Graph Learning and Optimization for Irregular-Structured Signal Processing

Saghar Bagheri

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

May 2025

Abstract

Graph Signal Processing (GSP) extends harmonic analysis tools, such as Fourier transforms and wavelets, to discrete signals defined on finite graphs, enabling tasks like signal denoising, prediction, and interpolation on irregular domains. A critical first step in GSP is to learn an appropriate graph that captures pairwise similarities or correlations inherent in the data, ensuring that subsequent graph-based filtering effectively leverages local structure for improved performance.

However, most existing graph learning methods assume static relationships, while real-world interactions often evolve over time. To address this problem, this thesis proposes a slowly time-varying graph learning framework that models the difference between consecutive adjacency matrices as a low-rank matrix. This approach accommodates gradual shifts in node-to-node similarities over time, enabling efficient graph updates with low computational overhead while maintaining alignment with the underlying data.

Beyond graph construction, the challenge of *dense* or *complete* graphs often arises, particularly in large-scale applications where representing all possible edges is computationally prohibitive. To address this issue, this thesis introduces a sparsification method guided by the Fiedler number, the second smallest eigenvalue of the Laplacian, which quantifies graph connectivity. By removing edges that minimally affect

the Fiedler number, the resulting sparser graph preserves essential connectivity while significantly reducing training and inference costs for deep learning models (*e.g.*, graph convolutional networks (GCNs)). Together, these contributions provide a flexible and computationally efficient approach to GSP in dynamic and large-scale graph settings.

Originality Statement

This dissertation was written by Saghar Bagheri under the supervision of Professor Gene Cheung. Parts of the work presented here have been published or submitted for publication in peer-reviewed journals and conference proceedings, as detailed below. The research in each publication was primarily conducted by the principal author, with guidance and support from senior authors. Collaboration with other co-authors involved data acquisition and preprocessing.

1. S. Bagheri, G. Cheung, T. Eadie, A. Ortega, *"Joint Time-varying Graph Estimation / Signal Interpolation via Smoothness and Low-Rank Priors,"* in preparation for *IEEE Transactions on Signal Processing.*

2. S. Bagheri, G. Cheung, T. Eadie, A. Ortega, *"Joint Signal Interpolation / Time-Varying Graph Estimation via Smoothness and Low-rank Priors," IEEE International Conference on Acoustics, Speech and Signal Processing*, Seoul, South Korea, April 2024.

3. S. Bagheri, G. Cheung, T. Eadie, *"Graph Sparsification for GCN towards Optimal Crop Yield Prediction," International Geoscience and Remote Sensing Symposium (IGARSS)*, Pasadena, CA, July 2023.

4. S. Bagheri, T. T. Do, G. Cheung, A. Ortega, *"Spectral Graph Learning with Core*

Eigenvectors Prior via Iterative GLASSO and Projection," IEEE Transactions on Signal Processing, vol. 72, pp. 3958–3972, 2024.

5. S. Bagheri, C. Dinesh, G. Cheung, T. Eadie, *"Unsupervised Graph Spectral Feature Denoising for Crop Yield Prediction," EUSIPCO'22*, Belgrade, Serbia, August 2022.

Acknowledgments

This dissertation represents the culmination of a deeply rewarding academic journey, made possible by the guidance, support, and encouragement of many wonderful individuals.

I am deeply grateful to my supervisor, Professor Gene Cheung, for his insightful mentorship, clear guidance, and unwavering support throughout my doctoral studies. His thoughtful feedback and dedication have significantly shaped this research and enriched my growth as a scholar.

I am truly thankful to my colleagues in the GISP laboratory at York University and my co-authors for fostering a collaborative and supportive environment. Their ideas, kindness, and camaraderie have made this experience profoundly meaningful. I also extend my gratitude to my supervisory committee for their valuable insights and generous investment of time, which have greatly strengthened this work.

Above all, I am immensely grateful to my family, partner, and friends. Their steadfast belief in me and unwavering support through every challenge and milestone have been the foundation of this achievement. Their encouragement means more to me than words can express.

Table of Contents

List of Figures

Chapter 1

Introduction

Modern data often exhibit inherent correlation structures that can be effectively modeled using graphs. A graph consists of nodes representing data points (*e.g.*, temperature readings from sensors in a rainforest) and edges encoding pairwise relationships (*e.g.*, feature distances or conditional correlations) [2, 3]. Such data are increasingly prevalent in real-world applications, including social networks, wireless sensor networks, transportation systems, and biological networks [4–8].

Graph Signal Processing (GSP) [9–11] provides mathematical tools, such as transforms and wavelets, for processing discrete signals defined on graphs, where the graph serves as a structured kernel capturing correlations between nodes. A statically defined graph[1] enables filters to exploit locally connected neighborhoods for tasks such as denoising, dequantization, compression, deblurring, matrix completion, and interpolation [15–27]. However, the optimal graph structure for a given dataset is often unknown *a priori*, necessitating a *graph learning* procedure. This thesis focuses on learning such graphs from data.

[1]Though the spectral analysis of *signed* graphs with both positive and negative edges has been studied recently [12–14], this thesis focuses on the more common graphs with *positive* edge weights.

While many conventional approaches assume *static* graphs, node-to-node similarities in real-world applications often evolve slowly over time. For example, crop yields in neighboring counties in the Corn Belt may become increasingly correlated as farmers adopt similar cultivation practices [28,29]. To capture these evolving pairwise similarities, the graph must be periodically updated to maintain an accurate representation of the underlying data structure. However, learning a *dynamic* graph is challenging, particularly with sparse observations [30]. Many existing dynamic graph learning methods impose regularization directly on the nodal domain [31, 32], constraining the extent of each update. In contrast, this dissertation proposes modeling the difference between consecutive adjacency matrices $\mathbf{W}^{(t+1)}$ and $\mathbf{W}^{(t)}$ as a *low-rank* matrix. While low-rank priors are commonly used in recommendation systems [33] and community/anomaly detection [6], their application to capturing temporal evolution in *similarity graphs* is underexplored. Empirical results on real-world datasets validate this approach [34].

Another challenge arises when learning a *dense* or *complete* graph that encodes all deducible pairwise similarities. Representing such graphs as dense adjacency or Laplacian matrices incurs significant computational overhead, particularly for algorithms such as *conjugate gradient* (CG) [35] when solving linear systems or for deep networks such as *graph convolutional networks* (GCNs) [1,36–38]. Traditional GSP graph sparsification methods, such as k-Nearest Neighbors (kNN) or ϵ-thresholding [9,10], often remove edges with significant weights or risk disconnecting the graph into subgraphs. To address this, this dissertation proposes a novel graph sparsification method based on the *Fiedler number*—the second smallest eigenvalue of the Laplacian matrix [39].

By iteratively removing edges with minimal impact on the Fiedler number, this approach preserves core connectivity while capturing the most critical relationships in the data [12, 40].

In summary, this dissertation advances graph learning in GSP through two key contributions. First, it introduces a method to learn slowly time-varying graphs using low-rank updates for consecutive adjacency matrices. Second, it proposes a Fiedler-based sparsification strategy to reduce computational overhead in dense graphs.

The remainder of this thesis is organized as follows. Chapter;2 surveys existing methods for graph construction, dynamic graph modeling, graph signal restoration, graph algorithm unrolling, and graph sparsification. Chapter;4 and 5 detail the proposed methods and experimental results for learning time-varying graphs and efficient graph sparsification, respectively. Finally, Chapter;6 presents conclusions and outlines directions for future research.

Chapter 2

Related Works

This chapter reviews representative methods for static graph learning, encompassing both statistical and feature-based approaches, in Section A. It then examines techniques for modeling slowly time-varying graphs in Section B. The chapter also surveys key methods for graph signal restoration within the graph signal processing (GSP) literature in Section C, followed by an overview of graph algorithm unrolling in Section D. Finally, it explores various graph sparsification strategies in Section E.

A Static Graph Learning

The field of *Graph Signal Processing* (GSP) studies the processing and analysis of discrete signals on structured kernels described by finite graphs [9, 10]. With a graph that captures the underlying similarity/correlation structure, computational methods can be developed to process data on this graph for various signal processing tasks, such as compression, denoising, and interpolation [17, 41]. Determining the optimal graph structure for a dataset is not always straightforward, presenting a technical challenge in graph learning and optimization for GSP applications.

There are two general approaches to static graph learning from data: (i) statistical

graph learning, and (ii) feature-based graph learning.

A.1 Statistical Graph Learning

Statistical approaches assume that multiple observations drawn from the same probabilistic generative model are available to deduce a graph structure that parameterizes the model. Specifically, denote by $\mathbf{Y} \in \mathbb{R}^{N \times M}$ a data matrix whose columns are the M signal observations $\{\mathbf{y}_i\}_{i=1}^{M}$, where $\mathbf{y}_i \in \mathbb{R}^N$. Given $\mathbf{Y}$, the methods proposed in [42–45] estimate parameters of a graphical probability model, $\mathcal{G}$. The most common model class is the *Gaussian Markov random field* (GMRF) [46], where the absence of an edge (i, j) in graph $\mathcal{G}$ indicates conditional independence (given all the other variables) between random variables i and j, represented as graph nodes. The joint probability of a zero-mean GMRF for random vector $\mathbf{x} \in \mathbb{R}^N$ is

$$p(\mathbf{x}|\mathbf{\Theta}) = \frac{|\mathbf{\Theta}|^{1/2}}{(2\pi)^{N/2}} \exp\left(-\frac{1}{2}\mathbf{x}^\top \mathbf{\Theta}\mathbf{x}\right), \tag{2.1}$$

where $\mathbf{\Theta}$ is the PD *precision* or inverse covariance matrix. Given $\mathbf{Y}$, learning the precision matrix $\mathbf{\Theta}$ (typically assumed sparse to reduce parameterization) is equivalent to learning an underlying (sparse) graph $\mathcal{G}$, where $\mathbf{\Theta}$ is interpreted as a *generalized* graph Laplacian [9] (see Section A.2 for definition).

One popular approach to estimate a sparse $\mathbf{\Theta}$ is the GLASSO [45], formulated as

$$\max_{\mathbf{\Theta}} \quad \log\det\mathbf{\Theta} - \mathrm{Tr}(\mathbf{\Sigma}\mathbf{\Theta}) - \rho\,\|\mathbf{\Theta}\|_1, \tag{2.2}$$

where $\mathbf{\Sigma}$ is the input empirical covariance matrix computed from (typically limited)

signal observations $\mathbf{Y}$, and $\rho > 0$ is a weight parameter for the ℓ_1-norm term—convexification of the non-convex ℓ_0-norm sparsity term. The objective of (2.2) was modified in [47, 48] with additional constraints to ensure that the solution $\mathbf{\Theta}$ has a desirable form, *e.g.*, a combinatorial graph Laplacian [47].

In [48], assumptions on *eigenvalues* of the graph Laplacian matrix are introduced in the learning algorithm. Different choices of eigenvalue constraints are introduced to ensure the solutions have specific nodal-domain graph structures assumed to be known *a priori*. For example, a graph can be learned so that the first eigenvalue $\lambda_1 = 0$ of the combinatorial graph Laplacian matrix has multiplicity k, ensuring the graph has k connected components; *i.e.*, eigenvalue set $\mathcal{S}_\lambda$ is of the form:

$$\mathcal{S}_\lambda = \{\{\lambda_j = 0\}_{j=1}^{k}, c_1 \leq \lambda_{k+1} \leq ... \leq \lambda_N \leq c_2\} \tag{2.3}$$

where c_1, c_2 are constant parameters depending on the graph edges [49, 50].

A.2 Feature-based Graph Learning

In feature graph learning, a graph is constructed based on the (typically learned) *features* associated with each data point [17, 21, 51]. Specifically, suppose that each data point i is endowed with a *feature vector* $\mathbf{f}_i \in \mathbb{R}^K$. A *feature distance* $d(\mathbf{f}_i, \mathbf{f}_j)$ between nodes i and j can then be computed, and a corresponding non-negative edge weight is subsequently calculated as

$$w_{i,j} = \exp(-d(\mathbf{f}_i, \mathbf{f}_j)). \tag{2.4}$$

A classical definition of feature distance is the *Mahalanobis distance*:

$$d(\mathbf{f}_i, \mathbf{f}_j) = (\mathbf{f}_i - \mathbf{f}_j)^\top \mathbf{M} (\mathbf{f}_i - \mathbf{f}_j) \tag{2.5}$$

where $\mathbf{M} \succ 0$ is a *positive definite* (PD) metric matrix [52]. The optimization of matrix $\mathbf{M}$, known as *metric learning*, has been extensively studied in the literature [40, 53–58]. Feature graph learning is particularly effective in scenarios where multiple graph signal observations are unavailable, thus making it difficult to compute a reliable empirical covariance matrix. Instead, it leverages domain knowledge to deduce reasonable features for each node. However, one limitation of feature graph learning is its dependence on non-negative feature distances, which removes the possibility of negative edge weights capturing pairwise dissimilarities or anti-correlations in a *signed graph*. Recently, signed graphs have demonstrated their usefulness in several applications [13, 59, 60].

