epsilon)^{\beta} \lambda_{(D)N}$ *and* $0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \leq \lambda_N \lambda_{(D)N}$ *if* $\lambda_N, \lambda_{(D)N} \geq 1$, *where* $\lambda_{(D)N} = \lambda_{max}(\mathbf{D}_h \mathbf{D}_h^{\mathsf{T}})$.
2. *When* $\upsilon \to \infty$, $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$ *and* $\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \to \infty$.
3. *When* $\epsilon \to \infty$ *and* $\beta > 0$, *or when* $\beta \to \infty$ *and* $\lambda_N + \epsilon > 1$, $\lambda_{max}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$ *and so* $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$.

Proof: See Appendix I.

The main result of Theorem 8.4.1 is that the upper bound of the minimum eigenvalue for the Hessian associated with GraphTRSS is looser than the corresponding upper bound of TGSR in [26] for $\epsilon > 0$, $\beta > 1$, and $\lambda_N + \epsilon \geq 1$, which favors better condition numbers for the Sobolev problem. In practice, there is a range of values of ϵ and β where $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) < \kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$, and then the GraphTRSS converges faster than the problem described in (8.5). We compute the condition numbers of the Hessian of both GraphTRSS and TGSR in four real-world datasets in Section 8.6, Fig. 8-12 to show that indeed $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) < \kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$ in a large range of ϵ values. Theorem 8.4.1 also shows some intuitions on how to choose the parameters for GraphTRSS and TGSR regarding convergence rate in results 2) and 3). For example, we should keep small values of υ, ϵ, and β to avoid harming the convergence rate of these algorithms. Similarly, we should maintain small values of ϵ and β to get benefits in the convergence rate of GraphTRSS.

The final question is about the influence of parameter β in the Sobolev reconstruction algorithm in (8.15). Intuitively, the parameter β is changing the shape of the frequencies of $\mathbf{L}$. For simplicity, let us assume $\epsilon = 0$, and let us consider only the time-varying graph signal $\mathbf{X}$ without the temporal difference operator $\mathbf{D}_h$ in (8.15). Since the matrix of eigenvectors of $\mathbf{L}$ is an orthogonal matrix ($\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}$), we have:

$$\text{tr}(\mathbf{X}^{\mathsf{T}} \mathbf{L}^{\beta} \mathbf{X}) = \text{tr}(\hat{\mathbf{X}}^{\mathsf{T}} \mathbf{\Lambda}^{\beta} \hat{\mathbf{X}}) = \sum_{t=1}^{M} \sum_{i=1}^{N} \hat{\mathbf{x}}_t^2(i) \lambda_i^{\beta}, \tag{8.18}$$

where $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_M] = \mathbf{U}^{\mathsf{T}}\mathbf{X}$. Eqn. (8.18) is penalizing each Fourier term of each $\hat{\mathbf{x}}_t$ with powers of the eigenvalues of $\mathbf{L}$. For example, when $\beta = 1$, we penalize the higher frequencies stronger than the lower frequencies of each $\hat{\mathbf{x}}_t$, *i.e.*, a smooth

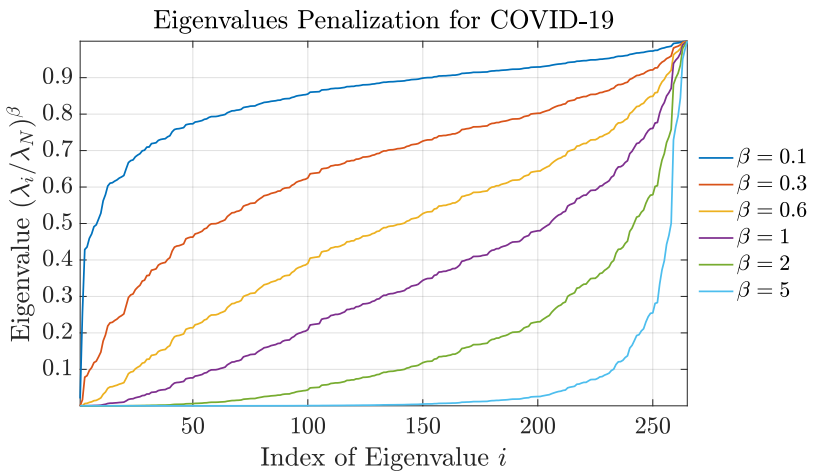

Figure 8-3: Eigenvalue penalization of the Laplacian matrix for different values of β from the dataset of COVID-19.

function in G. For $\beta > 1$, we also obtain a smooth function, but the penalization in the higher frequencies is more prominent, while for $\beta < 1$, we get more penalization in the lower frequencies. Fig. 8-3 shows the eigenvalues of $\mathbf{L}^\beta$ for several values of β, where we normalized the eigenvalues such that $(\lambda_i/\lambda_N)^\beta \ \forall \ i \in \mathcal{V}$ for visualization purposes.

There are two important aspects for choosing the parameters ϵ and β when using (8.15). From a graph topological point of view, we notice that adding $\epsilon\mathbf{I}$ to the Laplacian adds self-loops at the vertices in the graph with weights ϵ, and from Theorem 8.4.1, we notice that the value of ϵ should be small to avoid harming $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$. Therefore, it is reasonable to maintain a small value of ϵ so that the graph topology is not heavily modified and also to avoid harming $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$. Secondly, we also know from Theorem 8.4.1 that choosing a large value of β harms $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$. In practice, different datasets can benefit from specific penalization shapes like the ones in Fig. 8-3. As a result, we set $\beta = 1$ in the experiments to maintain a good $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ and be as fair as possible on the comparison with Qiu's method [26]. However, we do additional experiments where we see that different values of β can improve the performance of (8.15) in different datasets, *i.e.*, different datasets require different assumptions. Finally, we can conclude that TGSR by Qiu *et al.* [26] is a specific case of GraphTRSS when $\epsilon = 0$ and $\beta = 1$.

8.5 Experimental Framework

This section presents the datasets used in this chapter and the experimental framework details. We divide our experiments into three parts: 1) synthetic dataset, 2) COVID-19 datasets, and 3) environmental datasets. The graph G can be constructed based on the coordinate locations of the nodes in each dataset with k-NN algorithm. Let $\mathbf{M} \in \mathbb{R}^{N\times 2}$ be the matrix of coordinates of all nodes such that $\mathbf{M} = [\mathbf{m}_1, \ldots, \mathbf{m}_N]^\mathsf{T}$, where $\mathbf{m}_i \in \mathbb{R}^2$ is the vector with the latitude and the longitude of node i. The weights of each edge (i, j) are given by the Gaussian kernel:

$$\mathbf{W}(i,j) = \exp\left(-\frac{\|\mathbf{m}_i - \mathbf{m}_j\|_2^2}{\sigma^2}\right), \tag{8.19}$$

where $\|\mathbf{m}_i - \mathbf{m}_j\|_2^2$ is the Euclidean distance between the vertices i and j, and σ is the standard deviation given by $\sigma = \frac{1}{|\mathcal{E}|}\sum_{(i,j)\in\mathcal{E}} \|\mathbf{m}_i - \mathbf{m}_j\|_2$. The Gaussian kernel in (8.19) assigns higher weights to geographically-close locations and vice versa. The construction of the graph depends on the specific application, and GraphTRSS is not sensitive to the graph construction methods.

8.5.1 Datasets

GraphTRSS is evaluated on one synthetic dataset and four real-world datasets including, 1) a synthetic graph, 2) global COVID-19 cases [21], 3) USA COVID-19 cases [21], 4) mean concentration of Particulate Matter (PM) 2.5 [26], and 5) sea surface temperature [26].

Synthetic Graph and Signals

We use the synthetic dataset created by Qiu *et al.* [26], where 100 nodes are generated randomly from the uniform distribution in a 100×100 square area. Therefore, Qiu *et al.* [26] used a k-NN to construct the graph. The time-varying graph signal is generated with the recursive function $\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{L}^{-\frac{1}{2}}\mathbf{f}_t$, where: 1) $\mathbf{x}_1$ is a low-frequency graph signal with energy 10^4, 2) $\mathbf{L}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{U}^\mathsf{T}$, where $\mathbf{\Lambda}^{-\frac{1}{2}} =$

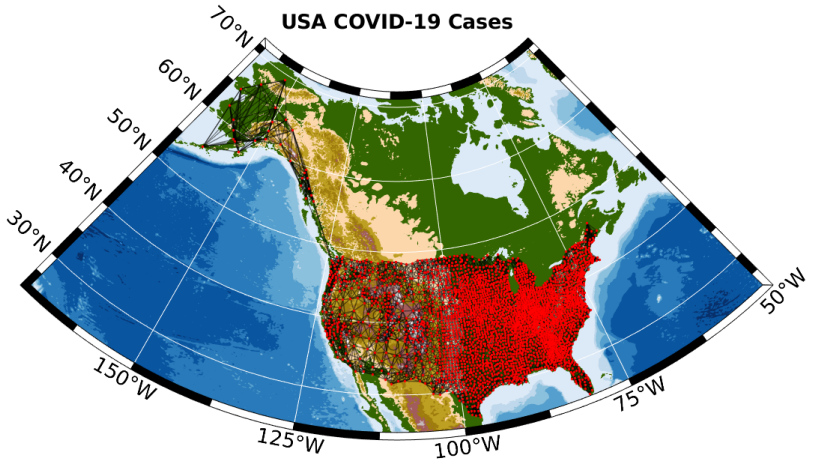

Figure 8-4: Graph with the places in the United States in the Johns Hopkins University dataset [21]. The graph was constructed with a k-NN with $k = 10$.

$\text{diag}(0, \lambda_2^{-\frac{1}{2}}, \ldots, \lambda_N^{-\frac{1}{2}})$, and 3) $\mathbf{f}_t$ is an i.i.d. white Gaussian signal such that $\|\mathbf{f}_t\|_2 = \alpha$. As a result, the synthetic time-varying graph signal satisfies Definition 8.3.1.

Global COVID-19

We use the global COVID-19 dataset provided by the Johns Hopkins University [21]. This dataset contains the cumulative number of daily COVID-19 cases for 265 locations in the world (Fig. 8-1 shows the locations in the dataset), and we use the data between January 22, 2020, and November 18, 2020; *i.e.*, 302 days. The graph is constructed with the k-NN method with $k = 10$ using the coordinates of each place [28].

USA COVID-19

We also use the USA COVID-19 dataset provided by the Johns Hopkins University [21]. Fig. 8-4 shows the map with the points of the graph in the USA COVID-19 dataset. This dataset is more fine-grained than the global set since the data contain 3232 localities in the US. We use the same temporal window and k-NN method as in the global dataset. The experiments related to the US can give us a different view of our algorithm since several travel restrictions were applied between countries in the pandemic, while inside countries, there was slightly more freedom to move around.

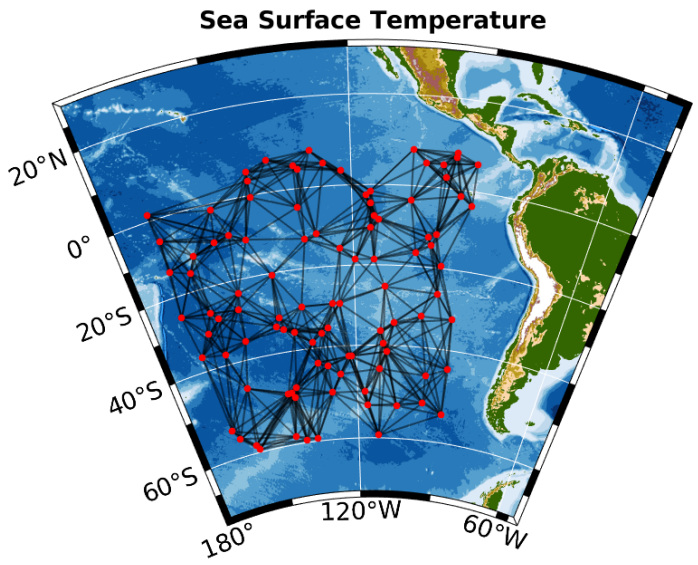

Figure 8-5: Graph with the spots in the sea for the dataset of temperature. The graph was constructed with a k-NN with $k = 10$.

Particulate Matter 2.5

In this chapter, we use the daily mean PM 2.5 concentration dataset from California provided by the US Environmental Protection Agency[2]. We use the data captured daily from 93 sensors in California for the first 220 days in 2015.

Sea Surface Temperature

We use the sea surface temperature captured monthly and provided by the NOAA Physical Sciences Laboratory (PSL) from their website[3]. In this chapter, we use the same experimental framework as in [26] for comparison purposes, *i.e.*, we use a subset of 100 points on the Pacific Ocean within a time frame of 600 months. Fig. 8-5 shows the locations of the spots in the sea for this dataset.

8.5.2 Evaluation Metrics

We use the Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and the Mean Absolute Percentage Error (MAPE) to compare GraphTRSS with the meth-

[2] `https://www.epa.gov/outdoor-air-quality-data`
[3] `https://psl.noaa.gov/`

ods of the literature. RMSE $= \sqrt{\frac{\sum_{i=1}^{N_x}(\hat{\mathbf{x}}_i - \mathbf{x}_i^*)^2}{N_x}}$, MAE $= \frac{\sum_{i=1}^{N_x}|\hat{\mathbf{x}}_i - \mathbf{x}_i^*|}{N_x}$, and MAPE $= \frac{1}{N_x}\sum_{i=1}^{N_x}|\frac{\mathbf{x}_i^* - \hat{\mathbf{x}}_i}{\mathbf{x}_i^*}|$, where $\hat{\mathbf{x}}$ is the recovered signal, $\mathbf{x}^*$ is the ground truth signal, and N_x is the length of the signal.

8.5.3 Experiments

For the synthetic dataset, we perform experiments analyzing several sampling densities, Signal-to-Noise Ratio (SNR) levels, smoothness α of the synthetic time-varying graph signal, and the condition number of $\nabla^2_{\mathbf{z}} f_S(\mathbf{z})$ and $\nabla^2_{\mathbf{z}} f_L(\mathbf{z})$. For the real dataset, we perform several experiments for reconstruction and forecasting. In the same way, we study the convergence, the ϵ and β parameters, and the temporal difference operators for GraphTRSS. We compare GraphTRSS with Natural Neighbor Interpolation (NNI) by Sibson [22], Graph Regularization (GR) by Narang *et al.* [23], Tikhonov regularization by Perraudin *et al.* [24,25], Time-varying Graph Signal Reconstruction (TGSR) by Qiu *et al.* [26], and Random Sampling and Decoder (RSD) by Puy *et al.* [27]. NNI [22] is based on Voronoi tessellation, where the interpolation is given by a linear combination of the neighbor points of the element to be interpolated in the Voronoi partition. GR [23] can be viewed as the solution of (8.5) without the time component. Tikhonov regularization [24] is based on joint stationarity, where the solution is constrained to be smooth in the graph and in time. RSD method [27] solves an optimization problem where we have smoothness in the graph and a probability distribution function for the sampling. The role of RSD and GR in the experiments is to see how these static graph signal methods compare to time-varying graph signal algorithms. Similarly, since RSD and GR cannot reconstruct graph signals where we do not have any sample at a specific time t, they are not included in some of the experiments as in forecasting. GR, Tikhonov, TGSR, and GraphTRSS have a maximum number of 20000 iterations in the optimization algorithm for a fair comparison. We optimize the parameters of each method in all experiments for a fair comparison.

In the reconstruction experiments, we adopt two sampling strategies. The first reconstruction experiment performs a random sampling on each temporal snapshots

for the number of nodes N, *i.e.*, we randomly select a percentage of nodes on each time step as in [26]. For RSD by Puy *et al.* [27], we use their optimal sampling procedure because their reconstruction algorithm depends on their specific sampling distribution. Puy *et al.* [27] proposed to estimate the reconstructed static graph signal $\mathbf{x}_{\text{rec}}$ by solving the following problem:

$$\mathbf{x}_{\text{rec}} = \arg\min_{\mathbf{z}} \|\mathbf{P}^{-\frac{1}{2}}(\mathbf{M}\mathbf{z} - \mathbf{x}(\mathcal{S}))\|_2 + \eta \mathbf{z}^{\mathsf{T}} g(\mathbf{L})\mathbf{z}, \tag{8.20}$$

where $\eta \in \mathbb{R}^+$, $g : \mathbb{R} \to \mathbb{R}$ is a non-negative and non-decreasing polynomial function, $\mathbf{M}$ is the sampling matrix, $\mathcal{S}$ is the set of sampled nodes, and $\mathbf{P} \in \mathbb{R}^{N \times N}$ is a random matrix that is designed jointly with $\mathbf{M}$. RSD approach samples nodes using the diagonal of $\mathbf{P}$ as the probability distribution where the nodes are drawn, while the sampling method we use for the other methods uses a uniform distribution. In the current chapter, we use the optimal computation of $\mathbf{P}$ that requires the spectral decomposition of $\mathbf{L}$. However, for large graphs, this optimal approach may be computationally prohibitive.