Different GSP applications have different unique requirements and demand different methods to define pairwise similarities/correlations, or edge weights between nodes for graph construction [61–63]. For example, in temperature sensing systems, where sensors have known physical 2D locations, pairwise similarities between nodes can be based on the physical distances between sensors [61]. On the other hand, in machine learning applications, pairwise similarities are typically defined as the distance between feature vectors $\mathbf{f}_i$ in feature space, learned directly from data using *convolutional neural nets* (CNN), for example [64, 65].

The *bilateral filter* [66, 67] is one specific distance notion commonly used in image

processing. Specifically, edge weight $w_{i,j}$ between nodes i and j is defined as

$$w_{i,j} = \exp\left(-\frac{\|\mathbf{l}_i - \mathbf{l}_j\|_2^2}{2\sigma_l^2}\right) \exp\left(-\frac{|x_i - x_j|^2}{2\sigma_x^2}\right), \tag{2.6}$$

where σ_l and σ_x are parameters, and $\mathbf{l}_i \in \mathbb{R}^2$ and $x_i \in \mathbb{R}$ are the respective 2D pixel grid location and pixel intensity of pixel (node) i. Distance $d_{i,j}$ in this case is

$$d_{i,j} = \frac{\|\mathbf{l}_i - \mathbf{l}_j\|_2^2}{2\sigma_l^2} + \frac{|x_i - x_j|^2}{2\sigma_x^2}, \tag{2.7}$$

and $w_{i,j} = \exp(-d_{i,j})$. Some works in image processing [68] also use the simpler form of (2.6), where only pixel intensities are considered to compute edge weights:

$$w_{i,j} = \exp\left(-\frac{|x_i - x_j|^2}{2\sigma_x^2}\right). \tag{2.8}$$

(2.8) means that an edge weight is small when two pixels have a large difference in intensity, such as pixels on two different sides of an object boundary.

B Learning of Slowly Time-varying Graphs

As discussed in Chapter 1, many real-world datasets exhibit node-to-node relationships that evolve slowly over time [28,29]. In such cases, a *static* similarity graph may fail to capture these continuous shifts, thus potentially leading to suboptimal performance in tasks such as filtering, interpolation, or prediction. Accordingly, various methods have been proposed to learn and parameterize *slowly time-varying* graphs in a way that accurately reflects changing similarities. These approaches can be categorized into three main groups based on their assumptions about the graph structure,

along with a fourth category leveraging deep learning techniques.

B.1 Covariance Matrix Re-estimation

One approach to handling time-varying graphs is to re-estimate covariance matrices at different time instants, assuming that matrix entries vary smoothly [69, 70]. The main idea is to compute (or update) a covariance matrix whenever new data becomes available, typically using a sliding window or a batch of recent observations. By interpreting each covariance matrix as an adjacency-like structure, one can capture evolving pairwise relationships among nodes. However, this strategy depends on having sufficient data samples at each time step to ensure that covariance estimates remain accurate. In data-starved scenarios—such as those involving infrequent or costly measurements—covariance re-estimation may yield noisy or unreliable results. For example, if we have limited data points due to seasonal variations in crop yields, the re-estimation of covariance matrices may not provide accurate results.

B.2 Nodal Domain Assumptions

A second category of methods makes structural assumptions in the *nodal* domain, such as sparsity in graph topology and/or smoothness of topology over time [31, 32]. For instance, [32] assumes that the graph edges change smoothly over time, *i.e.*, the Frobenius norm of the change in the adjacency matrix is small. Meanwhile, [31] assumes that only a few graph edges require updating, using a sparsity prior on the difference between two successive adjacency matrices. This approach modifies the smoothness-based optimization method from [32] by incorporating an ℓ_1-regularization term for the temporal variation. Such nodal-domain assumptions are particularly appealing

when a significant fraction of edge weights remain stable over time, as they explicitly encourage minimal but targeted adjustments. Nevertheless, they may still require careful tuning of regularization parameters to balance between flexibility (allowing essential structural changes) and stability (preventing excessive fluctuations).

B.3 Spectral Domain Assumptions

The third category involves assumptions in the *spectral* domain of the graph, specifically, the first k singular vectors (or eigenvectors) of the graph remain constant over time, while the remaining $N - k$ singular vectors are re-estimated [33, 71, 72]. These methods leverage the spectral properties of the graph, focusing on the most significant singular vectors that capture the core structure of the graph. Specifically, they assume that the remaining $N - k$ singular vectors, which represent finer details, are updated to reflect changes. This implies no significant changes in the core static low-rank structure of the adjacency matrices over time. Such an approach can be especially effective when long-term structural consistency is expected in the data. However, if large shifts occur in the dominant singular vectors, this method may lose its effectiveness, as it relies on the stability of the top k singular vectors over time.

B.4 Deep Learning Based

Beyond these parametric or model-based methods, *graph neural networks* (GNNs) offer a data-driven framework to model time-varying graphs. By integrating temporal modeling techniques (*e.g.*, recurrent units or temporal convolutions) with graph convolutions, GNNs aim to capture both evolving spatial and temporal patterns [73–75]. *Dynamic GNNs* [76–78] further adjust network parameters to changes in topology,

while *lifelong learning* on graphs [79–81] attempts to maintain performance over extended time horizons. Despite their flexibility, these deep learning approaches often require significant training data and may lack clear mathematical interpretability when assessing generalization or performance guarantees. In scenarios with limited observations, purely data-driven methods might struggle to learn meaningful updates to the adjacency matrix, necessitating the use of additional priors or explicit constraints.

In contrast to the above methods, which employ covariance re-estimation, nodal constraints, spectral assumptions, and end-to-end GNN training, this thesis assumes that the difference between two consecutive adjacency matrices, $\mathbf{W}^{(t+1)}$, $\mathbf{W}^{(t)}$, can be effectively approximated by a low-rank matrix. Unlike previous works, this low-rank prior remains relevant even in data-scarce conditions, offering a structured yet efficient approach to track gradual changes in node-to-node similarities [30–32, 71, 72, 82].

C Graph Signal Restoration

Graph-based signal restoration methods leverage structured data kernels specified by graphs to aid the recovery process. A graph $\mathcal{G}$ of N nodes can be specified by its adjacency matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$, where $W_{i,j}$ is the edge weight connecting nodes i and j; $W_{i,j} = 0$ if there is no edge connecting i and j. A common smoothness prior for graph signal restoration is the quadratic *graph Laplacian regularizer* (GLR) [17],

$$\text{GLR}(\mathbf{x}) = \mathbf{x}^\top \mathbf{L} \mathbf{x} \tag{2.9}$$

which quantifies smoothness of a signal $\mathbf{x}$ given a graph $\mathcal{G}$ specified by graph Laplacian $\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}$ (where $\mathbf{D}$ is the diagonal degree matrix for $D_{i,i} = \sum_j W_{i,j}$), and has been applied to tasks like denoising [17], dequantization [19], and interpolation [25]. GLR will be discussed in more technical detail in Section A.4, as it plays a key role in the interpolation framework presented in Section A.1.

An alternative prior interprets the adjacency matrix $\mathbf{W}$ of a graph as a shift operator, leading to the *graph shift variation* (GSV) [83]:

$$\text{GSV}(\mathbf{x}) = \|\mathbf{x} - \frac{1}{|\lambda_{\max}|}\mathbf{W}\mathbf{x}\|_2^2 \tag{2.10}$$

where $|\lambda_{\max}|$ is the *spectral radius* (largest eigenvalue in magnitude) of $\mathbf{W}$. In words, GSV (2.10) states that the target signal $\mathbf{x}$ and its shifted version $\mathbf{W}\mathbf{x}$ should be similar. Note that GSV offers a notion of signal smoothness that is well defined even if the underlying graph is directed (in which case $\mathbf{W}$ is asymmetric).

In a subsequent work, *Left Eigenvectors of the Random Walk Graph Laplacian* (LeRAG) [19] introduces an alternative smoothness prior similar to GSV. First, define the *random walk graph Laplacian* as $\mathbf{L}_r \triangleq \mathbf{D}^{-1}\mathbf{L}$. Then, define the LeRAG prior as

$$\text{LeRAG}(\mathbf{x}) = \mathbf{x}^\top \mathbf{L}_r^\top \mathbf{L}_r \mathbf{x}. \tag{2.11}$$

Note that, ignoring the scalar $\frac{1}{|\lambda_{\max}|}$ for simplicity, GSV can be written as $\|(\mathbf{I} - \mathbf{W})\mathbf{x}\|_2^2 = \|\mathcal{L}\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathcal{L}^\top \mathcal{L}\mathbf{x}$, where $\mathcal{L} \triangleq \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$ is the normalized graph Laplacian. LeRAG utilizes the left eigenvectors of $\mathbf{L}_r$, which provides similar low-pass filtering properties as GLR but with normalized frequencies. This method was used for soft decoding of JPEG-compressed images.

Beyond these quadratic priors, signal smoothness can also be enforced via an ℓ_1-norm based formulation known as *Graph total variation* (GTV) [84, 85]. GTV minimizes the weighted absolute differences of samples between connected nodes. It was shown that the signal-dependent GTV converges faster to piecewise constant (PWC) signal reconstruction and was successfully employed in image deblurring [24].

Although minimizing signal-dependent GLR / GTV promotes PWC signal reconstruction, it can suffer from the well-known 'staircase' effect in slowly varying spatial regions. To alleviate this effect, a higher-order graph smoothness prior called the *gradient graph Laplacian regularizer* (GGLR) [25] has been proposed for image restoration problems. GGLR promotes piecewise-planar (PWP) reconstructions, thereby mitigating staircase artifacts and delivering higher-fidelity signal recovery than standard GLR.

D Graph Algorithm Unrolling

Algorithm unrolling [86] implements each iteration of a model-based iterative optimization algorithm as a neural layer, forming a feed-forward network when stacked sequentially. This approach enables end-to-end learning of identified parameters via back-propagation from data. A notable example is the unrolling of the *iterative soft thresholding algorithm* (ISTA) for sparse coding, resulting in the *Learned ISTA* (LISTA) network [87]. More recently, research demonstrated that unrolling an iterative algorithm designed to minimize a sparse rate reduction (SRR) objective can yield a family of interpretable "white-box" transformer-like neural networks [88].

To the best of current knowledge, the first deep unrolling of a graph-based algorithm is the *deep graph Laplacian regularizer* (DGLR) for image denoising [89].

Subsequent studies applied *deep graph total variation* (DGTV) to natural image denoising [90] and light field image denoising [91]. In [92], researchers unrolled an ADMM-based iterative algorithm minimizing an objective regularized by both graph total variation (GTV) and graph Laplacian regularizer (GLR). However, this approach assumes a statically fixed graph defined *a priori*, without learning the graph from data. Recent work in [93] showed that a graph learning module with edge weight normalization, tuned end-to-end from data, resembles the self-attention mechanism in a transformer, enabling unrolled graph-based algorithms to form lightweight transformer-like neural networks. Drawing inspiration from [93], this dissertation incorporates an initial graph learning module. However, in the context of a data-starved scenario, it leverages a proposed low-rank prior to update each adjacency matrix $\mathbf{W}^{(t)}$ at time t to $\mathbf{W}^{(t+1)}$ at the next time step $t+1$.

E Graph Sparsification

After an appropriate graph is learned from data, further *graph optimization* is often desirable, particularly when dealing with dense or *complete* graphs that connect every pair of nodes. While a complete graph $\mathcal{G}$—containing edges (i,j) that connect every node i to every other node j, with weights $w_{i,j}$ representing their pairwise similarities—is certainly the most informative graph structure, representing $\mathcal{G}$ as a dense matrix results in high computational and processing costs. This is a significant concern when dealing with very large graphs in practical applications such as iterative linear program solvers and deep graph convolutional networks.

The following subsections overview potential methods for sparsifying a complete graph. Each strategy offers different trade-offs in terms of implementation complexity,

edge-weight preservation, and overall graph connectivity.

E.1 k Nearest Neighbors

A widely employed technique for sparsifying a complete graph $\mathcal{G}$ is the *k-nearest neighbors* (kNN) algorithm. In this method, each node i retains only the k strongest connections $(i, j) \in \mathcal{E}$ based on edge weights $w_{i,j}$ and removes the others where possible. By discarding all but these top-k edges, the algorithm ensures a comparatively sparse graph structure.

This process is performed symmetrically, ensuring that if $w_{i,j}$ is one of k largest edge weights for node j, then edge (i, j) remains intact. This two-way approach guarantees that each node in the graph has at least k connected edges. One key advantage of kNN is its simplicity and relative computational efficiency since identifying the top-k edges for each node can be performed independently. Additionally, guaranteeing a minimum degree of k for each node maintains the connectivity of the sparsified graph. However, a potential disadvantage is that some edges with large weights $w_{i,j}$ might get discarded simply because they are not in the top-k connections of node j. Striking a balance between the choice of k and the desired level of sparsity is thus essential to preserve important links for effective filtering or inference.

E.2 ϵ-Thresholding

Another common method to sparsify a complete graph $\mathcal{G}$ is *ϵ-thresholding*. This approach involves removing all edges $(i, j) \in \mathcal{E}$ with weights $w_{i,j} < \epsilon$, where $\epsilon \in \mathbb{R}_+$ is a chosen positive parameter. By retaining only edges with relatively large weights, ϵ-thresholding typically aligns with the goal of preserving the most significant

similarities in the data. A major advantage of ϵ-thresholding is that it naturally maintains crucial information for capturing strong local correlations. However, a notable disadvantage of this method is that it can induce excessive local sparsification, especially if some nodes have many edges below ϵ. In extreme cases, this can lead to nodes becoming completely isolated, stopping their interaction with the rest of the graph. Such isolation negatively impacts the overall effectiveness of graph-based algorithms by reducing the filtering performance.

E.3 Hybrid kNN / ϵ-Thresholding

A *hybrid* of kNN and ϵ-thresholding can combine the strengths of both methods to sparsify a complete graph $\mathcal{G}$. In this approach, edges may be removed from each node i until it has an upper bound of $d_{\max}$ edges (similar to limiting to the top k), then edges are further removed if i) $w_{i,j} < \epsilon$; and ii) the number of edges $(i, j) \in \mathcal{E}$ connected to i remains more than $d_{\min}$. By appropriately selecting the parameters $d_{\max}$, $d_{\min}$ and ϵ, each node maintains a minimum of $d_{\min}$ connections to the graph, while edges (i, j) with small weights $w_{i,j} < \epsilon$ are removed. Although this hybrid approach typically requires more careful tuning than pure kNN and ϵ-thresholding, it can yield sparser graphs that retain crucial links for downstream GSP operations.

E.4 All-Pair Shortest Paths

A more computationally intensive strategy for reducing the density of a complete graph $\mathcal{G}$ involves iteratively removing edges based on a metric that evaluates the graph's "connectedness". One potential metric is the *all-pair shortest paths* (APSP), which computes the sum of the shortest paths between every pair of distinct nodes

$i, j \in \mathcal{N}$ in the graph such that $j \neq i$. The APSP can be determined using the known Floyd-Warshall algorithm, which has a complexity of $\mathcal{O}(N^3)$ [94]. Subsequently, edges are removed iteratively, choosing the edge whose removal minimally increases (or changes) the total APSP at each step. Although APSP-based methods can provide a more principled way to preserve global connectivity, their high computational cost often becomes prohibitive in large-scale scenarios. Nevertheless, they can yield finely tuned results by carefully pruning edges that have the least impact on overall distances, thereby maintaining essential connections within the sparsified graph.

Chapter 3

Preliminaries

This chapter presents the key concepts and theoretical foundations underpinning the methods developed in this dissertation. Section A outlines the fundamental principles of Graph Signal Processing, including graph definitions, the graph Laplacian and its spectrum, polynomial filtering, and the Graph Laplacian Regularizer (GLR). Section B examines Graph Convolutional Networks (GCNs), detailing their layer-wise propagation rule and addressing limitations such as oversmoothing, shallow architectures, and computational complexity. Together, these sections provide a concise overview of classical graph signal processing and modern deep learning approaches on graphs.

A Graph Signal Processing Definitions

Digital Signal Processing (DSP) focuses on analyzing and manipulating discrete signals sampled on regular data kernels, such as time samples for digital audio signals or pixel grids for digital images. In contrast, *Graph Signal Processing* (GSP) extends DSP concepts to signals residing on irregular data kernels represented by finite graphs [9, 10, 95].

A.1 Graph Definitions and Graph Signals

A graph is a mathematical construct representing pairwise relationships between data entities. It consists of a set of nodes, also known as vertices, interconnected by edges. These edges can be either weighted, where each edge is assigned a scalar value indicating the strength of the connection, or unweighted, where all edges have a uniform weight, typically set to one. Mathematically, a graph is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where $\mathcal{V}$ is a finite set of nodes with $|\mathcal{V}| = N$ indicating the number of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{W}$ is an adjacency matrix with dimension $N \times N$. For a weighted graph $\mathcal{G}$, an edge $(i, j) \in \mathcal{E}$ connects nodes i and j with an associated weight $w_{i,j} \in \mathbb{R}$, and is represented in the adjacency matrix as $W_{i,j} = w_{i,j}$. If there is no edge between nodes i and j, then $W_{i,j} = 0$. Additionally, the adjacency matrix $\mathbf{W}$ may contain self-loops, indicated by its diagonal elements $W_{i,i}$, each connecting node i with itself.

A graph can be either *directed* or *undirected*. In an undirected graph, an edge (i, j) allows traversal in both directions, and $W_{i,j} = W_{j,i}$. As a result, the adjacency matrix $\mathbf{W}$ of an undirected graph is symmetric. On the other hand, in a directed graph, edges (i, j) and (j, i) can have different weights, indicating directionality. Thus, the adjacency matrix $\mathbf{W}$ of a directed graph is typically not symmetric.

The weight of an edge connecting nodes i and j can be defined in various ways, depending on the application. Edges can represent pairwise similarities, dissimilarities, or distances between distinct feature vectors. Additionally, edge weights may represent statistical quantities such as conditional correlations, as discussed in Section A.

A *graph signal* $\mathbf{x}$ on $\mathcal{G}$ is a discrete signal of length N, where each signal sample

$x_i \in \mathbb{R}$ is assigned to a corresponding node i in $\mathcal{V}$. Thus, a graph signal can be represented as a vector $\mathbf{x} \in \mathbb{R}^N$. The interpretation of a graph signal varies by application, encompassing measurements such as temperature, image pixel intensity, or economic data mapped to farmlands or counties.

A.2 Graph Spectrum

Given an adjacency matrix $\mathbf{W}$ for an undirected graph, the diagonal *degree matrix* $\mathbf{D}$ is defined as

$$D_{i,i} = \sum_j W_{i,j} \tag{3.1}$$

The *combinatorial graph Laplacian matrix* $\mathbf{L}$ [9] is then defined as

$$\mathbf{L} \triangleq \mathbf{D} - \mathbf{W}. \tag{3.2}$$

If a graph is undirected, then $\mathbf{L}$ is real and symmetric, and hence it can be eigen-decomposed via the Spectral Theorem as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ [10], where $\mathbf{\Lambda}$ is a diagonal matrix with real eigenvalues λ_k on the diagonal, and $\mathbf{U}$ is a matrix composed of orthonormal eigenvectors as columns. Assuming that all edge weights are positive in $\mathbf{W}$, it can be proven that $\mathbf{L}$ is *positive semi-definite* (PSD), *i.e.*, eigenvalues of $\mathbf{L}$ are non-negative [10]. Eigenvalues can be interpreted as *graph frequencies*, and their corresponding eigenvectors can be interpreted as *graph Fourier modes*. The set of eigenvectors $\mathbf{U}^\top$ collectively define the *graph Fourier transform* (GFT) [95]. Specifically, graph signal $\mathbf{x}$ can be converted to its frequency representation via GFT using the equation $\boldsymbol{\alpha} = \mathbf{U}^\top\mathbf{x}$. GFT can be considered a generalization of known

discrete transforms such as the *discrete cosine transform* (DCT) [96].

There exist different variants of the graph Laplacian matrix, each with distinct properties. A *normalized graph Laplacian*, which is a symmetric normalized version of $\mathbf{L}$ is defined as

$$\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}. \tag{3.3}$$

A *random walk graph Laplacian*, which is an asymmetric normalized version of $\mathbf{L}$ is defined as

$$\mathbf{L}_r = \mathbf{D}^{-1}\mathbf{L}. \tag{3.4}$$

Finally, to properly account for self-loops, a *generalized graph Laplacian* is defined as

$$\mathbf{L}_g = \mathbf{L} + \text{diag}(\mathbf{W}). \tag{3.5}$$

Each variant also has its own unique spectral properties. For example, $\mathbf{L}_n$ and $\mathbf{L}_r$ have the same set of eigenvalues where $0 \leq \lambda_k \leq 2$. $\mathbf{L}_n$ and $\mathbf{L}$ are both symmetric, but $\mathbf{L}_n$ does not have the constant vector as a first eigenvector.

A.3 Graph Filters

A P-tap *graph filter* $P(\mathbf{L}) = \sum_{p=0}^{P} a_p \mathbf{L}^p$ is a polynomial of Laplacian $\mathbf{L}$ with filter coefficients $\{a_p\}_{p=0}^{P}$, which filters the signal $\mathbf{x}$ by selecting or attenuating specific graph frequencies. Given an eigen-decomposition of $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, the graph frequency response for filter $P(\mathbf{L}) = \sum_{p=0}^{P} a_p \mathbf{U}\mathbf{\Lambda}^p\mathbf{U}^\top$ is $f(\lambda) = \sum_{p=0}^{P} a_p \lambda^p$. Importantly, this

P-tap graph filter can be implemented distributedly within a P-hop neighborhood of each node, allowing for efficient local computations.

Efficient graph filters can be designed for various signal processing tasks, including graph signal compression, denoising, and interpolation. In most cases, it is assumed that the target signal is "smooth" with respect to the underlying similarity graph (*e.g.*, a bandlimited signal in graph frequencies). With this assumption, we can approach restoration tasks by designing an appropriate graph low-pass filter.

A.4 Graph Laplacian Regularizer (GLR)

The extent to which a signal $\mathbf{x} \in \mathbb{R}^N$ is smooth w.r.t. a graph $\mathcal{G}$ specified by Laplacian $\mathbf{L}$ can be quantified using the *graph Laplacian regularizer* (GLR) [17]:

$$\text{GLR}(\mathbf{x}) = \mathbf{x}^\top \mathbf{L}\mathbf{x} = \sum_{(i,j)\in\mathcal{E}} w_{i,j}(x_i - x_j)^2. \tag{3.6}$$

Thus, GLR can be used as a signal prior for graph signal restoration applications such as denoising [17], dequantization [19], and interpolation [41]. Specifically for interpolation, denote by $\mathbf{H} \in \{0,1\}^{M\times N}$ a sampling matrix that picks out M observable samples from N. The interpolation problem can be formulated as

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2^2 + \mu\, \mathbf{x}^\top \mathbf{L}\mathbf{x}, \tag{3.7}$$

where $\mu \in \mathbb{R}_+$ is a weight parameter. Given that $\mathbf{L}$ is PSD for a positive graph, the optimization in (3.7) is convex and quadratic, and solution $\mathbf{x}^*$ to (3.7) can be

computed by solving the following linear system:

$$(\mathbf{H}^\top \mathbf{H} + \mu \mathbf{L})\mathbf{x}^* = \mathbf{H}^\top \mathbf{y}. \tag{3.8}$$

If $\mathbf{L}$ is sparse, then (3.8) can be solved in linear time using fast numerical methods like *conjugate gradient* (CG) [35] without matrix inversion.

B Graph Convolutional Networks

Graph Convolutional Networks (GCNs) constitute a class of neural networks designed to operate on data represented by graphs [1, 97, 98]. As mentioned in Section A, a graph can be described by its adjacency matrix $\mathbf{A}$ and a node feature matrix (graph signals) $\mathbf{X}$.

B.1 GCN Definition

A fundamental operation in GCNs is to iteratively aggregate and transform node features based on a node's local neighborhood structure. Denote by $h_v^{(l)}$ the representation (or *embedding*) of node v at the l-th layer. The propagation rule for the $(l+1)$-th layer is often defined as:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{|\mathcal{N}(v)|, \cdot, |\mathcal{N}(u)|}}, \mathbf{A}_{uv}, h_u^{(l)}, \mathbf{W}^{(l)}\right), \tag{3.9}$$

where $\mathcal{N}(v)$ denotes the set of neighbors of v in the graph, $\mathbf{A}_{uv}$ is the entry of the adjacency matrix corresponding to the edge between u and v, and $\mathbf{W}^{(l)}$ is a learnable weight matrix at layer l. Finally, $\sigma(\cdot)$ is a pointwise nonlinearity that introduces non-linear modeling capacity, similar to activation functions used in classical neural

networks.