The second reconstruction experiment performs a random sampling of entire time-snapshots for the total number of graph signals M. Both experiments compute the error metrics for each method on the non-sampled nodes for a set of sampling densities $\mathcal{M}$. The set $\mathcal{M}$ was chosen according to each dataset, for example, $\mathcal{M} = \{0.5, 0.6, \ldots, 0.9, 0.995\}$ for COVID-19 datasets. Real graph signals tend to be approximately bandlimited instead of strictly bandlimited. The datasets we consider in this chapter are challenging because some of the graph signals contain high-frequency components. For example, Fig. 8-6 shows the graph Fourier transforms of some representative elements of $\mathbf{X}\mathbf{D}_h$ in COVID-19 global, where we use the Hadamard power $(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1})^{(2)} = (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}) \circ (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1})$ for visualization purposes. We compute the bandwidth of each $\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}$ as the index where we have 90% of the spectral energy. Finally, we compute the average of all bandwidths for $\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}$ $\forall$ $1 < i \leq N$. We get an average bandwidth of 155.73 for COVID-19 global. 155.73 is 58.77% of the total amount of nodes. Therefore, we could expect

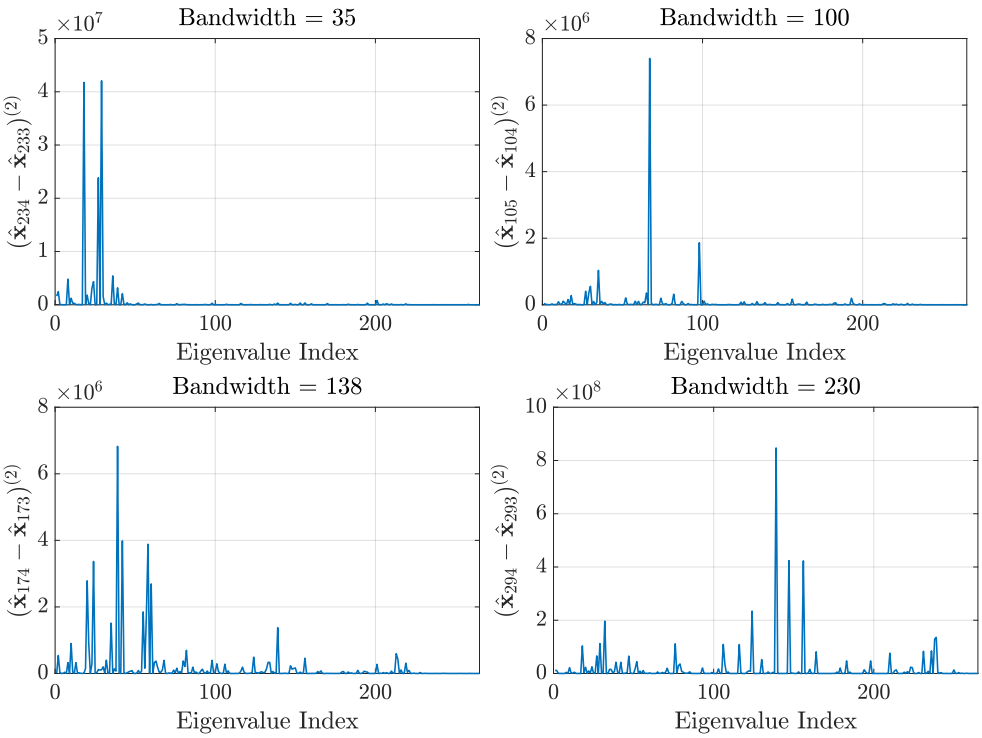

Figure 8-6: Graph Fourier transform of some elements of $\mathbf{X}\mathbf{D}_h$ for COVID-19 global dataset.

good reconstruction errors after this sampling density on average. However, notice that real-world datasets are not strictly bandlimited (as shown in Fig. 8-6), and we have noise, so perfect reconstruction is not possible.

We evaluate each method with a Monte Carlo cross-validation with 100 repetitions for each sampling density (50 repetitions for the COVID-19 USA dataset). Deterministic sampling methods do not work well with these reconstruction methods as stated in Section 8.3.

In the forecasting experiment, we compute the error metrics for several temporal snapshots t in the set $\{1, 2, \ldots, 10\}$. For example, since COVID-19 datasets are sampled daily, we will predict the new COVID-19 cases in the last day, in the two last days, and so on until the last ten days. However, for the sea surface temperature, we will predict up to ten months since this dataset is sampled monthly, *i.e.*, we are interpolating multiple time steps ahead in all datasets.

We perform some studies to analyze the parameters of GraphTRSS. The first of these studies computes the average RMSE (in $\mathcal{M}$) and the average number of

iterations required to satisfy the stopping condition with variations in ϵ. The second study performs reconstruction with the three temporal difference operators (one, two, and three steps). The last study computes reconstruction with several values of β. These studies are evaluated with a Monte Carlo cross-validation with ten repetitions.

We also compare the loss function vs the iteration number for TGSR and GraphTRSS in reconstruction, where the loss is the evaluation of (8.5) and (8.15) at each iteration. In this case, 50 repetitions for each sampling densities in $\mathcal{M}$ are performed, where each repetition is computed with the best parameters of each method. For a fair comparison, GraphTRSS and Qiu's method use the same sampling matrix $\mathbf{J}$. We also compute the running times for several sampling densities. Finally, we compute $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ and $\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$ for $\beta = 1$ and several values of ϵ in all real datasets. All experiments were executed in MATLAB R2017b on a 2.3GHz MacBook Pro with 16GB memory. The code of GraphTRSS has been made available[4].

8.6 Results and Discussion

This section presents the results of the experiments on the selected datasets. GR and RSD methods cannot reconstruct entire temporal snapshots because these algorithms only use spatial information. Therefore, GR and RSD are not included in the experiments of entire snapshots sampling and forecasting. A detailed analysis for each dataset is provided as follows.

8.6.1 Synthetic Graph and Signals

Fig. 8-7 shows the results in the synthetic dataset. GraphTRSS performs better than the existing methods for all values in $\mathcal{M}$ for the random sampling scheme, as shown in Fig. 8-7(a). Our algorithm is also more robust against noise than the compared methods, as shown in Fig. 8-7(b). Similarly, Fig. 8-7(c) shows the benefits of GraphTRSS regarding different levels of smoothness α as given in Definition 8.3.1. Our algorithm shows better performance than other time-varying graph signal methods as TGSR and

[4] `https://github.com/jhonygiraldo/GraphTRSS`

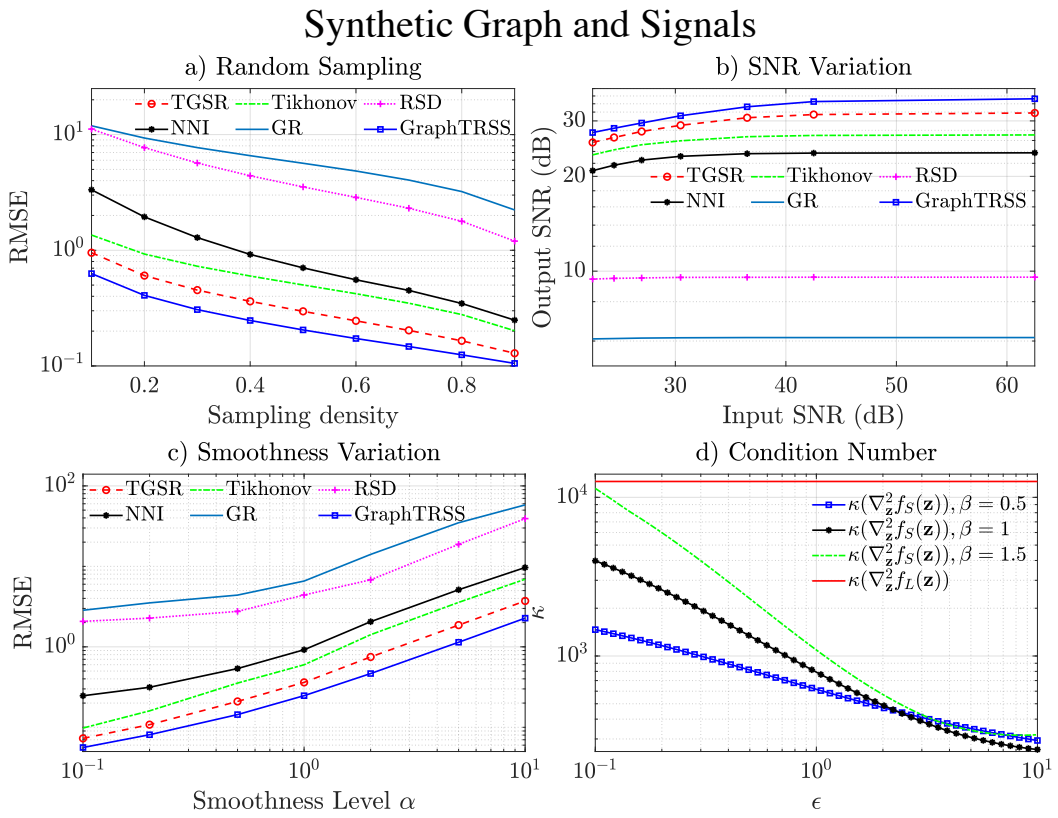

Figure 8-7: Comparison of GraphTRSS with several methods in the literature on synthetic data for four experiments on: a) reconstruction with several sampling densities, b) variation of the SNR, c) variation of the smoothness level in (8.4), and d) $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ and $\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$ with several values of ϵ and β. Our algorithm is compared with Natural Neighbor Interpolation (NNI) [22], Graph Regularization (GR) [23], Tikhonov regularization [24, 25], Time-varying Graph Signal Reconstruction (TGSR) [26], and Random Sampling and Decoder (RSD) [27].

other static graph signal reconstruction schemes as GR. Finally, Fig. 8-7(d) shows the condition numbers of the Hessian associated with GraphTRSS ($\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$) and TGSR ($\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$) with several values of ϵ and β. As noted from Theorem 8.4.1, we have that: 1) there is a range of values of $\epsilon > 0$ where $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) < \kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$, 2) $\epsilon > 0$ promotes good values of the condition number $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ for GraphTRSS, and 3) big values of β harm the condition number $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ of GraphTRSS.

8.6.2 Real Datasets Summary

Table 8.1 shows the summary of all error metrics for random, entire snapshots sampling, and forecasting in the real datasets. The results are computed as the mean

Table 8.1: Summary of the average error metrics in the real datasets for several sampling schemes. The best and second-best performing methods on each category are shown in **red** and **blue**, respectively.

Method	Random Sampling											
	COVID-19 Global			COVID-19 USA			Sea Surface Temperature			PM 2.5 Concentration		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
NNI (Sibson [22])	1555.14	189.15	**4.23**	54.92	10.90	2.17	**0.77**	**0.56**	**0.07**	4.95	2.96	0.59
GR (Narang *et al.* [23])	4744.25	1119.07	90.95	65.49	13.94	3.47	2.43	1.76	0.40	5.37	3.32	0.67
Tikhonov (Perraudin *et al.* [24])	1253.71	**183.60**	6.07	37.00	**6.03**	**1.03**	0.95	0.70	0.12	4.32	2.66	0.55
TGSR (Qiu *et al.* [26])	**1136.39**	213.96	11.30	**33.51**	6.14	1.19	**0.36**	**0.26**	**0.03**	**3.90**	**2.28**	**0.39**
RSD (Puy *et al.* [27])	2045.37	506.30	36.17	58.59	12.05	2.50	5.56	4.62	0.97	5.43	3.46	0.72
GraphTRSS (ours)	**1134.15**	**152.76**	**2.41**	**33.47**	**5.96**	**1.10**	**0.36**	**0.26**	**0.03**	**3.83**	**2.21**	**0.38**
Method	Entire Snapshots Sampling											
	COVID-19 Global			COVID-19 USA			Sea Surface Temperature			PM 2.5 Concentration		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
NNI (Sibson [22])	1342.08	**170.64**	**3.40**	56.69	11.71	2.60	**0.98**	**0.68**	**0.08**	10.87	4.44	0.85
Tikhonov (Perraudin *et al.* [24])	**1255.05**	182.97	5.94	36.96	**6.12**	**1.04**	1.17	0.86	0.13	**4.54**	**2.81**	**0.56**
TGSR (Qiu *et al.* [26])	1459.45	788.10	100.06	**36.38**	13.75	4.74	19.19	19.15	2.13	10.94	9.31	1.49
GraphTRSS (ours)	**1114.16**	**143.69**	**1.49**	**32.96**	**5.45**	**0.88**	**0.94**	**0.67**	**0.07**	**4.42**	**2.70**	**0.48**
Method	Forecasting											
	COVID-19 Global			COVID-19 USA			Sea Surface Temperature			PM 2.5 Concentration		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
NNI (Sibson [22])	16722.30	2371.30	**6.43**	904.23	283.23	19.54	3.04	2.23	**0.23**	30.71	17.15	2.76
Tikhonov (Perraudin *et al.* [24])	3612.06	**1190.20**	30.48	99.88	**31.2**	**2.03**	**1.94**	**1.53**	0.30	**4.80**	**3.23**	**0.67**
TGSR (Qiu *et al.* [26])	**3313.50**	2472.08	238.65	**91.5**	54.36	6.88	18.72	18.69	2.39	10.09	9.11	1.27
GraphTRSS (ours)	**2416.30**	**583.36**	**4.80**	**75.33**	**20.65**	**0.86**	**1.20**	**0.98**	**0.13**	**4.36**	**2.83**	**0.54**

of all results over the set $\mathcal{M}$. GraphTRSS shows the best performance in almost all cases against the other methods.

8.6.3 COVID-19 Datasets

Fig. 8-8 and 8-9 show the results of the experiments in the COVID-19 datasets. GraphTRSS is better than the other methods for the reconstruction experiments when using different sampling strategies. We can notice that the results in the COVID-19 USA dataset have lower RMSE than the global dataset, *i.e.*, we can get better estimations if we have more fine-grained data as in the case of the COVID-19 USA dataset. From the experiment with entire temporal snapshots, we can notice that TGSR method [26] performs poorly compared to the other methods for what we observe in random sampling. This behavior is expected because entire snapshots sampling does not satisfy the second condition of Theorem 8.3.1 for TGSR by Qiu *et al.* [26].

All tested methods, including ours, are designed to reconstruct time-varying graph signals, and therefore these methods do not have prior assumptions about the behavior in the time domain of the underlying processes. Correspondingly, in forecasting, we

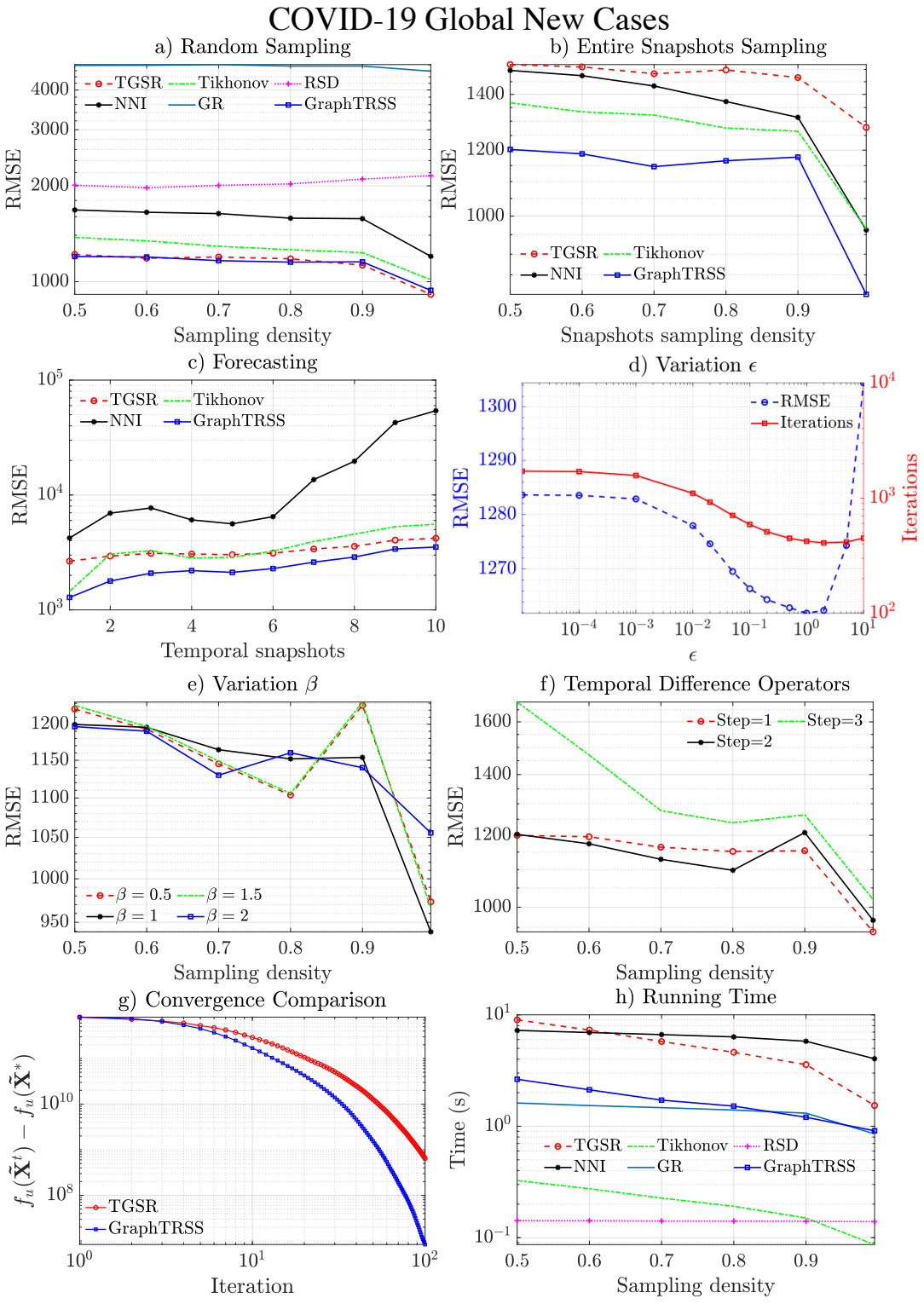

Figure 8-8: Comparison of GraphTRSS with several methods in the literature on COVID-19 global dataset for several experiments in terms of: a) random sampling, b) entire snapshots sampling, c) forecasting, d) variation of parameter ϵ, e) variation of parameter β, f) several temporal difference operators, g) convergence comparison, and h) running time.