Conceptually, the summation in (3.9) collects features from neighboring nodes $u \in \mathcal{N}(v)$, reweights them by an inverse-degree factor $\frac{1}{\sqrt{|\mathcal{N}(v)|\cdot|\mathcal{N}(u)|}}$, and then transforms the result via $\mathbf{W}^{(l)}$ and $\sigma(\cdot)$. This mechanism resembles a *graph filtering* step (Section A.3), where neighborhood information is aggregated to update node embeddings.

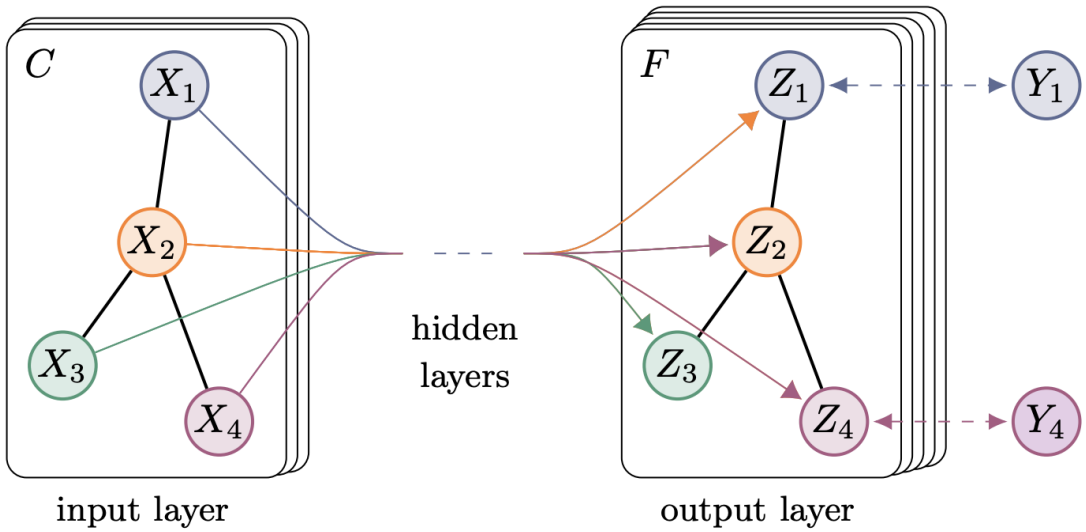

Figure 3.1: Illustration of a multi-layer graph convolutional network (GCN) for semi-supervised learning with C input channels and F output feature maps. The graph structure (edges) remains consistent across layers, and node labels are denoted by Y_i [1].

B.2 Limitations and Challenges

Despite the effectiveness of Graph Convolutional Networks (GCNs) in various graph-based tasks, such as node classification, link prediction, and recommendation systems, they encounter several significant limitations:

Shallow Architecture. Unlike deep neural networks used in fields such as image recognition or language modeling, GCNs typically employ fewer layers. While deeper networks can capture hierarchical representations by learning both local and global features, stacking multiple GCN layers often leads to numerical instabilities or oversmoothing of node features. This shallow architecture limits the ability of GCNs to model complex relationships in large or highly intricate graphs.

Oversmoothing. GCNs aggregate information iteratively from neighboring nodes, which can cause node features to become excessively similar as the number of layers increases. This phenomenon, known as *oversmoothing*, reduces the discriminative power of learned embeddings by homogenizing node representations across the graph. Consequently, subtle distinctions among nodes—critical for downstream tasks—may be lost, diminishing the overall effectiveness of the network.

Computational Complexity on Dense Graphs. Each convolutional layer in a GCN functions as a form of *graph filter* (see Section A.3). In dense or fully connected graphs, the computational cost of applying these filters escalates rapidly with the number of edges, rendering GCN training and inference computationally expensive. This challenge motivates approaches to *sparsify* graphs (*e.g.*, by removing less significant edges) to alleviate the computational burden. As detailed in Chapter 5, managing graph density is crucial for efficiently training large-scale GCNs on extensive datasets.

In summary, GCNs provide a robust framework for extending deep learning to the graph domain, leveraging local neighborhoods to capture graph structure. However,

addressing challenges such as shallow architectures, oversmoothing, and high computational costs in dense graphs has spurred research into improved graph filtering techniques, innovative architectures, and enhanced regularization strategies to realize the full potential of graph-based deep learning.

Chapter 4

Methodology: Modeling Slowly Time-varying Graphs

A Main Problem Formulation

This section formulates the joint graph learning/signal interpolation problem to parameterize a time-varying graph model, given partial observations of a signal at different time instants. The key assumption, as discussed in Chapter 1, is that the difference between two adjacency matrices $\mathbf{W}^{(t+1)} - \mathbf{W}^{(t)}$ in two consecutive instants, $t+1$ and t, is low-rank.

A.1 Time-varying Graph Signal Interpolation

Specifically, this section addresses a signal interpolation scenario in which the underlying graph $\mathcal{G}_1$, specified by a (typically sparse[2]) adjacency matrix $\mathbf{W}^{(1)}$ at the first time instant $t = 1$ is known. This is practical for both the scenario where *full* signal

[2]Graph models used in the GSP literature are often sparse for computational reasons [9, 10]. It is assumed that the initial $\mathbf{W}^{(1)}$ is sparse, and that subsequent constructed $\mathbf{W}^{(t)}$'s for $t > 1$ remain sparse through careful algorithm design.

observations from the past $t < 1$ are available, and the scenario where *partial* observations at $t > 1$ are available at $t = 1$, so that a single empirical covariance matrix can be estimated from all collected data, and in turn a sparse inverse covariance matrix (interpreted as graph Laplacian) can be computed using techniques such as GLASSO or CLIME [45, 99].

Then, in subsequent instants $t \in \{2, \ldots, T\}$, partial signal observations $\mathbf{y}_t \in \mathbb{R}^{M_t}$, $M_t < N$, become available[3]. The goal is to interpolate $\mathbf{y}_t$ to $\mathbf{x}_t \in \mathbb{R}^N$ for each instant t, assuming that $\mathbf{x}_t$ is smooth w.r.t. to graph $\mathcal{G}_t$, and that the adjacency matrices $\mathbf{W}^{(t)}$'s are slowly time-varying. Denote by $\mathbf{X} \in \mathbb{R}^{N \times T}$ a matrix variable that contains $\mathbf{x}_t$ as the t-th column. The problem can be interpreted as *matrix completion*: given sparse entries $\mathbf{y}_t$ in each column t, complete the remaining $N - M_t$ entries to restore the entire N-by-T data matrix $\mathbf{X}$. A matrix can be completed assuming a low-rank prior using methods such as *robust principal component analysis* (RPCA) [30]. However, by assuming that each signal $\mathbf{x}_t$ at time t is smooth w.r.t. underlying graph $\mathcal{G}_t$ that is slowly time-varying, it is possible to more specifically model the spatial similarities of each $\mathbf{x}_t$, leading to improved signal interpolation.

Concrete Example: One practical real-world example of this setting is *farmland valuation*: given that crop-growing fields are sold only in some counties of a farming state like Iowa in a given quarter, the problem is to estimate field prices in other counties with no recorded sales. Pairwise similarities between neighboring counties (with similar environmental variables such as rainfall and soil composition) can change slowly over time due to regional factors, such as migrating population, county-specific economic policies, and farming practices.

[3] $\mathbf{y}_t$ can also be viewed as a set of samples observed between discrete time instants $t - 1$ and t.

A.2 Slowly Time-Varying Graph Model

This subsection describes the slowly time-varying graph model. Specifically, adjacency matrix $\mathbf{W}^{(t+1)}$ of graph $\mathcal{G}_{t+1}$ at time $t+1$ is assumed to differ from adjacency matrix $\mathbf{W}^{(t)}$ of graph $\mathcal{G}_t$ at time t approximately by a small number K of rank-1 matrices, where $K \ll N$, *i.e.*,

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \sum_{k=1}^{K} \mathbf{u}_k^{(t)} (\mathbf{v}_k^{(t)})^\top \tag{4.1}$$

where $\mathbf{u}_k^{(t)}, \mathbf{v}_k^{(t)} \in \mathbb{R}^N$ specify a rank-1 update matrix. In essence, (4.1) states that the difference matrix $\mathbf{W}^{(t+1)} - \mathbf{W}^{(t)}$ is a low-rank matrix. This model is applicable to both weighted graphs, where $\mathbf{W}^{(t)}$ contains real-valued edge weights, and unweighted graphs, where $\mathbf{W}^{(t)}$ contains binary entries (0 or 1), as the rank-1 updates can represent changes in edge weights or connectivity.

Concrete Example: Continuing the farmland valuation example, consider the case where a county i's average field price increases at time $t+1$ thanks to a new municipal tax exemption. Consequently, county i's pairwise similarities with *all* other counties change—-a *rank*-2 *update* in $\mathbf{W}^{(t+1)}$ due to changes in row i and column i. Note that this rank-2 update may not be small in Frobenius norm $\|\mathbf{W}^{(t+1)} - \mathbf{W}^{(t)}\|_F^2$ [32], or small in ℓ_0-norm $\|\mathbf{W}^{(t+1)} - \mathbf{W}^{(t)}\|_0$ [31]), as previous models assumed. This low-rank assumption (4.1) is empirically validated for small time intervals $[t, t+1]$ later using real-world datasets. Note that small time intervals can vary across different datasets and applications, which will be discussed in more detail in Section E.3.b.

Given this time-varying graph model (4.1), signal $\mathbf{x}_t$ can be generated at time t

given adjacency matrix $\mathbf{W}^{(t)}$ and signal $\mathbf{x}_{t-1}$ at time $t-1$ as

$$\Pr(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mathbf{0}, (\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1})\,\mathcal{N}(\mathbf{x}_t|\mathbf{x}_{t-1}, \beta\mathbf{I}), \tag{4.2}$$

where $\alpha > 0$ is a small parameter, given that combinatorial graph Laplacian $\mathbf{L}^{(t)}$ has rank $N-1$ (assuming a connected graph) and thus is not invertible. (4.2) states that probability $\Pr(\mathbf{x}_t|\mathbf{x}_{t-1})$ is proportional to *both* the Gaussian-distributed probability of $\mathbf{x}_t$ with zero mean and covariance $(\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1}$ *and* the iid Gaussian-distributed probability of $\mathbf{x}_t$ with mean at $\mathbf{x}_{t-1}$ and variance β.

A.3 General Problem Formulation

Consider the isolated case of reconstructing signal $\mathbf{x}_2 \in \mathbb{R}^N$ at time $t = 2$, given signal $\mathbf{x}_1 \in \mathbb{R}^N$ and adjacency matrix $\mathbf{W}^{(1)} \in \mathbb{R}^{N\times N}$ at time $t = 1$, as well as partial observation $\mathbf{y}_2 \in \mathbb{R}^{M_2}$ at time $t = 2$, where $M_2 < N$. Let $\mathbf{Q} \in \mathbb{R}^{M_2\times M_2}$ be a diagonal matrix where $Q_{i,i}$ specifies the reliability of observation y_i (to be discussed in detail later in Section A.4). Denote by $\mathbf{H}^{(2)} \in \{0,1\}^{M_2\times N}$ a sampling matrix that picks out M_2 observed entries from $\mathbf{x}_2$, *i.e.*,

$$H_{i,j} = \begin{cases} 1 & \text{if the } i\text{-th sample is index } j \\ 0 & \text{o.w.} \end{cases}. \tag{4.3}$$

A joint optimization is then formulated for adjacency matrix $\mathbf{W}^{(2)}$ *and* signal $\mathbf{x}_2$. Following the signal model (4.2), for $\mathbf{x}_2$, GLR [4] ($\mathbf{x}_2^\top \mathbf{L}^{(2)}\mathbf{x}_2$) [17] and ℓ_2-norm ($\|\mathbf{x}_2 - \mathbf{x}_1\|_2^2$)

[4] The standard form $\mathbf{x}^\top\mathbf{L}\mathbf{x}$ is employed here as prior for signal $\mathbf{x}$ due to its convenient ℓ_2-norm that is amenable to fast optimization. Other graph signal priors such as graph total variation (GTV) [24], could also be used.

are included to promote both graph smoothness and similarity to $\mathbf{x}_1$. Given time-varying graph model (4.1), for $\mathbf{W}^{(2)}$ a low-rank prior $\Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)})$ is introduced to promote similarities between $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$. This results in the following optimization formulation:

$$\begin{aligned}\min_{\mathbf{W}^{(2)},\mathbf{x}_2} \; & (\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2)^\top \mathbf{Q}(\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2) + \mu\, \mathbf{x}_2^\top \mathbf{L}^{(2)}\mathbf{x}_2 \\ & + \xi\|\mathbf{x}_2 - \mathbf{x}_1\|_2^2 + \eta\, \Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)}), \\ \text{s.t.} \;\; & \mathbf{L}^{(2)} = \text{diag}(\mathbf{W}^{(2)}\mathbf{1}) - \mathbf{W}^{(2)} \\ & W_{i,i}^{(2)} = 0, \; \forall i, \quad W_{j,i}^{(2)} = W_{i,j}^{(2)} \geq 0, \; \forall i \neq j \end{aligned} \tag{4.4}$$

where $\mu, \xi, \eta \in \mathbb{R}_+$ are weight parameters, and $\Gamma(\mathbf{M})$ is the rank of matrix $\mathbf{M}$. The first constraint defines a combinatorial Laplacian matrix $\mathbf{L}^{(2)}$ from matrix variable $\mathbf{W}^{(2)}$. The last two constraints ensure that $\mathbf{W}^{(2)}$ is a symmetric adjacency matrix for a positive undirected graph $\mathcal{G}_2$ without self-loops.

A.4 Generalized Variational Term

Denote by $\{X_i\}$ a sequence of Gaussian *independent and identical distributed* (iid) random variables with mean μ and variance σ^2. Define $\bar{X}_N \triangleq \frac{1}{N}\sum_{i=1}^N X_i$ as the empirical sample mean over N observations. By the *weak law of large numbers* (WLLN) [100], the mean and variance of $\bar{X}_N$ are

$$E[\bar{X}_N] = \mu, \quad \text{Var}(\bar{X}_N) = \frac{\sigma^2}{N}. \tag{4.5}$$

The likelihood term for $\bar{X}_N$ can be written as

$$\Pr(\bar{X}_N = y) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{N}}} \exp\left(\frac{(y-\mu)^2}{2\frac{\sigma^2}{N}}\right) \tag{4.6}$$

Looking at the exponent, it implies that negative log likelihood term should account for the observation number M *linearly*; *i.e.*, $Q_{i,i} = N$.

In practice, the reliability of observations $y_{t,i}$'s varies (*e.g.*, each observation $y_{t,i}$ is an average per acre land sale value in a county over a variable number of transactions in a fiscal quarter). To account for this variability in signal reconstruction of $\mathbf{x}_t$, the following procedure is performed. Define $\mathbf{Q} \in \mathbb{R}^{M\times M}$ as a diagonal matrix, where $Q_{i,i} > 0$ is the weight associated with observation $y_{t,i}$. For example, if $y_{t,i}$ is an estimate of land prices in a county based on sales in a quarter, then $Q_{i,i}$ is chosen based on the number of sales (more sales means more stable price estimate).

B Algorithm Development: Version 1 (Conference)

B.1 Problem Formulation

In the first version of the algorithm, a simplified version of (4.4) is used without a ℓ_2-norm ($\|\mathbf{x}_2 - \mathbf{x}_1\|_2^2$) and $\mathbf{Q}$ matrix:

$$\begin{aligned} \min_{\mathbf{W}^{(2)},\mathbf{x}_2} \; & \|\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2\|_2^2 + \mu\, \mathbf{x}_2^\top \mathbf{L}^{(2)}\mathbf{x}_2 + \eta\, \|\mathbf{W}^{(2)} - \mathbf{W}^{(1)}\|_* \\ \text{s.t.} \; & \mathbf{L}^{(2)} = \operatorname{diag}(\mathbf{W}^{(2)}\mathbf{1}) - \mathbf{W}^{(2)} \\ & W^{(2)}_{i,i} = 0, \; \forall i, \qquad W^{(2)}_{j,i} = W^{(2)}_{i,j} \geq 0, \; \forall i \neq j \end{aligned} \tag{4.7}$$

where $\mu, \eta \in \mathbb{R}_+$ are weight parameters, and $\|\cdot\|_*$ is the *nuclear norm*—the sum of the matrix argument's singular values—that is the convexification of the non-convex matrix rank. The first constraint defines a combinatorial Laplacian matrix $\mathbf{L}^{(2)}$ from variable $\mathbf{W}^{(2)}$. The second and third constraints ensure that $\mathbf{W}^{(2)}$ is a valid adjacency matrix for a positive undirected graph without self-loops.

B.2 Optimization

(4.7) can be solved for variables $\mathbf{W}^{(2)}$ and $\mathbf{x}_2$ alternately. When $\mathbf{W}^{(2)}$ is fixed, (4.7) becomes an unconstrained quadratic programming (QP) problem with a system of linear equations (3.8) as the solution. It can be solved for optimal $\mathbf{x}_2^*$ efficiently via *conjugate gradient* (CG) [35].

When $\mathbf{x}_2$ is fixed, the optimization can be written as follows. An *indicator function* $\mathrm{I}_W(\mathbf{W})$ is defined for matrix $\mathbf{W}$ as

$$\mathrm{I}_W(\mathbf{W}) = \begin{cases} 0, & \text{if } \ W_{i,i} = 0, \forall i, \ W_{j,i} = W_{i,j} \geq 0, \forall i \neq j \\ \infty, & \text{o.w.} \end{cases} \tag{4.8}$$

Importantly, note that the set of matrices $\{\mathbf{W}\}$ such that $\mathrm{I}_W(\mathbf{W}) = 0$ is a *convex set*, which we denote as $\mathcal{C}$.

Then an unconstrained optimization can be defined as

$$\begin{aligned} \min_{\mathbf{W}^{(2)}} \ & \mu \operatorname{Tr}((\operatorname{diag}(\mathbf{W}^{(2)}\mathbf{1}) - \mathbf{W}^{(2)})\mathbf{x}_2\mathbf{x}_2^\top) \\ & + \eta \|\mathbf{W}^{(2)} - \mathbf{W}^{(1)}\|_* + \mathrm{I}_W(\mathbf{W}^{(2)}). \end{aligned} \tag{4.9}$$

The objective in (4.9) comprises three terms: the first is convex and differentiable

(smooth), and the second and third are convex and non-smooth. (4.9) can be minimized via a new variant[5] of *proximal gradient descent* (PGD) [101] as follows.

The first term can be expanded as

$$\mu \operatorname{Tr}(\operatorname{diag}(\mathbf{W}^{(2)}\mathbf{1})\mathbf{x}_2\mathbf{x}_2^\top) - \mu \operatorname{Tr}(\mathbf{W}^{(2)}\mathbf{x}_2\mathbf{x}_2^\top). \tag{4.10}$$

Given that $\operatorname{diag}(\mathbf{a}) = (\mathbf{a}\mathbf{1}^\top) \odot \mathbf{I}$ where $\mathbf{a} \in \mathbb{R}^N$, we can rewrite the first part as $\mu \operatorname{Tr}(((\mathbf{W}^{(2)}\mathbf{1}\mathbf{1}^\top) \odot \mathbf{I})\mathbf{x}_2\mathbf{x}_2^\top)$. Thus, the gradient $\nabla_{\mathbf{W}}$ of (4.10) w.r.t. $\mathbf{W}^{(2)}$ is

$$\nabla_{\mathbf{W}} = \mu\, \mathbf{x}_2^2\mathbf{1}^\top - \mu\, \mathbf{x}_2\mathbf{x}_2^\top. \tag{4.11}$$

Second, the proximal mapping for the nuclear norm $g(\mathbf{M}) = \eta \|\mathbf{M} - \mathbf{W}^{(1)}\|_*$ can be derived as follows. Denote by $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{N\times N}$ matrices containing the left and right singular vectors of $\mathbf{W}^{(1)}$ as columns, *i.e.*, $\mathbf{W}^{(1)} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{\Sigma} \in \mathbb{R}^{N\times N}$ is a diagonal matrix with $\mathbf{W}^{(1)}$'s singular values along its diagonal. By definition, the proximal mapping for $g(\mathbf{M})$ is

$$\operatorname{prox}_g(\mathbf{M}) = \arg\min_{\mathbf{Z}} \frac{1}{2}\|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta \|\mathbf{Z} - \mathbf{W}^{(1)}\|_* \,. \tag{4.12}$$

In words, (4.12) states that the variable $\mathbf{Z}$ should be close to the input $\mathbf{M}$ in Frobenius norm *and* close to $\mathbf{W}^{(1)}$ in nuclear norm.

(4.12) is solved approximately as follows. First, it is assumed initially that $\mathbf{M}$ and $\mathbf{W}^{(1)}$ share the same left and right singular vectors $\mathbf{U}$ and $\mathbf{V}$. This is reasonable, since $\mathbf{M}$ is a candidate solution for $\mathbf{W}^{(2)}$, and $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$ are similar by our low-rank

[5]Because the nuclear norm is computed on the *difference* between adjacency matrices $\mathbf{W}^{(2)} - \mathbf{W}^{(1)}$, not the adjacency matrix $\mathbf{W}^{(2)}$ directly, simple soft-thresholding can not be performed as done in [101].

assumption in (4.1). Thus, the optimal $\mathbf{Z}$ also shares the singular vectors $\mathbf{U}$ and $\mathbf{V}$. The optimal singular value s_i^* corresponding to each rank-1 matrix component $\mathbf{u}_i\mathbf{v}_i^\top$ is computed as

$$s_i^* = \arg\min_{s_i} \frac{1}{2}(s_i - a_i)^2 + \eta|s_i - b_i| \tag{4.13}$$

where a_i and b_i are the i-th singular value for $\mathbf{M}$ and $\mathbf{W}^{(1)}$, respectively. In practice, singular value decomposition (SVD) is computed on $\mathbf{W}^{(1)}$ to obtain $\mathbf{U}$, $\mathbf{V}$ and $\{b_i\}$. Then, $\{a_i\}$ are computed by projecting $\mathbf{M}$ onto rank-1 matrices $\mathbf{u}_i\mathbf{v}_i^\top$ that are the outer products of $\mathbf{W}^{(1)}$'s computed singular vectors. The solution to (4.12) is $\mathbf{Z}^* = \mathbf{USV}^\top$, where $\mathbf{S}$ is diagonal with $S_{i,i} = s_i^*$.

Later in the PGD iterations, when the magnitude of the gradient $\nabla_{\mathbf{W}}$ is less than a threshold γ, the singular vectors $\mathbf{U}^*$ and $\mathbf{V}^*$ of the average matrix $\mathbf{W}^* \triangleq (\mathbf{M}+\mathbf{W}^{(1)})/2$ are computed instead, and $\{b_i\}$ and $\{a_i\}$ are computed by projecting $\mathbf{M}$ and $\mathbf{W}^{(1)}$ onto rank-1 matrices $\mathbf{u}_i^*(\mathbf{v}_i^*)^\top$, respectively. Small gradient magnitude $|\nabla_{\mathbf{W}}|$ means that the solution has stabilized, and computing SVD on $\mathbf{W}^*$ means that the singular structure of the solution can evolve over time.

Third, one can easily show that the proximal mapping for the indicator function $\mathrm{I}_W(\mathbf{W}^{(2)})$ is a convex-set projection operator [101]:

$$\mathrm{prox}_{I_w,t}(\mathbf{M}) = \mathrm{proj}_{\mathcal{C}}(\mathbf{M}) \tag{4.14}$$

$$(\mathrm{proj}_{\mathcal{C}}(\mathbf{M}))_{i,j} = \begin{cases} 0 & \text{if } i = j \\ 0 & \text{if } i \neq j, M_{i,j} < 0 \\ \frac{M_{j,i}+M_{i,j}}{2} & \text{if } i \neq j, M_{j,i}, M_{i,j} \geq 0 \end{cases} \tag{4.15}$$

Finally, a PGD algorithm can be devised as:

$$\mathbf{W}^{(2)} \leftarrow \text{proj}_{\mathcal{C}}(\text{prox}_{g}(\mathbf{W}^{(2)} - \alpha\nabla_{\mathbf{W}})). \tag{4.16}$$

(4.16) works as follows. First, the first term in (4.9) is decreased by modifying $\mathbf{W}^{(2)}$ in the opposite gradient direction $-\nabla_{\mathbf{W}}$, where α is a step size. Then proximal mapping (4.12) is performed. Finally, convex-set projection (4.40) is executed. (4.16) is executed iteratively until solution $\mathbf{W}^{(2)}$ converges. Signal $\mathbf{x}_2$ and adjacency matrix $\mathbf{W}^{(2)}$ are alternately optimized until the pair converges.

To interpolate $\mathbf{y}_t \in \mathbb{R}^{M_t}$ to $\mathbf{x}_t \in \mathbb{R}^N$ for each $t \in \{2, \ldots, T\}$, each $\mathbf{x}_t$ is computed iteratively, *i.e.*, first $\mathbf{x}_2$ and $\mathbf{W}^{(2)}$ are computed using previous $\mathbf{W}^{(1)}$, then $\mathbf{x}_3$ and $\mathbf{W}^{(3)}$ using previous $\mathbf{W}^{(2)}$, etc.