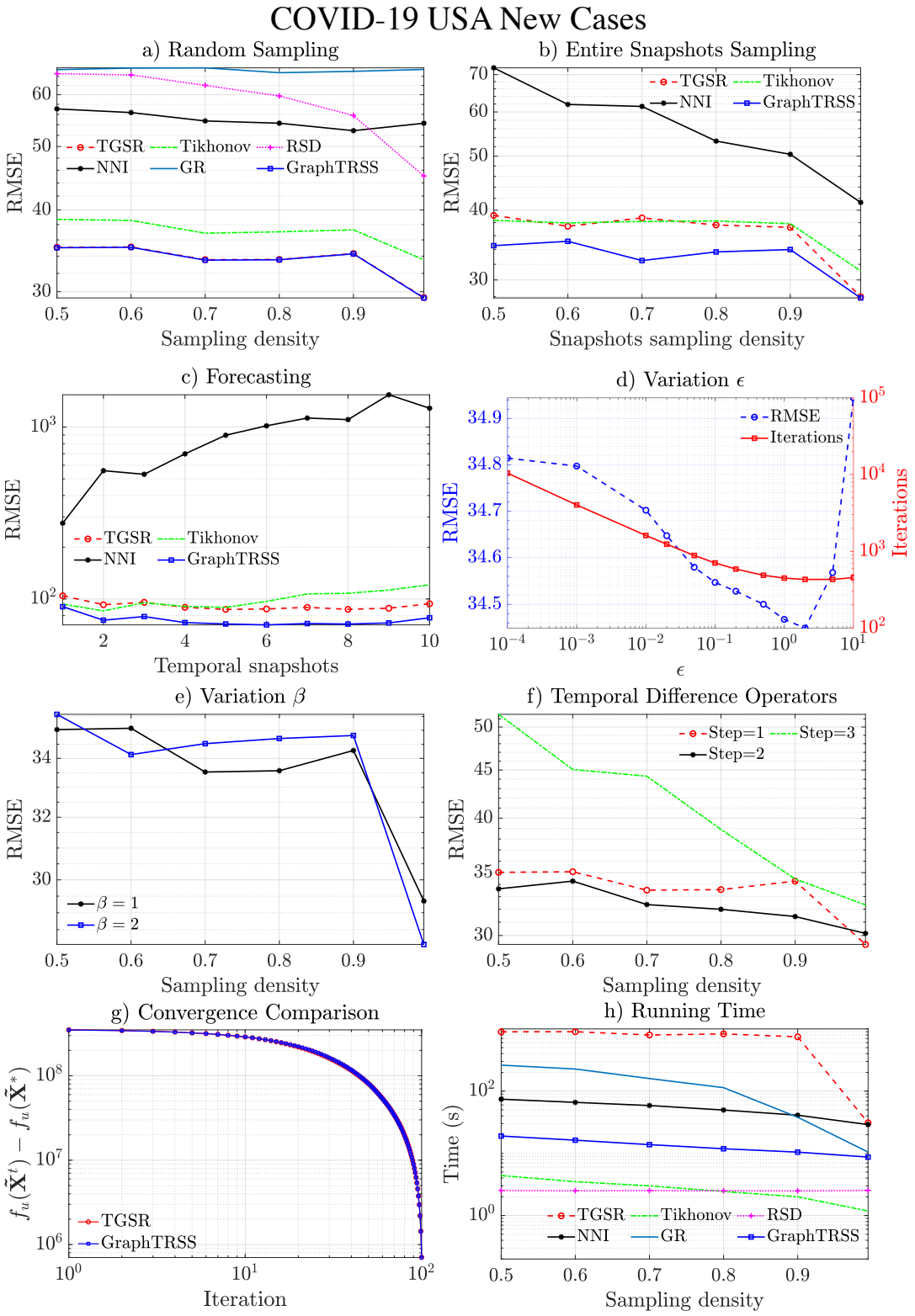

Figure 8-9: Comparison of GraphTRSS with several methods in the literature on COVID-19 USA dataset for several experiments in terms of: a) random sampling, b) entire snapshots sampling, c) forecasting, d) variation of parameter ϵ, e) variation of parameter β, f) several temporal difference operators, g) convergence comparison, and h) running time.

get weaker results than the performance shown in Fig. 8-8 and 8-9 for random and entire snapshots sampling. However, our algorithm readily shows better results than the other methods in both COVID-19 datasets, even forecasting multiple time steps ahead. Thus, the Sobolev function introduced in Definition 8.4.1 could potentially improve other graph-based forecasting methods of the literature [231], using proper time-domain prior assumptions or other machine learning strategies, which we leave for future work.

From the studies to analyze the parameters of GraphTRSS, one can notice that values of $\epsilon > 0$ improve the condition number; this is reflected in the number of iterations to satisfy the stopping condition as shown in Fig. 8-8(d) and 8-9(d) for COVID-19. However, one should be careful since big values of ϵ can heavily modify the structure of the graph leading to a degradation of the performance, as shown in Fig. 8-8(d) and 8-9(d) for the RMSE. For the variations of β and the temporal difference operators, we notice that values different from $\beta = 1$ and one time-step for the temporal difference operator might bring benefits for COVID-19 datasets.

Fig. 8-8(g) shows that GraphTRSS converges faster than TGSR method for COVID-19 global, where $\tilde{\mathbf{X}}^*$ is the solution of the optimization problem, and $\tilde{\mathbf{X}}^t$ is the solution at iteration t. Fig. 8-8(h) and 8-9(h) show the running time for each method. These experiments were computed with the best parameters of each method from the random sampling experiment. Arguably, GraphTRSS shows the best compromise between accuracy and running time among several methods for COVID-19 datasets, as shown in Fig. 8-8 and 8-9.

8.6.4 Environmental Datasets

Fig. 8-10 and 8-11 show the results for the experiments in the environmental datasets. The analysis of the comparison between our algorithm and the other methods is mostly similar to the analysis for Fig. 8-8 and 8-9, *i.e.*, GraphTRSS shows the best compromise between accuracy and running time. However, we should notice two interesting results in Fig. 8-10 and 8-11. Firstly, we notice that the parameters $\beta = 1.5$ or $\beta = 2$ show better results than $\beta = 1$ in the environmental datasets. Secondly,

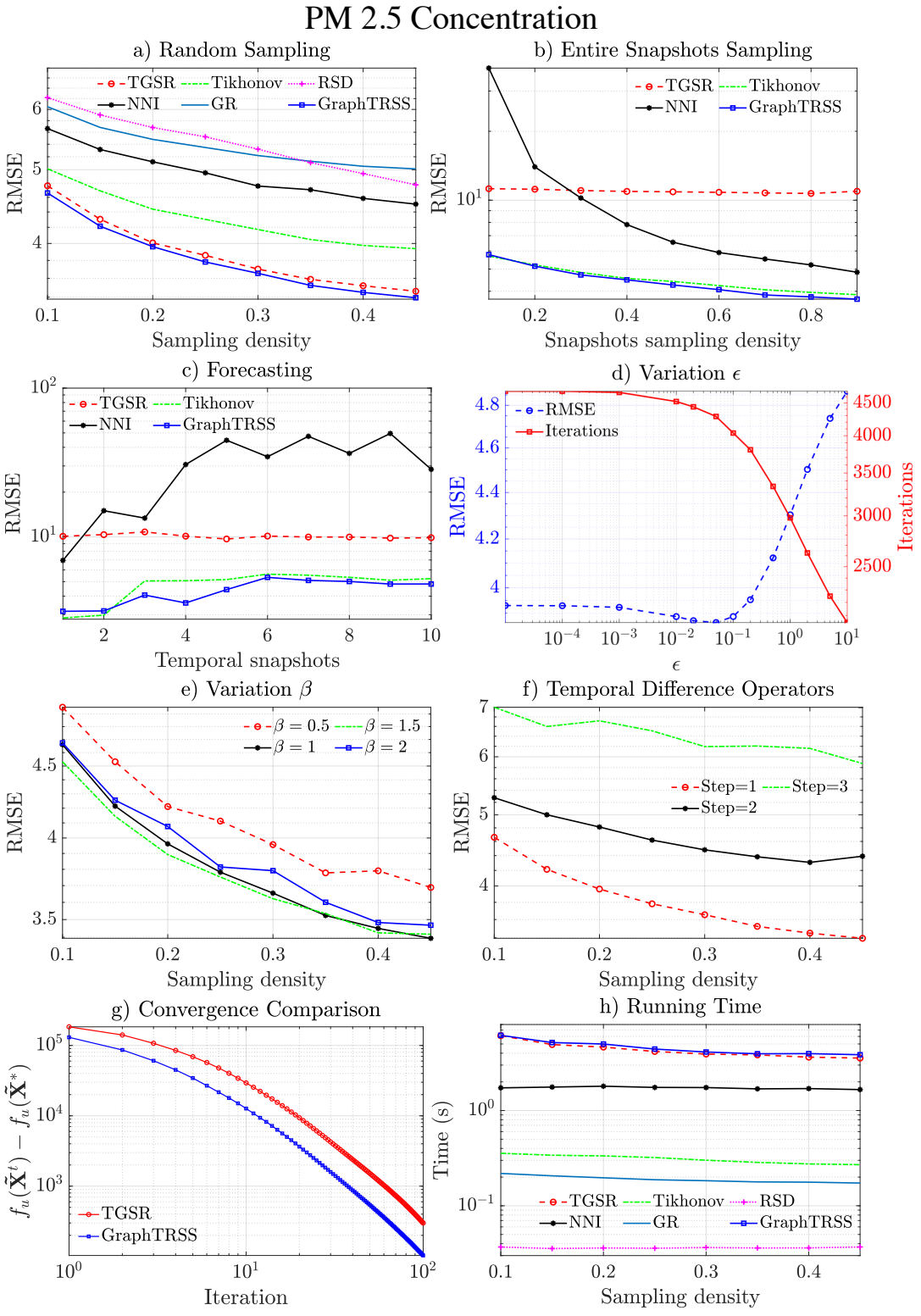

Figure 8-10: Comparison of GraphTRSS with several methods in the literature in the PM 2.5 dataset for several experiments in terms: a) random sampling, b) entire snapshots sampling, c) forecasting, d) variation of parameter ϵ, e) variation of parameter β, f) several temporal difference operators, g) convergence comparison, and h) running time.

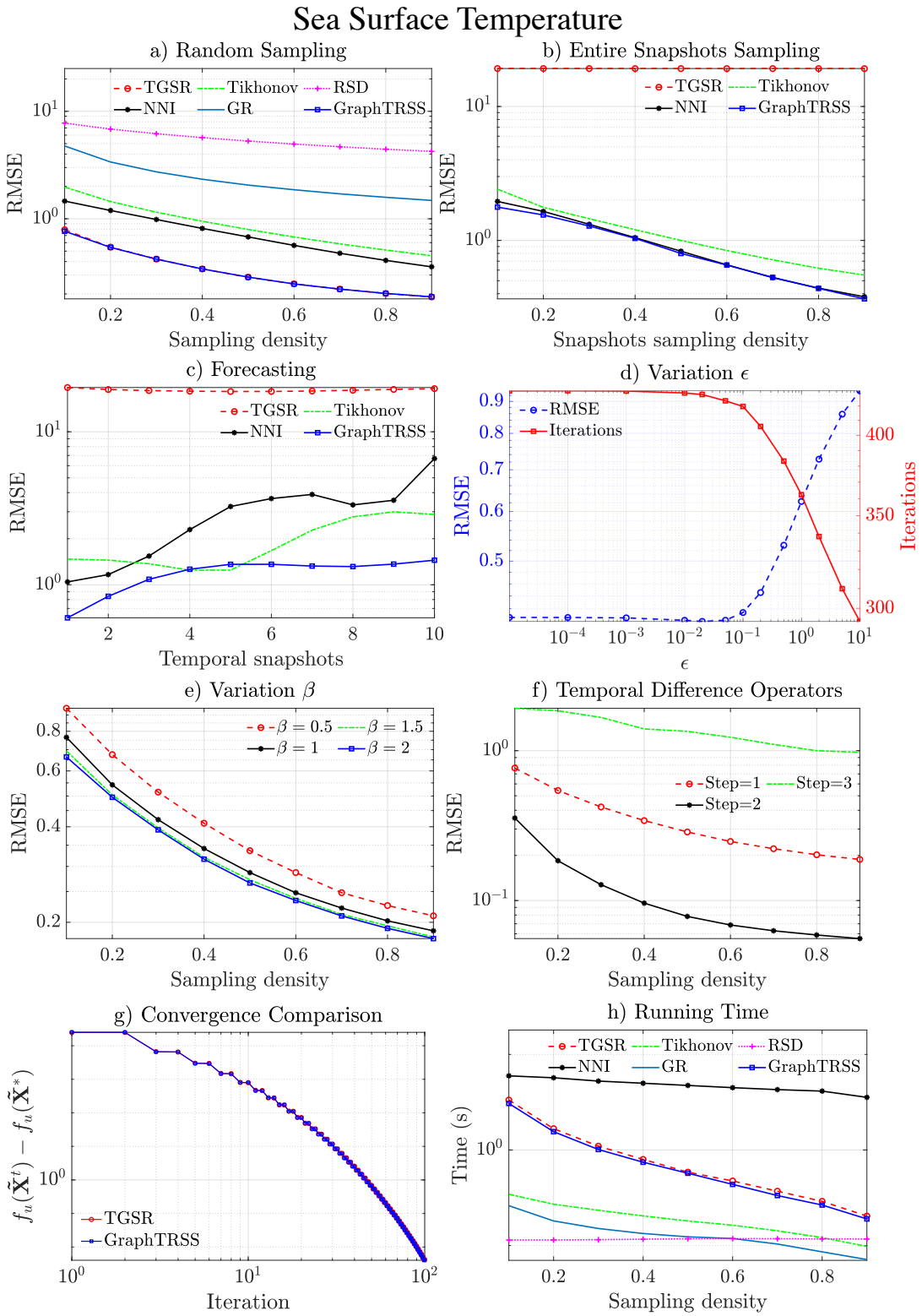

Figure 8-11: Comparison of GraphTRSS with several methods in the literature in the sea surface temperature dataset for several experiments in terms of: a) random sampling, b) entire snapshots sampling, c) forecasting, d) variation of parameter ϵ, e) variation of parameter β, f) several temporal difference operators, g) convergence comparison, and h) running time.

Table 8.2: The average number of iterations to satisfy the stopping condition for GraphTRSS and TGSR [26].

Method	COVID-19 Global	COVID-19 USA	PM 2.5 Conc.	Sea Surface Temp.
TGSR [26]	1735.9	48659	4690.1	418.8
GraphTRSS	510.0	427	4266.6	416.0

the two-time steps temporal operator $\mathbf{D}_{h2}$ shows a big performance improvement compared to the one-step temporal operator for the sea surface temperature dataset. Perhaps, we could propose other temporal operators and more elaborated optimization functions based on several β parameters, which we leave for future work.

8.6.5 Additional Analysis

Table 8.2 shows the number of iterations that TGSR method and GraphTRSS require to satisfy the stopping condition in (8.17). Table 8.2 shows that GraphTRSS satisfies the stopping condition around 3.4 times faster than TGSR in global COVID-19; and about 114 times faster in USA COVID-19. We limit the maximum number of iterations of TGSR to 60000 for the USA COVID-19 dataset because of time limitations. The experiments with the lowest sampling densities would take more time to satisfy the stopping condition if we increase the maximum number of iterations for TGSR method in COVID-19 USA. This behavior is another undesirable effect of bad condition numbers.

Fig. 8-12 shows the condition number of the Hessian functions for the COVID and environmental datasets with random sampling with density 0.5. The condition number of the Hessian of (8.5) is not displayed in Fig. 8-12 for COVID-19 USA because, in that case, $\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \to \infty$. Furthermore, as predicted by Theorem 8.4.1: 1) there is a range of values of ϵ where $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) < \kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$ and then we get benefits in convergence rate, and 2) $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$ grows quickly for large values of ϵ. Notice that the condition numbers of $\nabla^2_{\mathbf{z}} f_S(\mathbf{z})$ and $\nabla^2_{\mathbf{z}} f_L(\mathbf{z})$ depend on the specific eigenvalues of the graphs we are addressing. As a consequence, it is reasonable to use the normalized Laplacian matrix $\mathbf{\Delta} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ since $\lambda_{max}(\mathbf{\Delta}) \leq 2$. The conclusions of Theorem 8.4.1 still hold with the normalized Laplacian matrix because now

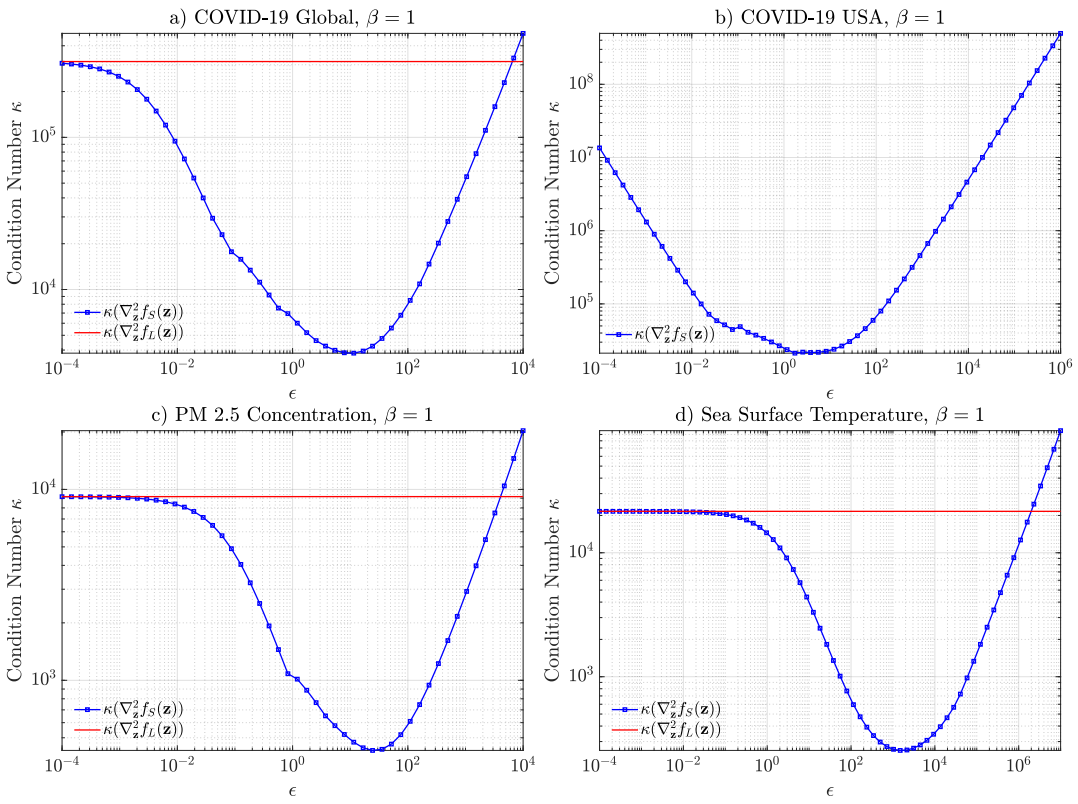

Figure 8-12: Condition number of the Hessian associated with the optimization problems for GraphTRSS ($\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z}))$) and TGSR [26] ($\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$) for all datasets.

$\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes \mathbf{\Delta}) \leq 2\lambda_{(D)N}$ and $\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes (\mathbf{\Delta} + \epsilon\mathbf{I})^\beta) \leq (2+\epsilon)^\beta \lambda_{(D)N}$ instead of $\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes \mathbf{L}) = \lambda_N \lambda_{(D)N}$ and $\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes (\mathbf{L} + \epsilon\mathbf{I})^\beta) = (\lambda_N + \epsilon)^\beta \lambda_{(D)N}$ for $\lambda_{max}(\mathbf{L}), \lambda_{max}(\mathbf{\Delta}), \lambda_{(D)N} \geq 1$. However, notice that the eigenvalue spreading is smaller if $\lambda_{max}(\mathbf{\Delta}) < \lambda_{max}(\mathbf{L})$, *i.e.*, we can have better convergences rates in both GraphTRSS and TGSR method [26]. Fig. 8-13 shows a small experiment where we compared GraphTRSS and TGSR method [26] when using normalized ($\mathbf{\Delta}$) and unnormalized ($\mathbf{L}$) Laplacian matrices. For the comparison with $\mathbf{\Delta}$ and $\mathbf{L}$, we notice that in both datasets and for both GraphTRSS and TGSR, the convergence rate is faster when using the normalized Laplacian as shown in Fig. 8-13(b) for both COVID-19 and sea surface temperature datasets. However, GraphTRSS is still faster than TGSR in the COVID-19 dataset when using the normalized Laplacian since the conclusions of Theorem 8.4.1 still hold.

Notice that the improvement in the condition number depends on the specific

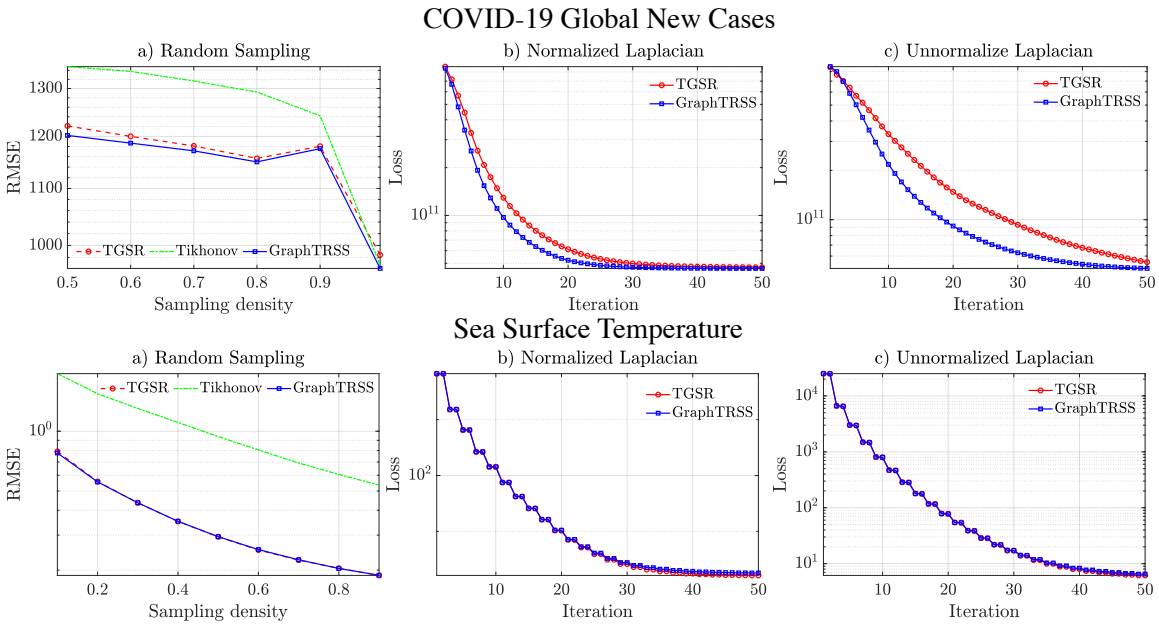

Figure 8-13: Normalized $\mathbf{\Delta}$ and unnormalized $\mathbf{L}$ Laplacian experiments in COVID-19 global and sea surface temperature, a) reconstruction error using $\mathbf{\Delta}$, b) loss vs number of iterations using $\mathbf{\Delta}$, and c) loss vs number of iterations using $\mathbf{L}$.

structure of each graph. Therefore, using the normalized Laplacian $\mathbf{\Delta} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ is reasonable since $\lambda_{max}(\mathbf{\Delta}) \leq 2$. The interested readers are referred to Section VI in the supplementary material for a small experiment with normalized Laplacian matrices.

The parameters ϵ and υ are optimized for each dataset to get good performance. However, we can notice in Fig. 8-12 that an ϵ value close to 10^{-1} is a good starting point since it improves the condition number, and it does not modify heavily the Laplacian matrix. Similar analysis about the parameter υ can be found in [26]. In practice, the parameters of GraphTRSS can be optimized in a small part of the dataset.

8.7 Conclusions

In this chapter, a new algorithm called GraphTRSS is introduced for the reconstruction of time-varying graph signals. The Sobolev norm is used to define a smoothness function for time-varying graph signals, and therefore this function is used to intro-

duce a new optimization problem for reconstruction from samples. We showed the good convergence properties of GraphTRSS by relying on the condition number of the Hessian associated with our problem, as shown in Theorem 8.4.1. Our algorithm improves convergence rate without relying on expensive matrix inversions or eigenvalue decompositions. GraphTRSS shows promising results for the estimation of new COVID-19 cases in both global and USA datasets, as well as in the reconstruction of environmental variables (PM 2.5 concentration and sea surface temperature). GraphTRSS could be useful in several scenarios, for example, 1) one could try to estimate the number of new COVID-19 cases where we have missing or unreliable information, 2) we can use our algorithm as a pre-processing step to get more reliable data for further forecasting of new cases with other well-established models of infectious diseases [232], and 3) we can also use GraphTRSS as a pre-processing step for further forecasting of new COVID-19 cases with other graph-based techniques such as [231]. GraphTRSS is also evaluated directly on forecasting, showing promising results.

This chapter shows several insights for possible future directions. For example, our algorithm and other GSP techniques could potentially increase the performance of various models in infectious disease (not only for COVID-19) for forecasting and imputation of data, among others. In the same way, this chapter could be the foundation for applying other GSP-based methods in virology. These methods can be, for example, graph neural networks [33, 171] and learning graphs from data [167].

Chapter 9

Conclusions

We have studied several problems in image and video processing, graph neural networks, and time-varying graph signals. This thesis has been divided into three main parts: 1) computer vision in Chapters 3-5, 2) machine learning in Chapters 6 and 7, and 3) signal processing in Chapter 8. Specifically, addressed some fundamental questions like: 1) the sample complexity of semi-supervised algorithms based on graph in Chapter 3, 2) the definition of sparse norms for GNNs in Chapter 6, 3) the relationship between over-smoothing and over-squashing in Chapter 7, and 4) the convergence rate of graph reconstruction algorithms in Chapter 8. Similarly, we have addressed the applications of: 1) moving object segmentation, 2) semantic segmentation, 3) cancer detection in images, 4) text classification, 5) action recognition, 6) node classification in graphs, and 7) reconstruction of time-varying graph signals for COVID-19 and environmental data. We proposed several algorithms that either: 1) reached state-of-the-art performances at the time of publication, or 2) explained certain important phenomena in data science.

Even though there is a big variety in the topics addressed in this thesis, most of them can be motivated by the desire of solving problems in computer vision, machine learning, or signal processing using limited supervision. To that end, we have used concepts from spectral graph theory, graph signal processing, graph neural networks, and also differential geometry specifically in Chapter 7. We summarize several ideas for future work that can be derived from this thesis as follows:

1. How concepts of graph signal processing can contribute to the field of computer vision, specifically in the problems of moving object segmentation, semantic segmentation, and object tracking to name a few.
2. How to design inductive graph learning methods to address computer vision problems in real-time.
3. How can we train convolutional neural networks and graph neural networks jointly for computer vision.
4. How architectural choices in graph neural networks can impact the performance in computer vision applications.
5. What is the performance of graph neural network architectures based on sparse norms in mainstream problems such as drug discovery or other computer vision problems.
6. How can we further propose novel functions in the adjacency matrix to improve the expressiveness of graph neural networks maintaining sparsity.
7. How can we solve the question if we really need and when we need deep graph neural networks.
8. How graph signal processing concepts can improve the performance of various models in infectious diseases and environmental data for forecasting and data imputation.

Appendix A

Instance Segmentation

The basic building blocks of the instance segmentation methods tested in Chapters 3 and 4 are: Mask R-CNN [20], Cascade Mask R-CNN [105], ResNet [106], ResNeSt [107], and Feature Pyramid Network (FPN) [233]. Mask R-CNN builds upon Faster R-CNN [108] by adding a branch for predicting an object mask in parallel with the already existing Faster R-CNN network for bounding box recognition. Furthermore, Cascade Mask R-CNN [105] builds upon Mask R-CNN by adding a sequence of detectors. Mask R-CNN contains: 1) a CNN for image feature extraction, 2) a region proposal layer, 3) ROI alignment, and 4) fully connected layers in parallel with convolutional layers to perform bounding box recognition and mask prediction, respectively. Figure A-1 shows the architecture of Mask R-CNN, where the CNN layer is replaced by a ResNet with 50 layers (ResNet-50) with FPN. In the case of Cascade Mask R-CNN, the CNN layer is replaced with a ResNeSt of 200 layers (ResNeSt-200). These residual networks are a special architecture of CNN that uses skip connections, this structure was motivated to avoid the problem of vanishing gradients [106]. For simplicity, let $f(\cdot)$ be a function representing a layer of a neural network, and let $\mathbf{S}$ be a matrix of certain dimensions representing the input of $f(\cdot)$, the skip connection in residual networks can be denoted by $f(\mathbf{S}) + \mathbf{S}$, *i.e.*, a residual unit adds the input of $f(\cdot)$ at the output of that layer.

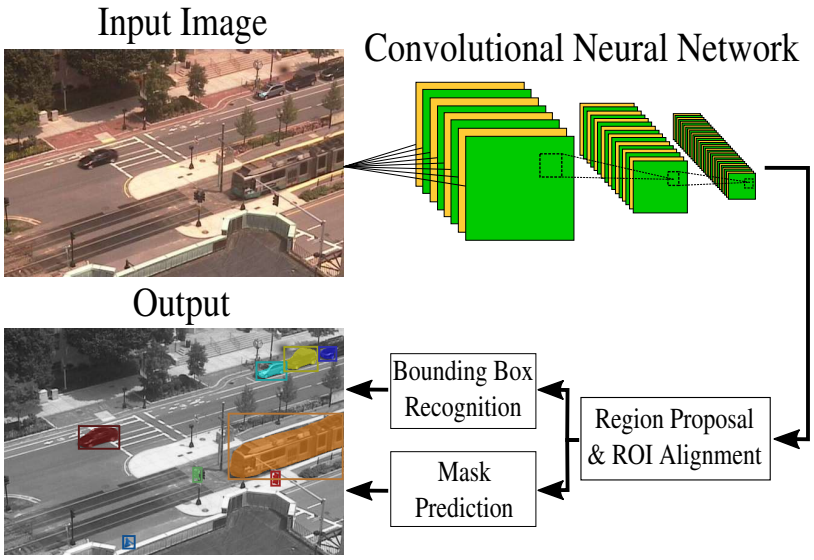

Figure A-1: Architecture of Mask R-CNN [20] for GraphMOS.

Appendix B

Vector of Features

We introduced in Section 3.3.2 the matrices representing the intensity and background images $\mathbf{I}_v^t$, $\mathbf{I}_v^{t-1}$, and $\mathbf{B}_v$ as well as the set of indices corresponding to the v-th segmented region $\mathcal{P}_v$ and the optical flow vectors of the current frame with support in the set of indices $\mathcal{P}_v$ for the horizontal and vertical direction $\mathbf{v}_x^t(\mathcal{P}_v)$ and $\mathbf{v}_y^t(\mathcal{P}_v)$. The vector $\mathbf{x}_v$ is obtained with the following steps:

1. The Local Binary Patterns (LBP) [110] are computed on the matrices: $\mathbf{I}_v^t$, $\mathbf{I}_v^{t-1}$, $\mathbf{B}_v$, and $|\mathbf{I}_v^t - \mathbf{B}_v|$. The LBP for each matrix is computed using 8 neighbors and a radius of 1 for each pixel in the input image, a linear interpolation to compute pixel neighbors, and finally a ℓ_2 normalization for each LBP cell histogram. For each matrix, a vector $\mathbb{R}^{1\times 59}$ is obtained for LBP representation. Let $f_{LBP} : \mathbf{A} \to \mathbb{R}^{1\times 59}$ be the function that represents the process of LBP feature extraction, the final LBP features $\mathbf{lbp}_v \in \mathbb{R}^{1\times 236}$ for a node v is given as $[f_{LBP}(\mathbf{I}_v^t), f_{LBP}(\mathbf{I}_v^{t-1}), f_{LBP}(\mathbf{B}_v), f_{LBP}(|\mathbf{I}_v^t - \mathbf{B}_v|)]$.
2. The vector of orientations $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{P}_v|}$ and the vector of magnitudes $\mathbf{m} \in \mathbb{R}^{|\mathcal{P}_v|}$ of the optical flow are computed as follows:

$$\boldsymbol{\theta} = \arctan\left(\mathbf{v}_y^t(\mathcal{P}_v) \oslash \mathbf{v}_x^t(\mathcal{P}_v)\right), \tag{B.1}$$

$$\mathbf{m} = \sqrt{\mathbf{v}_x^t(\mathcal{P}_v) \circ \mathbf{v}_x^t(\mathcal{P}_v) + \mathbf{v}_y^t(\mathcal{P}_v) \circ \mathbf{v}_y^t(\mathcal{P}_v)}, \tag{B.2}$$

where $\oslash$ represents the Hadamard division between matrices. A histogram of 49

bins is computed between $-\pi$ and π for $\boldsymbol{\theta}$, and between 0 and 30 for $\mathbf{m}$. Let $f_{OF} : \mathbf{a} \rightarrow \mathbb{R}^{1\times 49}$ be the function representing the histogram extraction process for the optical flow, then the whole representation of the motion $\mathbf{mov}_v \in \mathbb{R}^{1\times 109}$ of a node v is obtained as follows $[f_{OF}(\boldsymbol{\theta}), f_{OF}(\mathbf{m}), \max(\mathbf{m}), \bar{\mathbf{m}}, \max(\mathbf{m}) - \min(\mathbf{m}), \text{std}(\mathbf{m}), \text{mad}(\mathbf{m}), \min(\boldsymbol{\theta}), \max(\boldsymbol{\theta}), \bar{\boldsymbol{\theta}}, \max(\boldsymbol{\theta}) - \min(\boldsymbol{\theta}), \text{std}(\boldsymbol{\theta}), \text{mad}(\boldsymbol{\theta})]$, where $\bar{\mathbf{a}}$, $\text{std}(\mathbf{a})$, and $\text{mad}(\mathbf{a})$ are the mean, the standard deviation, and the mean absolute deviation of vector $\mathbf{a}$, respectively.