C Algorithm Development: Version 2 (Journal)

This version of the algorithm employs more advanced optimization techniques to address the objective (4.4). Here, (4.4) can be solved again via an alternating approach: solve for one variable while holding the other fixed, and vice versa, until solution convergence. When $\mathbf{W}^{(2)}$ is fixed, (4.4) becomes an unconstrained convex *quadratic programming* (QP) problem, whose solution can be obtained by solving a system of linear equations:

$$\left((\mathbf{H}^{(2)})^{\top}\mathbf{Q}\mathbf{H}^{(2)} + \mu\mathbf{L}^{(2)} + \xi\mathbf{I}\right)\mathbf{x}_2^* = (\mathbf{H}^{(2)})^{\top}\mathbf{Q}\mathbf{y}_2 + \xi\mathbf{x}_1. \tag{4.17}$$

Assuming that $\mathbf{L}^{(2)}$ is sparse (graph model $\mathcal{G}^{(t)}$ is assumed sparse), the coefficient matrix $(\mathbf{H}^{(2)})^{\top}\mathbf{Q}\mathbf{H}^{(2)} + \mu\mathbf{L}^{(2)} + \xi\mathbf{I}$ is symmetric, PD and sparse, and optimal $\mathbf{x}_2^*$ can

be computed via CG in linear time.

When $\mathbf{x}_2$ is fixed, the optimization (4.4) can be written as follows. An *indicator function* $\mathrm{I}_W(\mathbf{W})$ is first defined for matrix $\mathbf{W}$ as

$$\mathrm{I}_W(\mathbf{W}) = \begin{cases} 0, & \text{if } W_{i,i} = 0, \forall i,\ W_{j,i} = W_{i,j} \geq 0, \forall i \neq j \\ \infty, & \text{o.w.} \end{cases} . \tag{4.18}$$

Denote by $\mathcal{C}$ the set of matrices such that $\mathrm{I}_W(\mathbf{W}) = 0$, in other words, $\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{N\times N} \,|\, \mathrm{I}_W(\mathbf{W}) = 0\}$. It can be proved that $\mathcal{C}$ is convex.

Lemma 1. *$\mathcal{C}$ is a convex set.*

Proof. Denote by $\mathbf{W}_1, \mathbf{W}_2 \in \mathcal{C}$ two matrices in set $\mathcal{C}$. Consider $\mathbf{W} = \alpha\mathbf{W}_1 + (1-\alpha)\mathbf{W}_2$ for $0 \leq \alpha \leq 1$, a convex combination of $\mathbf{W}_1$ and $\mathbf{W}_2$. Diagonal entry i of $\mathbf{W}$ is

$$\begin{aligned} W_{i,i} &= \alpha W_{1,i,i} + (1-\alpha) W_{2,i,i} \\ &= \alpha 0 + (1-\alpha)0 = 0. \end{aligned} \tag{4.19}$$

On the other hand, off-diagonal entry $W_{i,j}$ is

$$W_{i,j} = \alpha W_{1,i,j} + (1-\alpha) W_{2,i,j} \geq 0 \tag{4.20}$$

$$\begin{aligned} W_{j,i} &= \alpha W_{1,j,i} + (1-\alpha) W_{2,j,i} \\ &= \alpha W_{1,i,j} + (1-\alpha) W_{2,i,j} = W_{i,j}. \end{aligned} \tag{4.21}$$

Thus, $\mathrm{I}_W(\mathbf{W}) = 0$, and $\mathcal{C}$ is a convex set. □

The implication of Lemma 1 is that $\mathrm{I}_W(\mathbf{W})$ is convex, albeit non-smooth. To

rewrite the objective in (4.4), GLR is first rewritten as

$$\mathbf{x}_2^\top \mathbf{L}^{(2)} \mathbf{x}_2 = \mathrm{Tr}(\mathbf{x}_2^\top \mathbf{L}^{(2)} \mathbf{x}_2) = \mathrm{Tr}(\mathbf{L}^{(2)} \mathbf{x}_2 \mathbf{x}_2^\top). \tag{4.22}$$

Expanding out the definition of $\mathbf{L}^{(2)}$ as variable $\mathbf{W}^{(2)}$, an unconstrained optimization for $\mathbf{W}^{(2)}$ can be defined as

$$\begin{aligned} \min_{\mathbf{W}^{(2)}} \quad & \mu \,\mathrm{Tr}((\mathrm{diag}(\mathbf{W}^{(2)}\mathbf{1}) - \mathbf{W}^{(2)})\mathbf{x}_2\mathbf{x}_2^\top) \\ & + \eta\, \Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)}) + \mathrm{I}_W(\mathbf{W}^{(2)}). \end{aligned} \tag{4.23}$$

The objective in (4.23) is composed of three terms: the first is convex and differentiable (smooth), the second is combinatorial, and the third is convex but non-smooth.

C.1 Proximal Gradient Descent

The three terms in (4.23) are minimized one at a time via a new variant[6] of *proximal gradient descent* (PGD) [101] as follows.

C.1.a Gradient Descent

The first term can be expanded as

$$\mu \,\mathrm{Tr}(\mathrm{diag}(\mathbf{W}^{(2)}\mathbf{1})(\mathbf{x}_2)(\mathbf{x}_2)^\top) - \mu \,\mathrm{Tr}(\mathbf{W}^{(2)}(\mathbf{x}_2)(\mathbf{x}_2)^\top). \tag{4.24}$$

[6]Because matrix rank $\Gamma(\cdot)$ is evaluated on the *difference* of matrices $\mathbf{W}^{(2)} - \mathbf{W}^{(1)}$, not the matrix variable $\mathbf{W}^{(2)}$, the proximal mapping for $\mathbf{W}^{(2)} - \mathbf{W}^{(1)}$ is unique. As a result, it cannot simply be convexified into a nuclear norm for soft-thresholding operations, as done in [101].

Given that $\text{diag}(\mathbf{a}) = (\mathbf{a}\mathbf{1}^\top) \odot \mathbf{I}$ where $\mathbf{a} \in \mathbb{R}^N$, the first part can be revised as

$$\mu \,\text{Tr}(((\mathbf{W}^{(2)}\mathbf{1}\mathbf{1}^\top) \odot \mathbf{I})\mathbf{x}_2\mathbf{x}_2^\top) \tag{4.25}$$

Thus, the gradient $\nabla_{\mathbf{W}^{(2)}}$ of (4.24) w.r.t. $\mathbf{W}^{(2)}$ is

$$\nabla_{\mathbf{W}^{(2)}} = \mu\,(\mathbf{x}_2)^2\mathbf{1}^\top - \mu\,(\mathbf{x}_2)(\mathbf{x}_2)^\top. \tag{4.26}$$

Thus, to reduce the first term, $\mathbf{W}^{(2)} - \epsilon\nabla_{\mathbf{W}^{(2)}}$ is computed, where $\epsilon > 0$ is a step size.

While the gradient $\nabla_{\mathbf{W}^{(2)}}$ for the first term in (4.23) is easily computable, reducing the second rank term via proximal mapping is more challenging.

C.1.b Proximal Mapping for Matrix Rank Term

By definition, the proximal mapping [101] for the weighted rank term $\eta\,\Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)})$ is

$$\text{prox}_{\mathbf{W}^{(1)}}(\mathbf{M}) = \arg\min_{\mathbf{Z}} \frac{1}{2}\|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta\,\Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) \tag{4.27}$$

where $\mathbf{M}$ is a candidate solution for $\mathbf{W}^{(2)}$. In words, (4.27) states that variable $\mathbf{Z}$ should be close to $\mathbf{M}$ in Frobenius norm *and* deviate from $\mathbf{W}^{(1)}$ by only a low-rank matrix.

The proximal mapping (4.27) is itself an optimization, and it can be efficiently approximated as follows[7]. Note that $\mathbf{W}^{(1)}$ is an adjacency matrix for an undirected

[7]In the conference version [34], the rank term $\Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)})$ is first convexified to the nuclear norm—the sum of singular values—with complexity $\mathcal{O}(N^3)$. In the recent version, a fast greedy algorithm is used to address the rank term directly with complexity $\mathcal{O}(N)$.

graph, and hence $\mathbf{W}^{(1)}$ is real and symmetric, thus eigen-decomposable by the Spectral Theorem [102]. Denote by $\mathbf{V} \in \mathbb{R}^{N\times N}$ a matrix containing the orthonormal eigenvectors of $\mathbf{W}^{(1)}$ as columns, *i.e.*, $\mathbf{W}^{(1)} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$, where $\mathbf{\Lambda} \in \mathbb{R}^{N\times N}$ is a diagonal matrix $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$, and $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_N|$.

C.1.c Full Greedy Algorithm

In this section, an intuitive greedy algorithm is developed for (4.27), albeit with high complexity; a low-complexity variant is presented in the sequel. Note that $\mathbf{W}^{(1)} = \sum_{i=1}^N \lambda_i \mathbf{v}_i \mathbf{v}_i^\top$ can be written as a linear combination of rank-1 matrices (outer-product of orthonormal eigenvectors $\mathbf{v}_i$'s), each scaled by eigenvalue λ_i. To achieve minimal matrix rank $\Gamma(\mathbf{Z} - \mathbf{W}^{(1)})$, $\mathbf{Z} \leftarrow \mathbf{W}^{(1)}$ is first initialized, which means

$$\Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) = \Gamma(\mathbf{W}^{(1)} - \mathbf{W}^{(1)}) = 0. \tag{4.28}$$

Next, to maximally reduce Frobenius norm $\|\mathbf{Z}-\mathbf{M}\|_F^2$ using a single rank-1 matrix $\mathbf{v}_i\mathbf{v}_i^\top$, $\mathbf{Z} - \mathbf{M}$ is projected onto the set of N rank-1 matrices $\{\mathbf{v}_i\mathbf{v}_i^\top\}_{i=1}^N$ via matrix inner-product $\alpha_i = \langle \mathbf{Z} - \mathbf{M}, \mathbf{v}_i\mathbf{v}_i^\top \rangle = \text{Tr}((\mathbf{Z} - \mathbf{M})\mathbf{v}_i\mathbf{v}_i^\top), \forall i$. The largest magnitude $|\alpha_i|$ among N inner-products is the best rank-1 approximation $\alpha_i\mathbf{v}_i\mathbf{v}_i^\top$ of error $\mathbf{Z}-\mathbf{M}$ given basis $\{\mathbf{v}_i\mathbf{v}_i^\top\}$. In other words, it is the maximal reduction in Frobenius norm $\|\mathbf{Z} - \mathbf{M}\|_F^2$ given the cost of increasing rank $\Gamma(\mathbf{Z} - \mathbf{W}^{(1)})$ by one.

Mathematically, it means changing scalar s_i to $s_i \leftarrow \lambda_i - \alpha_i$ in $\mathbf{Z} = \sum_{i=1}^N s_i\mathbf{v}_i\mathbf{v}_i^\top$, resulting in $\Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) = 1$. The algorithm then checks if objective (4.27) has decreased, and if so, it continues with the next largest inner product magnitude $|\alpha_i|$ to update s_i. The algorithm continues until the objective (4.27) increases, at which point it retains the previous $\mathbf{Z}$. See Fig. 4.1 for a flow chart of the algorithm.

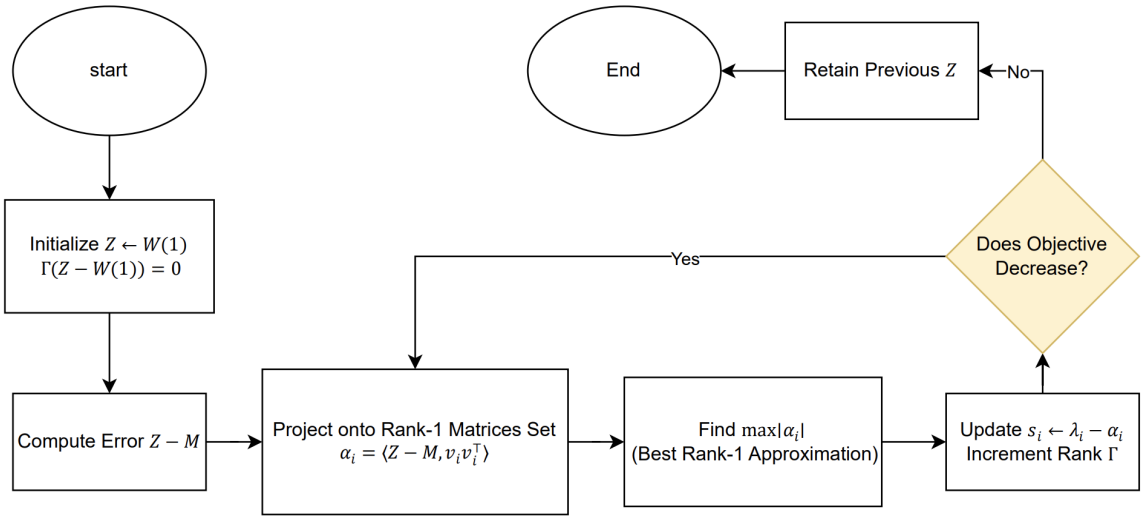

Figure 4.1: Flow chart for the full greedy algorithm.