3. The intensity features are computed with histograms of 127 bins between 0 and 255 for $\mathbf{I}_v^t(\mathcal{P}_v)$, $\mathbf{I}_v^{t-1}(\mathcal{P}_v)$, $\mathbf{B}_v(\mathcal{P}_v)$ and $|\mathbf{I}_v^t(\mathcal{P}_v) - \mathbf{B}_v(\mathcal{P}_v)|$. Let $f_{IN} : \mathbf{a} \rightarrow \mathbb{R}^{1\times 127}$ be the function representing the histogram extraction process for the intensity vectors, then the whole representation of the intensity $\mathbf{in}_v \in \mathbb{R}^{1\times 508}$ of a node v is obtained as follows $[f_{IN}(\mathbf{I}_v^t(\mathcal{P}_v)), f_{IN}(\mathbf{I}_v^{t-1}(\mathcal{P}_v)), f_{IN}(\mathbf{B}_v(\mathcal{P}_v)), f_{IN}(|\mathbf{I}_v^t(\mathcal{P}_v) - \mathbf{B}_v(\mathcal{P}_v)|)]$.
4. The deep features are computed using the 5th layer (Conv-5) of the VGG model [113]. The image $\mathbf{I}_v^t$ is resized to be compatible with the VGG network, and then a principal component analysis is applied to compress 512 dimensional features into 64 features from Conv-5. Let $f_{DL} : \mathbf{A} \rightarrow \mathbb{R}^{1\times 10816}$ be the function representing the deep feature extraction, then the deep features $\mathbf{dl}_v \in \mathbb{R}^{1\times 10816}$ of a node v is obtained as $\mathbf{dl}_v = f_{DL}(\mathbf{I}_v^t)$.
5. Finally, the feature vector $\mathbf{x}_v \in \mathbb{R}^{1\times 11669}$ is obtained as follows $[\mathbf{lbp}_v, \mathbf{mov}_v, \mathbf{in}_v, \mathbf{dl}_v]$.

Appendix C

Closed-form Solution Variational Problem

Proof. Let $\mathbf{z}_q$ be the graph signal associated with the q-th class. Formally, the variational problem can be stated as:

$$\arg\min_{\mathbf{z}_q} \mathbf{z}_q^{\mathsf{T}}(\mathbf{L}+\epsilon\mathbf{I})^{\alpha}\mathbf{z}_q \quad \text{s.t.} \quad \mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S}) = 0, \tag{C.1}$$

where $\mathbf{z}_q^{\mathsf{T}}(\mathbf{L}+\epsilon\mathbf{I})^{\alpha}\mathbf{z}_q$ is a quadratic convex function and $\mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S})$ is affine in $\mathbf{z}_q$, *i.e.*, Eqn. (C.1) is a convex problem. Equation (C.1) can be expressed as an unconstrained optimization problem using *Lagrange multipliers* such that:

$$\arg\min_{\mathbf{z}_q} \mathbf{z}_q^{\mathsf{T}}(\mathbf{L}+\epsilon\mathbf{I})^{\alpha}\mathbf{z}_q + \boldsymbol{\mu}^{\mathsf{T}}(\mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S})), \tag{C.2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{S}|}$ is the vector of Lagrange multipliers, and $\mathcal{L}(\mathbf{z}_q, \boldsymbol{\mu}) = \mathbf{z}_q^{\mathsf{T}}(\mathbf{L}+\epsilon\mathbf{I})^{\alpha}\mathbf{z}_q + \boldsymbol{\mu}^{\mathsf{T}}(\mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S}))$ is the Lagrange function. For an unconstrained problem, the well known necessary and sufficient condition for $\mathbf{z}_q$ to be optimal [119]:

$$\nabla\mathcal{L}(\mathbf{z}_q, \boldsymbol{\mu}) = 0. \tag{C.3}$$

Now, using Eqn. (C.3):

$$\nabla_{\mathbf{z}_q}\mathcal{L}(\mathbf{z}_q, \boldsymbol{\mu}) = 2(\mathbf{L} + \epsilon\mathbf{I})^{\alpha}\mathbf{z}_q + \mathbf{M}^{\mathsf{T}}\boldsymbol{\mu} = \mathbf{0}$$
$$\mathbf{z}_q = -\frac{1}{2}((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}}\boldsymbol{\mu}, \tag{C.4}$$

and also:

$$\nabla_{\boldsymbol{\mu}}\mathcal{L}(\mathbf{z}_q, \boldsymbol{\mu}) = \mathbf{M}\mathbf{z}_q - \mathbf{y}_q(\mathcal{S}) = \mathbf{0}$$
$$\mathbf{M}\mathbf{z}_q = \mathbf{y}_q(\mathcal{S}). \tag{C.5}$$

then using Eqn. (C.4) and (C.5):

$$\mathbf{z}_q = ((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}}(\mathbf{M}((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}})^{-1}\mathbf{y}_q(\mathcal{S}). \tag{C.6}$$

Equation (C.6) is the closed-form solution for the reconstruction of the graph signal associated with the q-th class. The reconstruction of the whole graph signal $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_Q]$ is achieved by concatenating the samples such that:

$$\mathbf{Z} = ((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}}(\mathbf{M}((\mathbf{L} + \epsilon\mathbf{I})^{-1})^{\alpha}\mathbf{M}^{\mathsf{T}})^{-1}\mathbf{Y}(\mathcal{S}), \tag{C.7}$$

where $\mathbf{Y}(\mathcal{S}) = [\mathbf{y}_1(\mathcal{S}), \mathbf{y}_2(\mathcal{S}), \ldots, \mathbf{y}_Q(\mathcal{S})]$. □

Appendix D

Proof of Theorem 6.4.1

Proof. 1) For the triangle inequality we have that:

$$\begin{aligned} \|\mathbf{x}+\mathbf{y}\|^2_{(\rho),\epsilon} &= \langle \mathbf{x}+\mathbf{y}, \mathbf{y}+\mathbf{x}\rangle_{(\rho),\epsilon} = \|(\mathbf{L}+\epsilon\mathbf{I})^{(\rho/2)}(\mathbf{x}+\mathbf{y})\|^2 \\ &\rightarrow \|\mathbf{x}+\mathbf{y}\|^2_{(\rho),\epsilon} = \|\mathbf{x}\|^2_{(\rho),\epsilon} + \|\mathbf{y}\|^2_{(\rho),\epsilon} + 2\langle \mathbf{x}, \mathbf{y}\rangle_{(\rho),\epsilon}, \end{aligned} \tag{D.1}$$

using the Cauchy–Schwarz inequality we have that:

$$\begin{aligned} \|\mathbf{x}+\mathbf{y}\|^2_{(\rho),\epsilon} &\leq \|\mathbf{x}\|^2_{(\rho),\epsilon} + 2\|\mathbf{x}\|_{(\rho),\epsilon}\|\mathbf{y}\|_{(\rho),\epsilon} + \|\mathbf{y}\|^2_{(\rho),\epsilon} \\ &\rightarrow \|\mathbf{x}+\mathbf{y}\|^2_{(\rho),\epsilon} \leq (\|\mathbf{x}\|_{(\rho),\epsilon} + \|\mathbf{y}\|_{(\rho),\epsilon})^2. \end{aligned} \tag{D.2}$$

2) Let s be a scalar, then for the absolute homogeneity property we have:

$$\|s\mathbf{x}\|_{(\rho),\epsilon} = \|(\mathbf{L}+\epsilon\mathbf{I})^{(\rho/2)}s\mathbf{x}\| = |s|\|(\mathbf{L}+\epsilon\mathbf{I})^{(\rho/2)}\mathbf{x}\| = |s|\|\mathbf{x}\|_{(\rho),\epsilon}. \tag{D.3}$$

3) For the positive definiteness property, we need the Schur product theorem:

Theorem D.0.1. *(Schur product theorem) Let $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{n\times n}$ be two positive definite matrices of dimension $n \times n$, then $\mathbf{S} \circ \mathbf{T}$ is also positive definite.*
Proof: see [160].

For $\epsilon = 0$ and $\rho = 1$ the sparse Sobolev norm becomes the graph Laplacian quadratic form [29], so that $\|\mathbf{x}\|^2_{(1),0} = \mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x} = 0 \iff \mathbf{x} = \tau\mathbf{1}$ (with τ some

constant value), and then $\|\mathbf{x}\|_{(\rho),\epsilon}$ becomes a semi-norm because it satisfies only the first two conditions of vector norms, *i.e.*, triangle inequality and absolute homogeneity. However, for $\epsilon > 0$, the term $\mathbf{L}+\epsilon\mathbf{I} \succ 0$ according to Eq. (3.6), and then $(\mathbf{L}+\epsilon\mathbf{I})^{(\rho)} \succ 0$ using the Schur product theorem. As a result:

$$\|\mathbf{x}\|^2_{(\rho),\epsilon} = \mathbf{x}^\mathsf{T}(\mathbf{L} + \epsilon\mathbf{I})^{(\rho)}\mathbf{x} = 0 \iff \mathbf{x} = 0 \;\forall\; \epsilon > 0. \tag{D.4}$$

As a consequence, the sparse Sobolev norm satisfies the properties of vector norms for all $\epsilon > 0$ according to Eq. (D.2), (D.3), and (D.4). Similarly, the sparse Sobolev definition satisfies the properties of semi-norms for $\epsilon = 0$ according to Eq. (D.2) and (D.3). □

Appendix E

Proof of Lemma 6.4.2

Proof. We can compute the spectral decomposition of the Kronecker product between matrices using concepts of product graphs [234]. The spectrum of the Kronecker product of several graphs is given by the Kronecker products of its spectral components. For example, the spectral decomposition of $\mathbf{L} \otimes \mathbf{L}$ is given by [234]:

$$\mathbf{L} \otimes \mathbf{L} = (\mathbf{U} \otimes \mathbf{U})(\mathbf{\Lambda} \otimes \mathbf{\Lambda})(\mathbf{U}^\mathsf{T} \otimes \mathbf{U}^\mathsf{T}). \tag{E.1}$$

Similarly, there is a relationship between the Kronecker and the Hadamard products as follows:

Theorem E.0.1. *(Visick [235]) For $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{n \times m}$, we have:*

$$\mathbf{S} \circ \mathbf{T} = \mathbf{P}_n^\mathsf{T}(\mathbf{S} \otimes \mathbf{T})\mathbf{P}_m, \tag{E.2}$$

where $\mathbf{P}_n \in \{0,1\}^{n^2 \times n}$ and $\mathbf{P}_m \in \{0,1\}^{m^2 \times m}$ are partial permutation matrices. If $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{n \times n}$ are square matrices, we have that $\mathbf{S} \circ \mathbf{T} = \mathbf{P}_n^\mathsf{T}(\mathbf{S} \otimes \mathbf{T})\mathbf{P}_n$.
Proof: see [235].

We can get a general form of the spectrum of the Hadamard product (for the Hadamard power of order 2) using Eq. (E.1) and Theorem E.0.1 as follows:

$$\mathbf{L} \circ \mathbf{L} = \mathbf{L}^{(2)} = \mathbf{P}_N^\mathsf{T}(\mathbf{U} \otimes \mathbf{U})(\mathbf{\Lambda} \otimes \mathbf{\Lambda})(\mathbf{U}^\mathsf{T} \otimes \mathbf{U}^\mathsf{T})\mathbf{P}_N. \tag{E.3}$$

□

Appendix F

Codebase

We have created a codebase written in PyTorch and PyG [187] for benchmarking methods to alleviate over-smoothing and over-squashing. The code contains: 1) a GNN model, 2) functions to compute the JLC and BFC metrics, 3) a data loader (a modified version of the code released in [202]) with 12 datasets, 4) a random hyperparameter search tool, and 5) a main file with the training procedure. The GNN is fully parameterized and we can do experiments combining several kinds of methods. For example, we can easily run a random hyperparameter search using PairNorm (PN) and Residual Dense Connections (RDC) using the following command:

```
$ python random_search_hyperparameters.py --PairNorm
  --ResidualDenseConnection --dataset Cora --gpu_number 0
```

This command will execute a hyperparameter random search in the GPU with identifier 0 with: 1) the PN and RDC methods combined together, and 2) the Cora dataset.

The full set of arguments that can be modified in a random hyperparameter search are:

- no_cuda: Flag to run the experiment in the CPU.
- gpu_number: Integer specifying the identifier of the GPU where the experiment is being executed.
- ResidualDenseConnection: Flag to activate the residual dense connections.

- PairNorm: Flag to activate the PairNorm normalization.
- DiffGroupNorm: Flag to activate the differentiable group normalization.
- DropEdge: Flag to activate the DropEdge method.
- JostLiuCurvature: Flag to activate our SJLR algorithm.
- FALayer: Flag to activate the fully adjacent layer.
- GraphDifussion: Flag to activate the graph diffusion convolution with personalized PageRank kernel.
- RicciCurvature: Flag to activate the SJLR method.
- GNN: String to determine the type of GNN (only 'GCN' is included in the first version of the code).
- dataset: String to determine the dataset for the experiment with the options 'Cora', 'Citeseer', 'Pubmed', 'Cornell', 'Texas', 'Wisconsin', 'chameleon', 'squirrel', 'Actor', 'Computers', 'Photo', and 'CoauthorCS'.

The main file has a broad set of arguments for hyperparameter search or for executing methods with specific hyperparameters—further details can be found in the main Python file in the repository. For example, we can reproduce the results of the best hyperparameters found in [202] for GDC with personalized PageRank kernel using the following command:

```
$ python main.py --verbose --lr 0.01 --weight_decay 0.09
  --hidden_units 64 --n_layers_set 2 --dropout 0.5
  --GraphDifussion --alphaGDC 0.05 --k 128 --dataset Cora
  --GNN GCN
```

Similarly, we can reproduce the results of SJLR in Table 7.1 using the Cornell dataset with the following command:

```
$ python main.py --verbose --lr 0.0122 --weight_decay 4e-4
  --hidden_units 32 --n_layers_set 2 --dropout 0.3109
  --JostLiuCurvature --pA 0.0495 --pD 0.8918 --tau 429.21
  --alpha 0.7742 --dataset Cornell --GNN GCN
```

This codebase is an open source project and is being released under the MIT license.

Appendix G

Proof of Lemma 7.4.3

Proof. Let $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ be the random walk transition matrix, and let $f : \mathcal{V} \to \mathbb{R}$ be any initial distribution with vector representation $\mathbf{f} \in \mathbb{R}^{N\times 1}$. If we want to measure the distance between $\mathbf{f}^\mathsf{T}\mathbf{P}^s$ and the stationary distribution in the ℓ_2 norm we need to compute:

$$\|\mathbf{f}^\mathsf{T}\mathbf{P}^s - \boldsymbol{\pi}\|_2 = \|\mathbf{f}^\mathsf{T}\mathbf{P}^s - \frac{\mathbf{1}^\mathsf{T}\mathbf{D}}{\mathrm{vol}(G)}\|_2. \tag{G.1}$$

Let $\boldsymbol{\phi}_i$ be the orthonormal eigenfunction associated with λ_i. We know that $\boldsymbol{\phi}_1 = \frac{\mathbf{1}^\mathsf{T}\mathbf{D}^{\frac{1}{2}}}{\sqrt{\mathrm{vol}(G)}}$.Therefore, we have:

$$a_1\boldsymbol{\phi}_1\mathbf{D}^{\frac{1}{2}} = \frac{1}{\sqrt{\mathrm{vol}(G)}}\frac{\mathbf{1}^\mathsf{T}\mathbf{D}^{\frac{1}{2}}}{\sqrt{\mathrm{vol}(G)}}\mathbf{D}^{\frac{1}{2}} = \frac{\mathbf{1}^\mathsf{T}\mathbf{D}}{\mathrm{vol}(G)}, \text{ where } a_1 = \frac{\langle \mathbf{f}^\mathsf{T}\mathbf{D}^{-\frac{1}{2}}, \mathbf{1}^\mathsf{T}\mathbf{D}^{\frac{1}{2}}\rangle}{\|\mathbf{1}^\mathsf{T}\mathbf{D}^{\frac{1}{2}}\|_2} = \frac{1}{\sqrt{\mathrm{vol}(G)}}. \tag{G.2}$$

As a consequence we have that:

$$\|\mathbf{f}^\mathsf{T}\mathbf{P}^s - \frac{\mathbf{1}^\mathsf{T}\mathbf{D}}{\mathrm{vol}(G)}\|_2 = \|\mathbf{f}^\mathsf{T}\mathbf{P}^s - a_1\boldsymbol{\phi}_1\mathbf{D}^{\frac{1}{2}}\|_2 = \|\mathbf{f}^\mathsf{T}\mathbf{D}^{-\frac{1}{2}}(\mathbf{I} - \boldsymbol{\Delta})^s\mathbf{D}^{\frac{1}{2}} - a_1\boldsymbol{\phi}_1\mathbf{D}^{\frac{1}{2}}\|_2, \tag{G.3}$$

since $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{I} - \boldsymbol{\Delta})\mathbf{D}^{\frac{1}{2}}$. Suppose we write $\mathbf{f}^\mathsf{T}\mathbf{D}^{-\frac{1}{2}} = \sum_i a_i \boldsymbol{\phi}_i$, then:

$$\begin{aligned}
\|\mathbf{f}^\mathsf{T}\mathbf{P}^s - \boldsymbol{\pi}\|_2 &= \|\sum_i a_i \boldsymbol{\phi}_i (\mathbf{I} - \boldsymbol{\Delta})^s \mathbf{D}^{\frac{1}{2}} - a_1 \boldsymbol{\phi}_1 (1-\lambda_1)^s \mathbf{D}^{\frac{1}{2}}\|_2 \\
&= \|\sum_{i\neq 1} a_i \boldsymbol{\phi}_i (1-\lambda_i)^s \mathbf{D}^{\frac{1}{2}}\|_2 = \|\sum_{i\neq 1} (1-\lambda_i)^s a_i \boldsymbol{\phi}_i \mathbf{D}^{\frac{1}{2}}\|_2 \\
&\leq (1-\lambda')^s \frac{\max_i \sqrt{d_i}}{\min_j \sqrt{d_j}} \\
&\leq e^{-s\lambda'} \frac{\max_i \sqrt{d_i}}{\min_j \sqrt{d_j}}, \text{ where } \lambda' = \begin{cases} \lambda_2 & \text{if } 1-\lambda_2 \geq \lambda_N - 1, \\ 2-\lambda_N & \text{otherwise.} \end{cases}
\end{aligned} \tag{G.4}$$

From Section 7.4.1 we know that only λ_2 is important in (G.4) since we can apply lazy walk. As a result, we can compute the value of s such that $\|\mathbf{f}^\mathsf{T}\mathbf{P}^s - \boldsymbol{\pi}\|_2 \leq \epsilon$ as follows:

$$s \geq \frac{1}{\lambda_2} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right) \rightarrow \lambda_2 \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{G.5}$$

Finally, using the Cheeger inequality in (7.2) we have that:

$$2h_G \geq \lambda_2 \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right) \rightarrow 2h_G \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{G.6}$$

□

Appendix H

Hyperparameters Search

We define a range of values or a set of number for the hyperparameter search that we called the search space: 1) learning rate $lr \in [0.005, 0.02]$, 2) weight decay $wd \in [0.0001, 0.001]$, 3) hidden units of each graph convolutional layer $h_u \in \{16, 32, 64, 128\}$, 4) dropout $d \in [0.3, 0.6]$, and 5) number of layers $L \in \{1, 2, 3\}$. We also define ranges, or sets, for the hyperparameters in each specific method:

1. Stochastic Jost and Liu curvature Rewiring: $p_A \in [0, 1]$, $p_D \in [0, 1]$, $\tau \in [1, 500]$, and $\alpha \in [0, 1]$.
2. DropEdge: Probability of dropping an edge $p_D \in [0, 1]$.
3. PairNorm: Scale $s \in \{0.1, 1, 10, 50, 100\}$.
4. Differentiable Group Normalization: number of clusters $c \in \{3, 4, \ldots, 10\}$, and balancing factor $bf \in [0.0005, 0.05]$.
5. Graph Diffusion Convolution with personalized PageRank kernel: $\alpha_{\text{GDC}} \in [0.01, 0.2]$, and $k \in \{16, 32, 64, 128\}$.
6. Stochastic Discrete Ricci Flow: Stochasticity level $\tau \in [1, 500]$, iterations $it \in [20, 4000]$, and Ricci curvature upper-bound $C^+ \in [0.1, 40]$.

Tables H.1-H.9 show the best hyperparameters computed from the validation sets in the datasets tested in Section 7.6.

Table H.1: Best hyperparameters for Cornell dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.0199	$9e-4$	128	0.5762	2	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0190	$8e-4$	128	0.5529	2	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0101	$1e-3$	128	0.3594	1	-	-	-	-	-	-	-	0.1956	128	-	-
DE [185]	0.0197	$7e-4$	32	0.3574	3	-	0.9722	-	-	-	-	-	-	-	-	-
PN [193]	0.0169	$6e-4$	128	0.5240	2	-	-	-	-	10	-	-	-	-	-	-
DGN [194]	0.0116	$7e-4$	128	0.5270	2	-	-	-	-	-	10	0.0293	-	-	-	-
FA [47]	0.0056	$2e-4$	32	0.4462	3	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0128	$8e-4$	64	0.4990	2	-	-	23.49	-	-	-	-	-	-	555	12.01
SJLR	0.0187	$6e-4$	64	0.5844	3	0.0973	0.9976	79.32	0.7035	-	-	-	-	-	-	-

Table H.2: Best hyperparameters for Texas dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.018	$8e-4$	128	0.3425	2	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0135	$7e-4$	128	0.3059	3	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0182	$7e-4$	16	0.4499	3	-	-	-	-	-	-	-	0.1357	128	-	-
DE [185]	0.0198	$7e-4$	64	0.5083	3	-	0.9934	-	-	-	-	-	-	-	-	-
PN [193]	0.0192	$2e-4$	64	0.3593	2	-	-	-	-	10	-	-	-	-	-	-
DGN [194]	0.01	$5e-4$	128	0.3226	2	-	-	-	-	-	5	0.0488	-	-	-	-
FA [47]	0.0167	$7e-4$	32	0.4446	3	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0075	$4e-4$	128	0.4235	2	-	-	20.73	-	-	-	-	-	-	3672	2.72
SJLR	0.0199	$1e-3$	128	0.4946	2	0.015	0.9296	90.58	0.3178	-	-	-	-	-	-	-

Table H.3: Best hyperparameters for Wisconsin dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.0188	$1e-3$	128	0.3538	2	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.019	$8e-4$	64	0.494	3	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0075	$7e-4$	64	0.4594	1	-	-	-	-	-	-	-	0.1946	128	-	-
DE [185]	0.0135	$8e-4$	32	0.5057	3	-	0.9996	-	-	-	-	-	-	-	-	-
PN [193]	0.018	$2e-4$	128	0.3635	2	-	-	-	-	100	-	-	-	-	-	-
DGN [194]	0.0184	$8e-4$	128	0.3389	2	-	-	-	-	-	5	0.0206	-	-	-	-
FA [47]	0.0123	$9e-4$	32	0.4565	2	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0082	$9e-4$	32	0.3961	2	-	-	69.99	-	-	-	-	-	-	3919	22.99
SJLR	0.0104	$1e-4$	32	0.5513	2	0.0644	0.9707	77.54	0.6372	-	-	-	-	-	-	-

Table H.4: Best hyperparameters for Chameleon dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.02	$8e-4$	16	0.4969	3	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0174	$9e-4$	64	0.5468	2	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0113	$1e-3$	16	0.3976	3	-	-	-	-	-	-	-	0.0201	128	-	-
DE [185]	0.0166	$1e-3$	16	0.4696	3	-	0.0678	-	-	-	-	-	-	-	-	-
PN [193]	0.0189	$1e-3$	32	0.4600	3	-	-	-	-	100	-	-	-	-	-	-
DGN [194]	0.0171	$8e-4$	16	0.5591	3	-	-	-	-	-	10	0.0475	-	-	-	-
FA [47]	0.0198	$4e-4$	128	0.3518	2	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0106	$9e-4$	16	0.5253	3	-	-	473.78	-	-	-	-	-	-	479	12.74
SJLR	0.0192	$9e-4$	128	0.5164	3	0.1388	0.1754	216.25	0.2388	-	-	-	-	-	-	-

Table H.5: Best hyperparameters for Squirrel dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.014	$1e-3$	16	0.5016	3	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0177	$1e-3$	128	0.4301	3	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0195	$8e-4$	128	0.5179	1	-	-	-	-	-	-	-	0.0434	128	-	-
DE [185]	0.0153	$8e-4$	64	0.3963	2	-	0.0415	-	-	-	-	-	-	-	-	-
PN [193]	0.0193	$7e-4$	16	0.4254	3	-	-	-	-	100	-	-	-	-	-	-
DGN [194]	0.0185	$9e-4$	128	0.3887	3	-	-	-	-	-	6	0.0079	-	-	-	-
FA [47]	0.0192	$5e-4$	32	0.4287	1	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.02	$9e-4$	32	0.5707	3	-	-	446.06	-	-	-	-	-	-	2136	30.11
SJLR	0.0179	$9e-4$	128	0.5786	3	0.1183	0.1746	348.67	0.0147	-	-	-	-	-	-	-

Table H.6: Best hyperparameters for Actor dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.0177	0.001	32	0.5611	2	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0155	$1e-3$	32	0.3782	2	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0051	$2e-4$	32	0.4514	1	-	-	-	-	-	-	-	0.1988	64	-	-
DE [185]	0.0199	$8e-4$	32	0.3998	1	-	0.9783	-	-	-	-	-	-	-	-	-
PN [193]	0.0196	$6e-4$	128	0.5338	2	-	-	-	-	0.1	-	-	-	-	-	-
DGN [194]	0.0188	$1e-3$	64	0.488	2	-	-	-	-	-	9	0.0073	-	-	-	-
FA* [47]	0.0078	0.0134	128	0.7800	2	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0171	$8e-4$	128	0.5228	2	-	-	456.02	-	-	-	-	-	-	444	21.78
SJLR	0.0147	$5e-4$	32	0.4151	1	0.1049	0.8413	281.1313	0.7295	-	-	-	-	-	-	-

* Certain hyperparameters were copied directly from [190].

Table H.7: Best hyperparameters for Cora dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.0133	$9e-4$	128	0.5196	3	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0128	$8e-4$	128	0.5771	3	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0189	$7e-4$	128	0.5905	2	-	-	-	-	-	-	-	0.1058	64	-	-
DE [185]	0.0094	$1e-3$	64	0.3592	3	-	0.0286	-	-	-	-	-	-	-	-	-
PN [193]	0.0195	$3e-4$	64	0.4613	3	-	-	-	-	100	-	-	-	-	-	-
DGN [194]	0.0162	$1e-3$	128	0.4333	3	-	-	-	-	-	3	0.0174	-	-	-	-
FA [47]	0.0120	$4e-4$	128	0.3383	1	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0167	$9e-4$	32	0.5225	2	-	-	239.01	-	-	-	-	-	-	79	7.37
SJLR	0.0148	$8e-4$	128	0.5115	2	0.0276	0.0128	324.40	0.0722	-	-	-	-	-	-	-

Table H.8: Best hyperparameters for Citeseer dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.014	$8e-4$	128	0.4543	2	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0197	$5e-4$	128	0.5692	2	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.0144	$1e-3$	128	0.4414	2	-	-	-	-	-	-	-	0.1695	64	-	-
DE [185]	0.016	$6e-4$	128	0.4341	2	-	0.0814	-	-	-	-	-	-	-	-	-
PN [193]	0.0185	$8e-4$	128	0.301	2	-	-	-	-	50	-	-	-	-	-	-
DGN [194]	0.0189	$9e-4$	128	0.3688	2	-	-	-	-	-	3	0.0499	-	-	-	-
FA [47]	0.0142	$5e-4$	16	0.5567	1	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0137	$6e-4$	128	0.4214	2	-	-	371.20	-	-	-	-	-	-	1212	30.18
SJLR	0.0163	$5e-4$	64	0.3216	2	0.2273	0.0362	35.30	0.2998	-	-	-	-	-	-	-

Table H.9: Best hyperparameters for Pubmed dataset.

Method	lr	wd	h_u	d	L	p_A	p_D	τ	α	s	c	bf	α_{GDC}	k	it	C^+
Baseline	0.0194	$5e-4$	64	0.5678	3	-	-	-	-	-	-	-	-	-	-	-
RDC [38]	0.0198	$7e-4$	128	0.4821	3	-	-	-	-	-	-	-	-	-	-	-
GDC [202]	0.013	$4e-4$	128	0.5693	2	-	-	-	-	-	-	-	0.1915	128	-	-
DE [185]	0.0121	$8e-4$	128	0.4476	3	-	0.0358	-	-	-	-	-	-	-	-	-
PN [193]	0.0116	$5e-4$	128	0.5268	3	-	-	-	-	10	-	-	-	-	-	-
DGN [194]	0.0148	$5e-4$	128	0.4678	3	-	-	-	-	-	7	0.0018	-	-	-	-
FA* [47]	0.0204	0.0215	128	0.3376	2	-	-	-	-	-	-	-	-	-	-	-
SDRF [190]	0.0181	$5e-4$	128	0.5972	3	-	-	102.56	-	-	-	-	-	-	3233	34.73
SJLR	0.0164	$1e-3$	16	0.4486	3	0.0616	0.1734	319.60	0.5751	-	-	-	-	-	-	-

* Certain hyperparameters were copied directly from [190].

Appendix I

Proof of Theorem 8.4.1 (Conditioning Number)

Proof. The problem associated with the Laplacian matrix in (8.5) can be rewritten as follows [26]:

$$f_L(\mathbf{z}) = \min_{\mathbf{z}} \frac{1}{2}\|\mathbf{Q}[\mathbf{z} - \text{vec}\,(\mathbf{Y})]\|_2^2 + \frac{\upsilon}{2}\mathbf{z}^\mathsf{T}[(\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes \mathbf{L}]\mathbf{z}, \tag{I.1}$$

where $\mathbf{z} = \text{vec}\,(\tilde{\mathbf{X}})$. Moreover, the gradient of $f_L(\mathbf{z})$ in (I.1) is such that:

$$\nabla_{\mathbf{z}} f_L(\mathbf{z}) = \mathbf{Q}[\mathbf{z} - \text{vec}\,(\mathbf{Y})] + \upsilon[(\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes \mathbf{L}]\mathbf{z}, \tag{I.2}$$

and the Hessian matrix of $f_L(\mathbf{z})$ is given as follows:

$$\nabla_{\mathbf{z}}^2 f_L(\mathbf{z}) = \mathbf{Q} + [\upsilon(\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes \mathbf{L}]. \tag{I.3}$$

Similarly, the Hessian matrix associated with the Sobolev formulation is given by:

$$\nabla_{\mathbf{z}}^2 f_S(\mathbf{z}) = \mathbf{Q} + [\upsilon(\mathbf{D}_h\mathbf{D}_h^\mathsf{T}) \otimes (\mathbf{L} + \epsilon\mathbf{I})^\beta]. \tag{I.4}$$

$\mathbf{D}_h\mathbf{D}_h^\mathsf{T}$ is a positive semi-definite matrix, so it has a matrix of eigenvalues $\mathbf{\Lambda}_D = \text{diag}(\lambda_{(D)1}, \lambda_{(D)2}, \ldots, \lambda_{(D)N})$, with $0 \leq \lambda_{(D)1} \leq \lambda_{(D)2} \leq \cdots \leq \lambda_{(D)N}$, and the corre-

sponding matrix of eigenvectors $\mathbf{U}_D$. As a consequence, from (I.3), we have:

$$\nabla^2_{\mathbf{z}} f_L(\mathbf{z}) = \mathbf{Q} + [\upsilon(\mathbf{U}_D \mathbf{\Lambda}_D \mathbf{U}_D^\intercal) \otimes (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal)],$$
$$\nabla^2_{\mathbf{z}} f_L(\mathbf{z}) = \upsilon \left[\frac{1}{\upsilon}\mathbf{Q} + (\mathbf{U}_D \otimes \mathbf{U})(\mathbf{\Lambda}_D \otimes \mathbf{\Lambda})(\mathbf{U}_D^\intercal \otimes \mathbf{U}^\intercal) \right], \tag{I.5}$$

where we used the property of Kronecker products $(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = \mathbf{AC}\otimes\mathbf{BD}$ [160]. $\mathbf{Q}$ is positive semi-definite because it is a diagonal matrix with diagonal elements either 0 or 1. As a result, from (I.5), we know that $\lambda_{min}(\frac{1}{\upsilon}\mathbf{Q}) = 0$, $\lambda_{max}(\frac{1}{\upsilon}\mathbf{Q}) = \frac{1}{\upsilon}$ if $\mathbf{J} \neq \mathbf{0}$, $\lambda_{min}((\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes \mathbf{L}) = 0$, and $\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes \mathbf{L}) = \lambda_N\lambda_{(D)N}$ if $\lambda_N, \lambda_{(D)N} \geq 1$. For the Sobolev problem, we have that:

$$(\mathbf{L} + \epsilon\mathbf{I})^\beta = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal + \epsilon\mathbf{I})^\beta = \mathbf{U}(\mathbf{\Lambda} + \epsilon\mathbf{I})^\beta\mathbf{U}^\intercal, \tag{I.6}$$

and then:

$$\nabla^2_{\mathbf{z}} f_S(\mathbf{z}) = \upsilon \left[\frac{1}{\upsilon}\mathbf{Q} + (\mathbf{U}_D \otimes \mathbf{U})(\mathbf{\Lambda}_D \otimes (\mathbf{\Lambda} + \epsilon\mathbf{I})^\beta)(\mathbf{U}_D^\intercal \otimes \mathbf{U}^\intercal) \right], \tag{I.7}$$

where $\lambda_{min}((\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes (\mathbf{L}+\epsilon\mathbf{I})^\beta) = 0$ and $\lambda_{max}((\mathbf{D}_h\mathbf{D}_h^\intercal) \otimes (\mathbf{L}+\epsilon\mathbf{I})^\beta) = (\lambda_N+\epsilon)^\beta\lambda_{(D)N}$ if $(\lambda_N + \epsilon)^\beta, \lambda_{(D)N} \geq 1$. We have a summation of two Hermitian matrices in (I.5) and (I.7) (we can trivially prove that (I.5) and (I.7) are Hermitian matrices since $(\mathbf{A} \otimes \mathbf{B})^\intercal = \mathbf{A}^\intercal \otimes \mathbf{B}^\intercal$). Therefore, we cannot get an exact value for the eigenvalues of these Hessian functions. The problem of describing the possible eigenvalues of the sum of two Hermitian matrices in terms of their spectra has attracted the interest of mathematicians for decades. The most complete description was conjectured by Horn *et al.* [236]. The interested readers are referred to [237] for an interesting discussion in the topic. However, we can get some bounds using the developments of Weyl [160]:

Theorem I.0.1 (Weyl's Theorem (4.3.1 in [160]))**.** *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be Hermitian matrices with set of eigenvalues* $\{a_1, a_2, \ldots, a_N\}$ *and* $\{b_1, b_2, \ldots, b_N\}$*, respectively, where the eigenvalues are in ascending order. Then,* $c_i \leq a_{i+j} + b_{N-j}$*, with* $j = 0, 1, \ldots, N - i$*, and* $a_{i-j+1} + b_j \leq c_i$*, with* $j = 1, \ldots, i$*, where* c_i *is the ith eigenvalue of* $\mathbf{A} + \mathbf{B}$*.*

Proof: see [160].