The full greedy algorithm is globally optimal under the following assumption:

Assumption: The eigenvector set $\{\mathbf{v}_i\}$ of $\mathbf{W}^{(1)}$ includes all eigenvectors corresponding to non-zero eigenvalues of $\mathbf{W}^{(1)} - \mathbf{M}$ as a subset.

In practical terms, the assumption states that the slowly time-varying changes in consecutive adjacency matrices are well described by scaled outer-products of eigenvectors of $\mathbf{W}^{(1)}$. The justification of this assumption is discussed in detail in Section C.1.f. It can be argued that the assumption is approximately true in practice. $\mathbf{M}$ is a candidate solution for $\mathbf{W}^{(2)}$, and because of the slowly time-varying assumption for consecutive adjacency matrices, $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$ ought to be similar.

To show the full greedy algorithm's optimality, the following constrained variant of optimization (4.27) is first considered:

$$\min_{\mathbf{Z}} \|\mathbf{Z} - \mathbf{M}\|_F^2, \quad \text{s.t.} \quad \Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) \leq r. \tag{4.29}$$

The two optimizations (4.27) and (4.29) are related by the following two lemmas.

Lemma 2. *Denote by* $\mathbf{Z}^*$ *an optimal solution to* (4.27) *for a given* η^**, such that* $\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) = r^*$*.* $\mathbf{Z}^*$ *is also an optimal solution to* (4.29) *for* $r = r^*$*.*

Proof. By definition of optimality, $\mathbf{Z}^*$ satisfies

$$\begin{aligned} \frac{1}{2}\|\mathbf{Z}^* - \mathbf{M}\|_F^2 + \eta^*\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) \leq \\ \frac{1}{2}\|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta^*\Gamma(\mathbf{Z} - \mathbf{W}^{(1)}), \quad \forall \mathbf{Z} \end{aligned} \tag{4.30}$$

Define $\mathcal{S} \triangleq \{\mathbf{Z} \,|\, \Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) \leq r^*\}$. Then,

$$\begin{aligned} \frac{1}{2}\|\mathbf{Z}^* - \mathbf{M}\|_F^2 + \eta^*\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) \leq \\ \frac{1}{2}\|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta^*\Gamma(\mathbf{Z} - \mathbf{W}^{(1)}), \quad \forall \mathbf{Z} \in \mathcal{S} \end{aligned} \tag{4.31}$$

$$\begin{aligned} \frac{1}{2}\|\mathbf{Z}^* - \mathbf{M}\|_F^2 + \eta^* \left(r^* - \Gamma(\mathbf{Z} - \mathbf{W}^{(1)})\right) \leq \\ \frac{1}{2}\|\mathbf{Z} - \mathbf{M}\|_F^2, \quad \forall \mathbf{Z} \in \mathcal{S} \end{aligned} \tag{4.32}$$

$$\|\mathbf{Z}^* - \mathbf{M}\|_F^2 \overset{(a)}{\leq} \|\mathbf{Z} - \mathbf{M}\|_F^2, \quad \forall \mathbf{Z} \in \mathcal{S} \tag{4.33}$$

where (a) is true since $\eta^* > 0$, and $r^* \geq \Gamma(\mathbf{Z} - \mathbf{W}^{(1)})$ for $\mathbf{Z} \in \mathcal{S}$. □

An optimal solution to either (4.27) or (4.29) may not be unique. Suppose an optimal solution $\mathbf{Z}^o$ to (4.29) is computed for $r = r^*$, which may be different from a solution $\mathbf{Z}^*$ of (4.27). The following lemma relates $\mathbf{Z}^o$ to (4.27).

Lemma 3. *Denote by* $\mathbf{Z}^*$ *an optimal solution to* (4.27) *for a given* η^**, such that* $\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) = r^*$*. Denote by* $\mathbf{Z}^o$ *an optimal solution to* (4.29) *for* $r = r^*$*.* $\mathbf{Z}^o$ *is also an optimal solution to* (4.27) *for the same* η^**.*

Proof. By Lemma 2, $\mathbf{Z}^*$ is also an optimal solution to (4.29) for $r = r^*$, and thus by optimality to (4.29), $\|\mathbf{Z}^* - \mathbf{M}\|_F^2 = \|\mathbf{Z}^o - \mathbf{M}\|_F^2$. Further, $\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) = r^* \geq \Gamma(\mathbf{Z}^o - \mathbf{W}^{(1)})$. Thus,

$$\begin{aligned} &\|\mathbf{Z}^o - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z}^o - \mathbf{W}^{(1)}) \leq \\ &\|\mathbf{Z}^* - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}), \end{aligned} \tag{4.34}$$

given $\eta^* > 0$.

By optimality of $\mathbf{Z}^*$ to (4.27),

$$\begin{aligned} &\|\mathbf{Z}^* - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) \leq \\ &|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z} - \mathbf{W}^{(1)}), \quad \forall \mathbf{Z}. \end{aligned} \tag{4.35}$$

Thus, it can be concluded

$$\begin{aligned} &\|\mathbf{Z}^o - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z}^o - \mathbf{W}^{(1)}) \leq \\ &|\mathbf{Z} - \mathbf{M}\|_F^2 + \eta^* \Gamma(\mathbf{Z} - \mathbf{W}^{(1)}), \quad \forall \mathbf{Z}, \end{aligned} \tag{4.36}$$

and $\mathbf{Z}^o$ is an optimal solution to (4.27). □

A crucial corollary of Lemma 2 and 3 is stated as follows.

Corollary 1. *The solution set to* (4.29) *for all possible r's is a superset that contains a set of solutions to* (4.27) *for all possible η's for* $\eta > 0$.

Proof. By Lemma 2, each solution $(\eta^*, \mathbf{Z}^*)$ with $\Gamma(\mathbf{Z}^* - \mathbf{W}^{(1)}) = r^*$, of the set of solutions $\{\mathbf{Z}^*\}$ to (4.27) for all possible $\eta > 0$, is an optimal solution to (4.29) for

$r = r^*$. By solving (4.29) instead for solution $\mathbf{Z}^o$ (which may not be the same as $\mathbf{Z}^*$), $\mathbf{Z}^o$ is also a solution to (4.27) at η^*. Thus, solving (4.29) for all possible r's creates a set of solutions to (4.27) for all possible η's for $\eta > 0$. □

This corollary is useful in this algorithm design because while the set of possible $\eta > 0$ is infinite, the set of possible $r = \Gamma(\mathbf{Z} - \mathbf{W}^{(1)}) \in \{0, 1, \ldots, N\}$ is countably finite. Specifically, starting at $r = 0$, solutions to (4.29) can be generated for increasing r and test them against objective in (4.27) for a given η for an optimal solution.

It can be proven that the full greedy algorithm computes an optimal solution to (4.27).

Theorem 1. *The full greedy algorithm computes an optimal solution to* (4.27)*, assuming that the eigenvector set of* $\mathbf{W}^{(1)}$ *contains all eigenvectors corresponding to non-zero eigenvalues of* $\mathbf{W}^{(1)} - \mathbf{M}$.

Proof. It can be shown that the full greedy algorithm generates optimal solutions to (4.29) for all possible r's.

Consider first the base case for $r = 0$. Feasible solution set $\mathcal{S}$ for (4.29) contains a single solution $\mathbf{Z}^* = \mathbf{W}^{(1)}$. The full greedy algorithm generates this solution at initialization.

Consider the general case for $r \geq 1$, meaning that a feasible solution $\mathbf{Z}$ can deviate from $\mathbf{W}^{(1)}$ by at most a rank-r matrix $\mathbf{R}$, *i.e.*, $\mathbf{Z} = \mathbf{W}^{(1)} - \mathbf{R}$. Denote by r^o the rank of $\mathbf{W}^{(1)} - \mathbf{M}$. To minimize $\|\mathbf{Z} - \mathbf{M}\|_F^2 = \|\mathbf{W}^{(1)} - \mathbf{R} - \mathbf{M}\|_F^2$ using at most a rank-r matrix $\mathbf{R}$, the optimal solution $\mathbf{R}^*$ is the best rank-r^* approximation of $\mathbf{W}^{(1)} - \mathbf{M}$, where $r^* = \min(r, r^o)$, *i.e.*, $\mathbf{R}^* = \sum_{i=1}^{r^*} \alpha_i \mathbf{u}_i \mathbf{u}_i^\top$, α_i is the i-th largest magnitude eigenvalue of $\mathbf{W}^{(i)} - \mathbf{M}$, and $\mathbf{u}_i$ is the corresponding eigenvector.

By assumption, $\{\mathbf{u}_i\}_{i=1}^{r^o}$ is a subset of $\{\mathbf{v}_i\}_{i=1}^{N}$. Given initialized $\mathbf{Z} = \mathbf{W}^{(1)} = \sum_{i=1}^{N} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top$, in each iteration i, computation of matrix inner-products identifies the i-th largest eigenvalue magnitude $|\alpha_i|$, whose corresponding eigenvector is in $\{\mathbf{v}_i\}$. The weight update $s_i \leftarrow \lambda_i - \alpha_i$ is equivalent to the i-th rank-1 update in $\mathbf{W}^{(1)} - \mathbf{R}$. Thus, the full greedy algorithm computes the best rank-r^* approximation $\mathbf{R}^*$ of $\mathbf{W}^{(1)} - \mathbf{M}$ that minimizes the Frobenius norm $\|\mathbf{W}^{(1)} - \mathbf{M}\|_F^2$, and therefore the optimal solution to (4.29).

Given solutions $\{\mathbf{Z}^o\}$ to (4.29) for increasing r, the full greedy algorithm finds the best one $\mathbf{Z}^o$ among $\{\mathbf{Z}^o\}$ that minimizes (4.27) for a given η, which by Corollary 1 is also the optimal solution to (4.27). □

The complexity of the full greedy algorithm is dominated by the computation of the full eigenvector set $\{\mathbf{v}_i\}$ of $\mathbf{W}^{(1)}$, which in the worst case is $\mathcal{O}(N^3)$.

Note that in the general case when $\mathbf{W}^{(1)} - \mathbf{M}$ has different eigenvectors than $\mathbf{W}^{(1)}$, the full greedy algorithm is not globally optimal. Further, the eigenvectors of solution $\mathbf{Z}^* = \mathbf{W}^{(1)} - \mathbf{R}^*$ to (4.29) can differ from original $\mathbf{W}^{(1)}$, since by definition eigenvectors must be successive orthonormal arguments that minimize the Rayleigh quotient $\min_{\mathbf{v}} \frac{\mathbf{v}^\top \mathbf{Z}^* \mathbf{v}}{\mathbf{v}^\top \mathbf{v}}$ for symmetric and real $\mathbf{Z}^*$. Thus, the eigen-structure of the adjacency matrices slowly evolves over time in the general case.

C.1.d Fast Greedy Algorithm

To reduce the computation complexity of the full greedy algorithm, a faster variant of the algorithm is developed that does not require full eigen-decomposition of $\mathbf{W}^{(1)}$.

In the previous algorithm, if the index of the largest magnitude $|\alpha_i|$ among N candidates always corresponds to one of B *extreme* eigenvectors of $\mathbf{W}^{(1)}$ corresponding to

the B largest magnitude eigenvalues, then these B extreme eigenvectors can be computed in linear time, assuming $B \ll N$, via fast numerical linear algebra algorithms, such as *Locally Optimal Block Preconditioned Conjugate Gradient* (LOBPCG) [103]. Thus, a simple fast variant is to first compute B extreme eigenvectors $\mathbf{v}_i$ of $\mathbf{W}^{(1)}$, then compute $\alpha_i = \langle \mathbf{W}^{(1)} - \mathbf{M}, \mathbf{v}_i \mathbf{v}_i^\top \rangle = \mathbf{v}_i^\top (\mathbf{W}^{(1)} - \mathbf{M}) \mathbf{v}_i$, for $i = \{N - B + 1, \ldots, N\}$. Assuming $\mathbf{W}^{(1)} - \mathbf{M}$ is sparse—$\mathbf{W}^{(1)}$ specifies a sparse graph while $\mathbf{M}$ is a candidate for sparse $\mathbf{W}^{(2)}$—each inner-product computation has complexity $\mathcal{O}(N)$. To ensure $\mathbf{W}^{(1)} - \mathbf{M}$ is sparse, after computing gradient descent $\mathbf{W}^{(2)} - \epsilon \nabla_{\mathbf{W}^{(2)}}$ in Section C.1.a, a *hard-thresholding* is performed to sparsify $\mathbf{W}^{(2)} - \epsilon \nabla_{\mathbf{W}^{(2)}}$ as input to the proximal mapping. Thus, the complexity of the fast greedy algorithm is $\mathcal{O}(N)$.