Using Theorem I.0.1 and (I.5), we can have the following inequalities for $\nabla^2_{\mathbf{z}} f_L(\mathbf{z})$:

$$\lambda_N \lambda_{(D)N} \leq \lambda_{max}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \leq \lambda_N \lambda_{(D)N} + \frac{1}{\upsilon}, \tag{I.8}$$

$$0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \leq \frac{1}{\upsilon}, \tag{I.9}$$

$$0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \leq \lambda_N \lambda_{(D)N}. \tag{I.10}$$

Similarly, for $\nabla^2_{\mathbf{z}} f_S(\mathbf{z})$ we have:

$$\begin{aligned}(\lambda_N + \epsilon)^\beta \lambda_{(D)N} &\leq \lambda_{max}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \\ &\leq (\lambda_N + \epsilon)^\beta \lambda_{(D)N} + \frac{1}{\upsilon}, \end{aligned} \tag{I.11}$$

$$0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \leq \frac{1}{\upsilon}, \tag{I.12}$$

$$0 \leq \lambda_{min}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \leq (\lambda_N + \epsilon)^\beta \lambda_{(D)N}, \tag{I.13}$$

Notice that we ignored the υ value that multiplies (I.5) and (I.7) in these inequalities since $\kappa(\upsilon \mathbf{A}) = \kappa(\mathbf{A})$. We can have some asymptotic analysis based on the inequalities (I.8)-(I.13):

1. For $\epsilon > 0$ and $\beta > 1$ we have that the upper bound in (I.13) is looser than the upper bound in (I.10) given that $\lambda_N + \epsilon \geq 1$, which favors better condition numbers for the Sobolev problem. For example, if we assume that the eigenvalues are equal to the upper bounds in (I.8), (I.10), (I.11), and (I.13) we have that:

$$\begin{aligned}\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) &= \left(1 + \frac{1}{\upsilon(\lambda_N + \epsilon)^\beta \lambda_{(D)N}}\right)^2 \\ &\leq \kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) = \left(1 + \frac{1}{\upsilon \lambda_N \lambda_{(D)N}}\right)^2. \end{aligned} \tag{I.14}$$

2. When $\upsilon \to \infty$, $\lambda_{min}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \to 0$ and $\lambda_{min}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to 0$ according to (I.9) and (I.12), and then $\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) \to \infty$ and $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$.
3. When $\epsilon \to \infty$ and $\beta > 0$, $\lambda_{max}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$ according to (I.11) and then $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$. When $\epsilon \to \infty$ and $\lambda_{min}(\nabla^2_{\mathbf{z}} f_L(\mathbf{z})) > 0$ we have that $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) >$

$\kappa(\nabla^2_{\mathbf{z}} f_L(\mathbf{z}))$.

4. when $\beta \to \infty$ and $\lambda_N + \epsilon > 1$, $\lambda_{max}(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$ according to (I.11) and then $\kappa(\nabla^2_{\mathbf{z}} f_S(\mathbf{z})) \to \infty$.

□

Bibliography

[1] J. H. Giraldo, S. Javed, and T. Bouwmans, "Graph moving object segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 2485–2503, 2022.

[2] J. H. Giraldo and T. Bouwmans, "Semi-supervised background subtrction of unseen videos: Minimization of the total variation of graph signals," in *IEEE International Conference on Image Processing*, 2020.

[3] J. H. Giraldo and T. Bouwmans, "GraphBGS: Background subtraction via recovery of graph signals," in *International Conference on Pattern Recognition*, 2021.

[4] J. H. Giraldo, S. Javed, M. Sultana, S. K. Jung, and T. Bouwmans, "The emerging field of graph signal processing for moving object segmentation," in *International Workshop on Frontiers of Computer Vision*, 2021.

[5] J. H. Giraldo, S. Javed, N. Werghi, and T. Bouwmans, "Graph CNN for moving object detection in complex environments from unseen videos," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2021.

[6] J. H. Giraldo, V. Scarrica, A. Staiano, F. Camastra, and T. Bouwmans, "Hypergraph convolutional networks for weakly-supervised semantic segmentation," in *IEEE International Conference on Image Processing*, 2022.

[7] J. H. Giraldo, S. Javed, A. Mahmood, F. D. Malliaros, and T. Bouwmans, "Sparse Sobolev graph neural networks," *Preprint*, 2022.

[8] J. H. Giraldo and T. Bouwmans, "On the minimization of Sobolev norms of time-varying graph signals: Estimation of new Coronavirus disease 2019 cases," in *IEEE International Workshop on Machine Learning for Signal Processing*, 2020.

[9] J. H. Giraldo, A. Mahmood, B. Garcia-Garcia, D. Thanou, and T. Bouwmans, "Reconstruction of time-varying graph signals via Sobolev smoothness," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 8, pp. 201–214, 2022.

[10] Y. Wang, P. M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, "CDnet 2014: An expanded change detection benchmark dataset," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2014.

[11] V. Mahadevan and N. Vasconcelos, "Spatiotemporal saliency in dynamic scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 171–177, 2009.

[12] P. L. St-Charles, G. A. Bilodeau, and R. Bergevin, "A self-adjusting approach to change detection based on background word consensus," in *IEEE/CVF Winter Conference on Applications of Computer Vision*, 2015.

[13] S. Bianco, G. Ciocca, and R. Schettini, "Combination of video change detection algorithms by genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 914–928, 2017.

[14] O. Tezcan, P. Ishwar, and J. Konrad, "BSUV-Net: A fully-convolutional neural network for background subtraction of unseen videos," in *IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020.

[15] X. Shu, F. Porikli, and N. Ahuja, "Robust orthonormal subspace learning: Efficient recovery of corrupted low-rank matrices," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2014.

[16] X. Zhou, C. Yang, and W. Yu, "Moving object detection by detecting contiguous outliers in the low-rank representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 3, pp. 597–610, 2012.

[17] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[18] F. Yang, Q. Sun, H. Jin, and Z. Zhou, "Superpixel segmentation with fully convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[19] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[21] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," *The Lancet Infectious Diseases*, vol. 20, no. 5, pp. 533–534, 2020.

[22] R. Sibson, "A brief description of natural neighbor interpolation," *Interpreting Multivariate Data*, 1981.

[23] S. K. Narang, A. Gadde, E. Sanou, and A. Ortega, "Localized iterative methods for interpolation in graph structured data," in *IEEE Global Conference on Signal and Information Processing*, 2013.

[24] N. Perraudin, A. Loukas, F. Grassi, and P. Vandergheynst, "Towards stationary time-vertex signal processing," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.

[25] A. Loukas and N. Perraudin, "Stationary time-vertex signal processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2019, no. 1, pp. 1–19, 2019.

[26] K. Qiu, X. Mao, X. Shen, X. Wang, T. Li, and Y. Gu, "Time-varying graph signal reconstruction," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 870–883, 2017.

[27] G. Puy, N. Tremblay, R. Gribonval, and P. Vandergheynst, "Random sampling of bandlimited signals on graphs," *Applied and Computational Harmonic Analysis*, vol. 44, no. 2, pp. 446–475, 2018.

[28] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[29] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[30] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations*, 2014.

[31] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016.

[32] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[34] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[35] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics*, vol. 38, no. 5, pp. 1–12, 2019.

[38] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?," in *IEEE/CVF International Conference on Computer Vision*, 2019.

[39] W. Uwents, G. Monfardini, H. Blockeel, M. Gori, and F. Scarselli, "Neural networks for relational learning: An experimental comparison," *Machine Learning*, vol. 82, no. 3, pp. 315–349, 2011.

[40] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[41] P. Gainza, F. Sverrisson, F. Monti, E. Rodola, D. Boscaini, M. M. Bronstein, and B. E. Correia, "Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning," *Nature Methods*, vol. 17, no. 2, pp. 184–192, 2020.

[42] H. E. Egilmez, Y. H. Chao, and A. Ortega, "Graph-based transforms for video coding," *IEEE Transactions on Image Processing*, vol. 29, pp. 9330–9344, 2020.

[43] S. Javed, A. Mahmood, J. Dias, and N. Werghi, "Robust structural low-rank tracking," *IEEE Transactions on Image Processing*, vol. 29, pp. 4390–4405, 2020.

[44] S. Javed, A. Mahmood, N. Werghi, K. Benes, and N. Rajpoot, "Multiplex cellular communities in multi-gigapixel colorectal cancer histology images for tissue phenotyping," *IEEE Transactions on Image Processing*, vol. 29, pp. 9204–9219, 2020.

[45] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," in *Advances in Neural Information Processing Systems*, 2020.

[46] Q. Li, Z. Han, and X. M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI Conference on Artificial Intelligence*, 2018.

[47] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations*, 2021.

[48] I. Pesenson, "Sampling in Paley-Wiener spaces on combinatorial graphs," *Transactions of the American Mathematical Society*, vol. 360, no. 10, pp. 5603–5627, 2008.

[49] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Transactions on Signal Processing*, vol. 63, no. 24, pp. 6510–6523, 2015.

[50] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of Machine Learning Research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[51] I. Pesenson, "Variational splines and Paley-Wiener spaces on combinatorial graphs," *Constructive Approximation*, vol. 29, no. 1, pp. 1–21, 2009.

[52] A. Anis, A. El Gamal, A. S. Avestimehr, and A. Ortega, "A sampling theory perspective of graph-based semi-supervised learning," *IEEE Transactions on Information Theory*, vol. 65, no. 4, pp. 2322–2342, 2018.

[53] A. Anis, A. Gadde, and A. Ortega, "Efficient sampling set selection for bandlimited graph signals using graph spectral proxies," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3775–3789, 2016.

[54] B. Garcia-Garcia, T. Bouwmans, and A. J. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, 2020.

[55] D. S. Lee, "Effective Gaussian mixture learning for video background subtraction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 5, pp. 827–832, 2005.

[56] Y. Sheikh and M. Shah, "Bayesian modeling of dynamic scenes for object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 11, pp. 1778–1792, 2005.

[57] T. S. F. Haines and T. Xiang, "Background subtraction with Dirichlet process mixture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 670–683, 2013.

[58] T. Bouwmans, "Traditional and recent approaches in background modeling for foreground detection: An overview," *Computer Science Review*, vol. 11, pp. 31–66, 2014.

[59] T. Bouwmans, A. Sobral, S. Javed, S. K. Jung, and E. H. Zahzah, "Decomposition into low-rank plus additive matrices for background/foreground separation: A review for a comparative evaluation with a large-scale dataset," *Computer Science Review*, vol. 23, pp. 1–71, 2017.

[60] O. Oreifej, X. Li, and M. Shah, "Simultaneous video stabilization and moving object detection in turbulence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 450–462, 2012.

[61] D. S. Pham, O. Arandjelović, and S. Venkatesh, "Detection of dynamic background due to swaying movements from motion features," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 332–344, 2014.

[62] S. Li, D. Florencio, W. Li, Y. Zhao, and C. Cook, "A fusion framework for camouflaged moving foreground detection in the wavelet domain," *IEEE Transactions on Image Processing*, vol. 27, no. 8, pp. 3918–3930, 2018.

[63] T. Bouwmans and E. H. Zahzah, "Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance," *Computer Vision and Image Understanding*, vol. 122, pp. 22–34, 2014.

[64] T. Bouwmans, S. Javed, M. Sultana, and S. K. Jung, "Deep neural network concepts for background subtraction: A systematic review and comparative evaluation," *Neural Networks*, 2019.

[65] L. Maddalena and A. Petrosino, "Towards benchmarking scene background initialization," in *International Conference on Image Analysis and Processing*, 2015.

[66] T. Bouwmans, F. El Baf, and B. Vachon, "Background modeling using mixture of Gaussians for foreground detection-a survey," *Recent Patents on Computer Science*, vol. 1, no. 3, pp. 219–237, 2008.

[67] L. A. Lim and H. Y. Keles, "Learning multi-scale features for foreground segmentation," *Pattern Analysis and Applications*, Aug 2019.

[68] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[69] C. Li, L. Lin, W. Zuo, J. Tang, and M. H. Yang, "Visual tracking via dynamic graph learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 11, pp. 2770–2782, 2018.

[70] S. Javed, A. Mahmood, T. Bouwmans, and S. K. Jung, "Spatiotemporal low-rank modeling for complex scene background initialization," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1315–1329, 2016.

[71] S. Javed, A. Mahmood, T. Bouwmans, and S. K. Jung, "Background-foreground modeling based on spatiotemporal sparse subspace clustering," *IEEE Transactions on Image Processing*, vol. 26, no. 12, pp. 5840–5854, 2017.

[72] S. Javed, A. Mahmood, S. Al-Maadeed, T. Bouwmans, and S. K. Jung, "Moving object detection in complex scene using spatiotemporal structured-sparse rpca," *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 1007–1022, 2018.

[73] S. Javed, A. Mahmood, M. M. Fraz, N. A. Koohbanani, K. Benes, Y. W. Tsang, K. Hewitt, D. Epstein, D. Snead, and N. Rajpoot, "Cellular community detection for tissue phenotyping in colorectal cancer histology images," *Medical Image Analysis*, p. 101696, 2020.

[74] D. Romero, M. Ma, and G. B. Giannakis, "Kernel-based reconstruction of graph signals," *IEEE Transactions on Signal Processing*, vol. 65, no. 3, pp. 764–778, 2016.

[75] A. Parada-Mayorga, D. L. Lau, J. H. Giraldo, and G. R. Arce, "Blue-noise sampling on graphs," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 3, pp. 554–569, 2019.

[76] L. Li, W. Huang, I. Y. Gu, and Q. Tian, "Foreground object detection from videos containing complex background," in *ACM Multimedia*, 2003.

[77] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[78] F. R. K. Chung, *Spectral graph theory*. No. 92, American Mathematical Society, 1997.

[79] X. Zhu and M. Rabbat, "Approximating signals supported on graphs," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

[80] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[81] S. K. Narang and A. Ortega, "Local two-channel critically sampled filter-banks on graphs," in *IEEE International Conference on Image Processing*, 2010.

[82] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *ACM International Symposium on Information Processing in Sensor Networks*, 2004.

[83] R. Wagner, V. Delouille, and R. Baraniuk, "Distributed wavelet de-noising for sensor networks," in *IEEE Conference on Decision and Control*, 2006.

[84] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 888–905, Aug 2000.

[85] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng, "Graph spectral image processing," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 907–930, 2018.

[86] I. Guskov, W. Sweldens, and P. Schröder, "Multiresolution signal processing for meshes," in *ACM Conference on Computer Graphics and Interactive Techniques SIGGRAPH*, 1999.

[87] K. Zhou, H. Bao, and J. Shi, "3D surface filtering using spherical harmonics," *Computer-Aided Design*, vol. 36, no. 4, pp. 363–375, 2004.

[88] R. Hammack, W. Imrich, and S. Klavžar, *Handbook of product graphs.* CRC press, 2011.

[89] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1999.

[90] F. El Baf, T. Bouwmans, and B. Vachon, "Fuzzy integral for moving object detection," in *IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, 2008.

[91] Y. Dong and G. N. DeSouza, "Adaptive learning of multi-subspace for foreground detection under illumination changes," *Computer Vision and Image Understanding*, vol. 115, no. 1, pp. 31–49, 2011.

[92] F. Shang, J. Cheng, Y. Liu, Z. Q. Luo, and Z. Lin, "Bilinear factor matrix norm minimization for robust PCA: Algorithms and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 9, pp. 2066–2080, 2017.

[93] N. Vaswani, T. Bouwmans, S. Javed, and P. Narayanamurthy, "Robust subspace learning: Robust PCA, robust subspace tracking, and robust subspace recovery," *IEEE Signal Processing Magazine*, vol. 35, no. 4, pp. 32–55, 2018.

[94] J. A. Ramirez-Quintana and M. I. Chacon-Murguia, "Self-adaptive SOM-CNN neural system for dynamic object detection in normal and complex scenarios," *Pattern Recognition*, vol. 48, no. 4, pp. 1137–1149, 2015.

[95] G. T. Cinar and J. C. Príncipe, "Adaptive background estimation using an information theoretic cost for hidden state estimation," in *IEEE International Joint Conference on Neural Networks*, 2011.

[96] M. Braham and M. Van Droogenbroeck, "Deep background subtraction with scene-specific convolutional neural networks," in *IEEE International Conference on Systems, Signals and Image Processing*, 2016.

[97] M. Mandal, V. Dhar, A. Mishra, and S. K. Vipparthi, "3DFR: A swift 3D feature reductionist framework for scene independent change detection," *IEEE Signal Processing Letters*, vol. 26, no. 12, pp. 1882–1886, 2019.

[98] L. A. Lim and H. Y. Keles, "Foreground segmentation using convolutional neural networks for multiscale feature encoding," *Pattern Recognition Letters*, vol. 112, pp. 256–262, 2018.

[99] Y. Wang, Z. Luo, and P. M. Jodoin, "Interactive deep learning method for segmenting moving objects," *Pattern Recognition Letters*, vol. 96, pp. 66–75, 2017.

[100] J. Garcia-Gonzalez, J. M. Ortiz-de Lazcano-Lobato, R. M. Luque-Baena, M. A. Molina-Cabello, and E. López-Rubio, "Foreground detection by probabilistic modeling of the features discovered by stacked denoising autoencoders in noisy video sequences," *Pattern Recognition Letters*, 2019.

[101] M. Sultana, A. Mahmood, S. Javed, and S. K. Jung, "Unsupervised deep context prediction for background estimation and foreground segmentation," *Machine Vision and Applications*, vol. 30, no. 3, pp. 375–395, 2019.

[102] J. I. Jung, J. Jang, and J. Hong, "Cosine focal loss-based change detection for video surveillance systems," in *IEEE Conference on Advanced Video and Signal Based Surveillance*, 2019.

[103] J. E. Vargas-Muñoz, A. S. Chowdhury, E. B. Alexandre, F. L. Galvão, P. A. V. Miranda, and A. X. Falcão, "An iterative spanning forest framework for superpixel segmentation," *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3477–3489, 2019.

[104] P. L. St-Charles, G. A. Bilodeau, and R. Bergevin, "SuBSENSE: A universal change detection method with local adaptive sensitivity," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 359–373, 2014.

[105] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High quality object detection and instance segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[106] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016.

[107] H. Zhang *et al.*, "ResNeSt: Split-attention networks," *arXiv preprint arXiv:2004.08955*, 2020.

[108] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015.

[109] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[110] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 7, pp. 971–987, 2002.

[111] M. Danelljan, G. Bhat, F. Shahbaz Khan, and M. Felsberg, "ECO: Efficient convolution operators for tracking," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[112] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, "Robust visual tracking via hierarchical convolutional features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 11, pp. 2709–2723, 2018.

[113] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference*, 2014.

[114] V. N. Gangapure, S. Nanda, and A. S. Chowdhury, "Superpixel-based causal multisensor video fusion," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 6, pp. 1263–1272, 2017.

[115] X. Zhou and M. Belkin, "Semi-supervised learning by higher order regularization," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[116] F. Mahmood, N. Shahid, U. Skoglund, and P. Vandergheynst, "Adaptive graph-based total variation for tomographic reconstructions," *IEEE Signal Processing Letters*, vol. 25, no. 5, pp. 700–704, 2018.

[117] A. Jung, A. O. Hero III, A. C. Mara, S. Jahromi, A. Heimowitz, and Y. C. Eldar, "Semi-supervised learning in network-structured data via total variation minimization," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6256–6269, 2019.

[118] A. Chambolle and T. Pock, "A first-order primal-dual algorithm for convex problems with applications to imaging," *Journal of Mathematical Imaging and Vision*, vol. 40, no. 1, pp. 120–145, 2011.

[119] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[120] N. Perraudin, D. I. Shuman, G. Puy, and P. Vandergheynst, "UNLocBoX a Matlab convex optimization toolbox using proximal splitting methods," *arXiv preprint arXiv:1402.0779*.

[121] N. Komodakis and J. C. Pesquet, "Playing with duality: An overview of recent primal-dual approaches for solving large-scale optimization problems," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 31–54, 2015.

[122] Y. Wu, A. Kirillov, F. Massa, W. Y. Lo, and R. Girshick, "Detectron2." `https://github.com/facebookresearch/detectron2`, 2019.

[123] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, "GSPBOX: A toolbox for signal processing on graphs," *arXiv preprint arXiv:1408.5781*, 2014.

[124] A. Sobral and T. Bouwmans, "BGS library: A library framework for algorithm's evaluation in foreground/background segmentation," in *Handbook on "Background Modeling and Foreground Detection for Video Surveillance", Chapter 23*, 2014.

[125] A. Sobral, T. Bouwmans, and E. Zahzah, "LRSLibrary: Low-rank and sparse tools for background modeling and subtraction in videos," in *Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing*, CRC Press, Taylor and Francis Group, 2015.

[126] S. Isik, K. Özkan, S. Günal, and Ö. N. Gerek, "SWCD: a sliding window and self-regulated learning-based background updating method for change detection in videos," *Journal of Electronic Imaging*, vol. 27, no. 2, pp. 1–11, 2018.

[127] R. Wang, F. Bunyak, G. Seetharaman, and K. Palaniappan, "Static and moving object detection using flux tensor with split gaussian models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2014.

[128] S. Jiang and X. Lu, "WeSamBE: A weight-sample-based method for background subtraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 9, pp. 2105–2115, 2017.

[129] S. H. Lee, G. C. Lee, J. Yoo, and S. Kwon, "WisenetMD: motion detection using dynamic background region analysis," *Symmetry*, vol. 11, no. 5, p. 621, 2019.

[130] M. Braham, S. Piérard, and M. Van Droogenbroeck, "Semantic background subtraction," in *IEEE International Conference on Image Processing*, 2017.

[131] O. Barnich and M. Van Droogenbroeck, "ViBe: A universal background subtraction algorithm for video sequences," *IEEE Transactions on Image Processing*, vol. 20, no. 6, pp. 1709–1724, 2010.

[132] J. He, L. Balzano, and A. Szlam, "Incremental gradient on the grassmannian for online foreground and background separation in subsampled video," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2012.

[133] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[134] Z. Kang, C. Peng, and Q. Cheng, "Robust PCA via nonconvex rank approximation," in *IEEE International Conference on Data Mining*, 2015.

[135] Z. Wang, M. J. Lai, Z. Lu, W. Fan, H. Davulcu, and J. Ye, "Orthogonal rank-one matrix pursuit for low rank matrix completion," *SIAM Journal on Scientific Computing*, vol. 37, no. 1, pp. A488–A514, 2015.

[136] A. Cioppa, M. Van Droogenbroeck, and M. Braham, "Real-time semantic background subtraction," in *IEEE International Conference on Image Processing*, 2020.

[137] M. O. Tezcan, P. Ishwar, and J. Konrad, "BSUV-Net 2.0: Spatio-temporal data augmentations for video-agnostic supervised background subtraction," *IEEE Access*, vol. 9, pp. 53849–53860, 2021.

[138] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[139] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ADE20K dataset," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[140] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer Assisted Intervention*, 2015.

[141] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015.

[142] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[143] Z. H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, vol. 5, no. 1, pp. 44–53, 2018.

[144] F. Z. Xing, E. Cambria, W. B. Huang, and Y. Xu, "Weakly supervised semantic segmentation with superpixel embedding," in *IEEE International Conference in Image Processing*, 2016.

[145] D. Lin, J. Dai, J. Jia, K. He, and J. Sun, "ScribbleSup: Scribble-supervised convolutional networks for semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016.

[146] M. Pu, Y. Huang, Q. Guan, and Q. Zou, "GraphNet: Learning image pseudo annotations for weakly-supervised semantic segmentation," in *ACM Multimedia*, 2018.

[147] B. Zhang, J. Xiao, J. Jiao, Y. Wei, and Y. Zhao, "Affinity attention graph neural network for weakly supervised semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[148] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107637, 2021.

[149] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[150] L. C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *European Conference on Computer Vision*, 2018.

[151] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015.

[152] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015.

[153] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *International Conference on Learning Representations*, 2016.

[154] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *IEEE/CVF International Conference on Computer Vision*, 2011.

[155] M. Tang, A. Djelouah, F. Perazzi, Y. Boykov, and C. Schroers, "Normalized cut loss for weakly-supervised CNN segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[156] P. Krähenbühl and V. Koltun, "Efficient inference in fully connected CRFs with Gaussian edge potentials," in *Advances in Neural Information Processing Systems*, 2011.

[157] P. Vernaza and M. Chandraker, "Learning random-walk label propagation for weakly-supervised semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[158] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "OGB-LSC: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

[159] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[160] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[161] J. N. Kather *et al.*, "Multi-class texture analysis in colorectal cancer histology," *Scientific Reports*, vol. 6, p. 27988, 2016.

[162] K. Lang, "NewsWeeder: Learning to filter netnews," in *Journal of Machine Learning Research*, 1995.

[163] D. Anguita, A. Ghio, L. Oneto, X. Parra Perez, and J. L. Reyes Ortiz, "A public domain dataset for human activity recognition using smartphones," in *International European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013.

[164] M. Fanty and R. Cole, "Spoken letter recognition," in *Advances in Neural Information Processing Systems*, 1991.

[165] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[166] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[167] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[168] V. Kalofolias, "How to learn a graph from smooth signals," in *International Conference on Artificial Intelligence and Statistics*, 2016.

[169] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning Laplacian matrix in smooth graph signal representations," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6160–6173, 2016.

[170] V. Kalofolias and N. Perraudin, "Large scale graph learning from smooth signals," in *International Conference on Learning Representations*, 2019.

[171] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[172] S. Abu-El-Haija *et al.*, "MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *International Conference on Machine Learning*, 2019.

[173] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti, "SIGN: Scalable inception graph neural networks," in *International Conference on Machine Learning - Workshops*, 2020.

[174] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," in *International Conference on Learning Representations*, 2020.

[175] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*, 2019.

[176] B. Nica, *A brief introduction to spectral graph theory*. European Mathematical Society Publishing House, 2018.

[177] D. Patterson *et al.*, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.

[178] J. Du, S. Zhang, G. Wu, J. M. Moura, and S. Kar, "Topology adaptive graph convolutional networks," *arXiv preprint arXiv:1710.10370*, 2017.

[179] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[180] X. Bresson and T. Laurent, "Residual gated graph ConvNets," *arXiv preprint arXiv:1711.07553*, 2017.

[181] C. Morris *et al.*, "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *AAAI Conference on Artificial Intelligence*, 2019.

[182] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2019.

[183] Z. Chen, S. Villar, L. Chen, and J. Bruna, "On the equivalence between graph isomorphism testing and function approximation with GNNs," in *Advances in Neural Information Processing Systems*, 2019.

[184] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning*, 2020.

[185] Y. Rong, W. Huang, T. Xu, and J. Huang, "DropEdge: Towards deep graph convolutional networks on node classification," in *International Conference on Learning Representations*, 2020.

[186] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2020.

[187] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *International Conference on Learning Representations - Workshops*, 2019.

[188] A. Gadde, A. Anis, and A. Ortega, "Active semi-supervised learning using sampling theory for graph signals," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2014.

[189] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI Conference on Artificial Intelligence*, 2020.

[190] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *International Conference on Learning Representations*, 2022.

[191] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," in *International Conference on Learning Representations*, 2019.

[192] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," in *International Conference on Learning Representations*, 2020.

[193] L. Zhao and L. Akoglu, "PairNorm: Tackling oversmoothing in GNNs," in *International Conference on Learning Representations*, 2020.

[194] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," in *Advances in Neural Information Processing Systems*, 2020.

[195] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," in *Advances in Neural Information Processing Systems*, 2020.

[196] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized PageRank graph neural network," in *International Conference on Learning Representations*, 2021.

[197] J. Cheeger, "A lower bound for the smallest eigenvalue of the Laplacian," *Problems in analysis*, vol. 625, no. 195-199, p. 110, 1970.

[198] Y. Ollivier, "Ricci curvature of markov chains on metric spaces," *Journal of Functional Analysis*, vol. 256, no. 3, pp. 810–864, 2009.

[199] J. Jost and S. Liu, "Ollivier's Ricci curvature, local clustering and curvature-dimension inequalities on graphs," *Discrete & Computational Geometry*, vol. 51, no. 2, pp. 300–322, 2014.

[200] H. Zeng *et al.*, "Decoupling the depth and scope of graph neural networks," in *Advances in Neural Information Processing Systems*, 2021.

[201] W. Huang, Y. Li, W. Du, R. Y. Da Xu, J. Yin, L. Chen, and M. Zhang, "Towards deepening graph neural networks: A GNTK-based optimization perspective," in *International Conference on Learning Representations*, 2022.

[202] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," in *Advances in Neural Information Processing Systems*, 2019.

[203] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Transactions on Signal Processing*, vol. 62, no. 12, pp. 3042–3054, 2014.

[204] R. Hamilton, "The Ricci flow on surfaces," *Mathematics and General Relativity*, vol. 71, pp. 237–262, 1998.

[205] A. Nilli, "On the second eigenvalue of a graph," *Discrete Mathematics*, vol. 91, no. 2, pp. 207–210, 1991.

[206] Y. Lin, L. Lu, and S. T. Yau, "Ricci curvature of graphs," *Tohoku Mathematical Journal, Second Series*, vol. 63, no. 4, pp. 605–627, 2011.

[207] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, pp. 1–22, 2021.

[208] J. Tang, J. Sun, C. Wang, and Z. Yang, "Social influence analysis in large-scale networks," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[209] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[210] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu, "Query-driven active surveying for collective classification," in *International Workshop on Mining and Learning with Graphs*, 2012.

[211] Y. Ma, X. Liu, N. Shah, and J. Tang, "Is homophily a necessity for graph neural networks?," in *International Conference on Learning Representations*, 2022.

[212] A. Anis, A. Gadde, and A. Ortega, "Towards a sampling theorem for signals on arbitrary graphs.," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

[213] H. Shomorony and A. S. Avestimehr, "Sampling large data on graphs," in *IEEE Global Conference on Signal and Information Processing*, 2014.

[214] A. Gavili and X. P. Zhang, "On the shift operator, graph frequency, and optimal filtering in graph signal processing," *IEEE Transactions on Signal Processing*, vol. 65, no. 23, pp. 6303–6318, 2017.

[215] F. Hua, R. Nassif, C. Richard, H. Wang, and A. H. Sayed, "Online distributed learning over graphs with multitask graph-filter models," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 6, pp. 63–77, 2020.

[216] W. Xia, J. Chen, and L. Yu, "Distributed adaptive multi-task learning based on partially observed graph signals," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 522–538, 2021.

[217] P. Di Lorenzo, S. Barbarossa, P. Banelli, and S. Sardellitti, "Adaptive least mean squares estimation of graph signals," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 4, pp. 555–568, 2016.

[218] S. P. Chepuri and G. Leus, "Graph sampling for covariance estimation," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 3, pp. 451–466, 2017.

[219] D. Valsesia, G. Fracastoro, and E. Magli, "Sampling of graph signals via randomized local aggregations," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 2, pp. 348–359, 2018.

[220] A. Venkitaraman, S. Chatterjee, and P. Händel, "Predicting graph signals using kernel regression where the input signal is agnostic to a graph," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 4, pp. 698–710, 2019.

[221] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *International Conference on Computational Learning Theory*, 2004.

[222] S. Chen, R. Varma, A. Singh, and J. Kovačević, "Signal recovery on graphs: Fundamental limits of sampling strategies," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 4, pp. 539–554, 2016.

[223] S. Chen, A. Sandryhaila, J. M. F. Moura, and J. Kovačević, "Signal recovery on graphs: Variation minimization," *IEEE Transactions on Signal Processing*, vol. 63, no. 17, pp. 4609–4624, 2015.

[224] N. Perraudin and P. Vandergheynst, "Stationary signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3462–3477, 2017.

[225] F. Grassi, A. Loukas, N. Perraudin, and B. Ricaud, "A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 817–829, 2017.

[226] X. Wang, M. Wang, and Y. Gu, "A distributed tracking algorithm for reconstruction of graph signals," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 728–740, 2015.

[227] M. Tsitsvero, S. Barbarossa, and P. Di Lorenzo, "Signals on graphs: Uncertainty principle and sampling," *IEEE Transactions on Signal Processing*, vol. 64, no. 18, pp. 4845–4860, 2016.

[228] P. Di Lorenzo, S. Barbarossa, and P. Banelli, "Sampling and recovery of graph signals," in *Cooperative and Graph Signal Processing*, pp. 261–282, Elsevier, 2018.

[229] G. Ortiz-Jiménez, M. Coutino, S. P. Chepuri, and G. Leus, "Sampling and reconstruction of signals on product graphs," in *IEEE Global Conference on Signal and Information Processing*, 2018.

[230] J. S. Arora, "More on numerical methods for unconstrained optimum design," in *Introduction to Optimum Design*, pp. 455–509, Boston: Academic Press, fourth ed., 2017.

[231] E. Isufi, A. Loukas, N. Perraudin, and G. Leus, "Forecasting time series with VARMA recursions on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 18, pp. 4870–4885, 2019.

[232] H. W. Hethcote, "The mathematics of infectious diseases," *SIAM Review*, vol. 42, no. 4, pp. 599–653, 2000.

[233] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[234] R. A. Varma and J. Kovačević, "Sampling theory for graph signals on product graphs," in *IEEE Global Conference on Signal and Information Processing*, 2018.

[235] G. Visick, "A quantitative version of the observation that the Hadamard product is a principal submatrix of the Kronecker product," *Linear Algebra and its Applications*, vol. 304, no. 1-3, pp. 45–68, 2000.

[236] A. Horn, "Eigenvalues of sums of hermitian matrices," *Pacific Journal of Mathematics*, vol. 12, no. 1, pp. 225–241, 1962.

[237] A. Knutson and T. Tao, "Honeycombs and sums of hermitian matrices," *Notices American Mathematical Society*, vol. 48, no. 2, 2001.