C.1.e Comparison of Full and Fast Greedy

To compare the Full Greedy and Fast Greedy algorithms, 10 synthetic graphs of sizes $N \in \{50, 80, 100, 120, 150, 180, 200, 250, 300, 400\}$ are generated first. Each initial adjacency matrix $\mathbf{W}^{(1)}$ was sampled from a Bernoulli model with probability $p = 0.1$, then symmetrized with zero diagonal terms. A ground-truth matrix $\mathbf{W}^{(2)}$ is then constructed by adding a rank ≤ 3 update to $\mathbf{W}^{(1)}$, ensuring $\mathbf{W}^{(2)} - \mathbf{W}^{(1)}$ is exactly low-rank. To simulate a noisy version of the ground truth $\mathbf{W}^{(2)}$, small Gaussian perturbations are further added to obtain $\mathbf{M}$, which serves as a candidate for $\mathbf{W}^{(2)}$. A hard-thresholding is then applied to $\mathbf{M}$ using $\epsilon = 10^{-3}$ to sparsify it.

Full eigen-decomposition[8] is applied for the full greedy algorithm and LOBPCG[9] for the fast greedy algorithm to compute B extreme eigenvectors where $B = 10$ for all the used graphs. All experiments were performed in Python 3.8.13 (conda-forge) on

[8] numpy.linalg.eig()

[9] scipy.sparse.linalg.lobpcg()

Table 4.1: Runtime (seconds) and Frobenius error $\|\mathbf{Z} - \mathbf{W}^{(2)}\|_F$ for Full Greedy vs. Fast Greedy across ten graph sizes.

Size N	Full		Fast	
	Time(s)	Error	Time(s)	Error
50	0.0534	88.9936	0.0430	89.6692
80	0.3041	161.4570	0.0361	162.0582
100	0.5062	162.0395	0.0294	162.5187
120	1.1903	205.6949	0.0470	206.5915
150	3.2531	249.7200	0.0774	250.3542
180	6.3777	297.4400	0.0854	297.9521
200	13.8917	362.0635	0.0755	362.9100
250	39.6063	457.9621	0.1361	458.8128
300	87.3934	517.3216	0.2724	518.0888
400	405.0162	714.6182	0.6717	715.4951

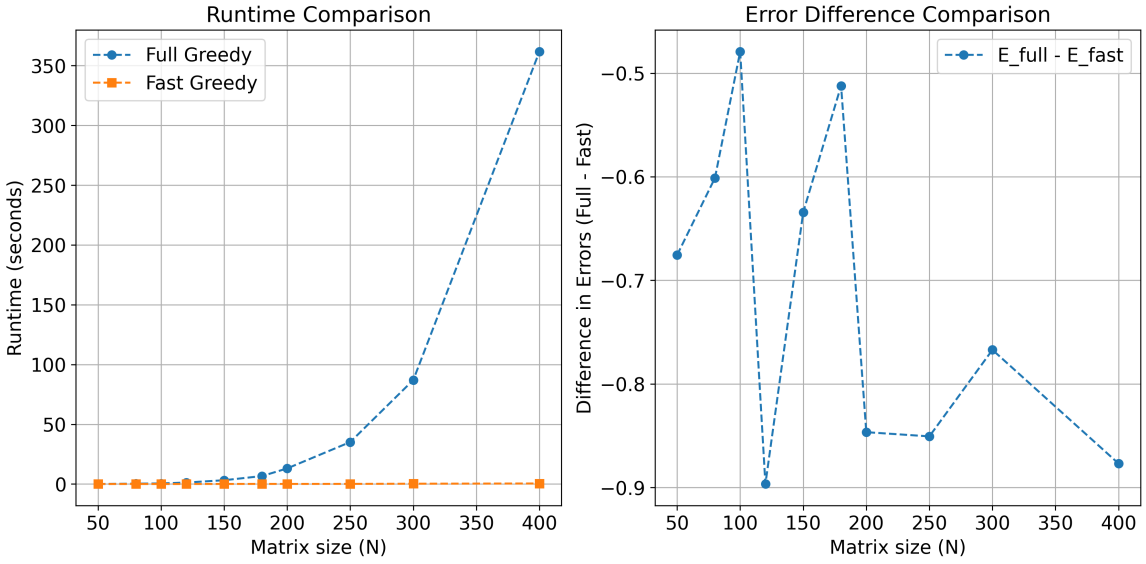

Figure 4.2: Comparison of runtime and estimation error for Full Greedy vs. Fast Greedy.

macOS 10.16 with an Intel(R) Core(TM) $i7 - 1068NG7$ CPU, using NumPy 1.24.4 and SciPy 1.10.1. The runtime (in seconds) and the Frobenius-norm error are recorded to measure how accurately each algorithm estimated the ground-truth matrix.

Table 4.1 presents these results, and Fig. 4.2 visualizes the same data. As Table 4.1

shows, the Full Greedy Algorithm achieves slightly lower reconstruction errors but requires significantly more time when N becomes larger. The Fast Greedy Algorithm, in contrast, completes each run in under a second for most sizes, offering dramatic speed-ups at the cost of a modest increase in error.

C.1.f Justification of Assumption

This section justifies the assumption that the important rank-1 update matrices $\mathbf{v}_i\mathbf{v}_i^\top$ for $\mathbf{W}^{(2)}$ are outer-products of eigenvectors $\mathbf{v}_i$ corresponding to the largest magnitude eigenvalues λ_i of $\mathbf{W}^{(1)}$. Specifically, the eigenvector $\mathbf{v}_1$ corresponding to the largest magnitude eigenvalue λ_1 of non-negative irreducible matrix $\mathbf{W}^{(1)}$ is called the *Perron vector*. By the Perron-Frobenius Theorem [104], it is a strictly positive vector, whose entry magnitudes $|v_{1,k}|$ denote the relative importance of node k. The importance measure here is *eigenvector centrality* [105], meaning that node k is important iff its neighbors l's are also important; *i.e.*, node k's importance value (centrality) v_k is a weighted linear combination of its neighbors' centralities v_l's:

$$v_k = \frac{1}{\lambda} \sum_{(k,l)\in\mathcal{E}} w_{k,l} v_l \tag{4.37}$$

$$\lambda \mathbf{v} = \mathbf{W}^{(1)}\mathbf{v}. \tag{4.38}$$

This implies $\mathbf{v}$ is an eigenvector of $\mathbf{W}^{(1)}$ with eigenvalue λ. Among $\mathbf{W}^{(1)}$'s eigenvectors, $\mathbf{v}_1$ is the only eigenvector with strictly positive entries. Hence, $\mathbf{v}_1$ is interpreted as an importance vector, and outer-product $\mathbf{v}_1\mathbf{v}_1^\top$ emphasizes interaction between important nodes.

C.2 Projection Operator

To address the third convex but non-smooth term in (4.23), one can easily show that the proximal mapping for indicator function $\mathrm{I}_W(\mathbf{W}^{(2)})$ is a convex-set projection operator [101]:

$$\mathrm{prox}_{I_W}(\mathbf{M}) = \mathrm{proj}_{\mathcal{C}}(\mathbf{M}) \tag{4.39}$$

$$(\mathrm{proj}_{\mathcal{C}}(\mathbf{M}))_{i,j} = \begin{cases} 0 & \text{if } i = j \\ 0 & \text{if } i \neq j, M_{i,j} < 0 \\ \frac{M_{j,i}+M_{i,j}}{2} & \text{if } i \neq j, M_{j,i}, M_{i,j} \geq 0 \end{cases} . \tag{4.40}$$

C.3 Algorithm Summary

The PGD algorithm can be summarized as follows:

$$\mathbf{W}^{(2)} \leftarrow \mathrm{proj}_{\mathcal{C}}(\mathrm{prox}_{g,\mathbf{W}^{(1)}}(\mathbf{W}^{(2)} - \epsilon\nabla_{\mathbf{W}})). \tag{4.41}$$

In words, first, the first term in (4.23) is decreased by updating $\mathbf{W}^{(2)}$ in the opposite gradient direction $-\nabla_{\mathbf{W}}$, where $\epsilon > 0$ is a step size. Proximal mapping is then performed in (4.27). Finally, convex-set projection is performed (4.40). (4.41) is executed iteratively until solution $\mathbf{W}^{(2)}$ converges. Signal $\mathbf{x}_2$ and adjacency matrix $\mathbf{W}^{(2)}$ are alternately optimized until the pair converges.

1. Perform gradient descent $\mathbf{M}^1 \leftarrow \mathbf{W}^{(2)} - \epsilon\nabla_{\mathbf{W}}$.
2. Perform proximal mapping $\mathbf{M}^2 \leftarrow \mathrm{prox}_{g,\mathbf{W}^{(1)}}(\mathbf{M}^1)$.
3. Perform projection $\mathbf{W}^{(t)} \leftarrow \mathrm{proj}_{\mathcal{C}}(\mathbf{M}^2)$.

To interpolate $\mathbf{y}_t \in \mathbb{R}^{M_t}$ to $\mathbf{x}_t \in \mathbb{R}^N$ for each $t \in \{2, \ldots, T\}$, each $\mathbf{x}_t$ is computed iteratively, *i.e.*, first $\mathbf{x}_2$ and $\mathbf{W}^{(2)}$ are computed using previous $\mathbf{W}^{(1)}$, then $\mathbf{x}_3$ and $\mathbf{W}^{(3)}$ are computed using previous $\mathbf{W}^{(2)}$, etc.

C.4 Proof of Convergence

By incorporating the constraints via the indicator function (4.18) into the objective, the optimization problem (4.4) for $\mathbf{W}^{(2)}$ and $\mathbf{x}_2$ can be reformulated in an unconstrained form as follows:

$$\begin{aligned}\min_{\mathbf{W}^{(2)},\mathbf{x}_2} \; & (\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2)^\top \mathbf{Q}(\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2) + \xi \left\|\mathbf{x}_2 - \mathbf{x}_1\right\|_2^2 \\ & + \mu \operatorname{Tr}((\operatorname{diag}(\mathbf{W}^{(2)}\mathbf{1} - \mathbf{W}^{(2)})\mathbf{x}_2\mathbf{x}_2^\top) \\ & + \eta\, \Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(2)}) + \mathbf{I}_W(\mathbf{W}^{(2)}). \end{aligned} \tag{4.42}$$

In the proposed alternating optimization, when $\mathbf{W}^{(2)}$ is fixed, solving for the *optimal* solution $\mathbf{x}_2^*$ via the linear system (4.17) ensures that the sum of the first, second, and third terms in (4.42) is monotonically non-increasing with respect to the previous solution $\mathbf{x}_2$.

Conversely, when $\mathbf{x}_2$ is fixed, the proposed proximal gradient descent (PGD) algorithm (4.41) computes a new $\mathbf{W}^{(2)*}$, ensuring that the sum of the third, fourth, and fifth terms in (4.42) is monotonically non-increasing with respect to the previous solution $\mathbf{W}^{(2)}$. According to [101], if the gradient $\nabla_{\mathbf{W}}$ is *Lipschitz* continuous with constant L, the PGD algorithm (4.41) converges at a rate of $\mathcal{O}(1/k)$ when a step size $\epsilon \in (0, 1/L]$ is employed. Furthermore, [106] notes that convergence is maintained for

step sizes smaller than $2/L$, although for step sizes larger than $1/L$, the method ceases to be a *majorization-minimization* approach. The constant L can be upper-bounded by the largest eigenvalue of the Hessian matrix.

Since all terms in (4.42) are lower-bounded by zero, the objective function itself is lower-bounded. Consequently, the alternating optimization algorithm does not oscillate and converges to a local minimum in a finite number of steps.

C.5 Algorithm Unrolling

C.5.a Unrolling of Optimization Algorithm

Given the alternating algorithm to compute $\mathbf{x}_2$ and $\mathbf{W}^{(2)}$ developed in the previous section, this section focuses on unrolling its iterations into layers to form a lightweight feed-forward network amenable to end-to-end data-driven parameter tuning. The tunable parameters in our algorithm are:

1. prior weights μ, ξ and η in objective (4.4);
2. two CG parameters κ and β corresponding to conjugate gradient descent step size and momentum used to solve linear system (4.17) for $\mathbf{x}_2^*$ given $\mathbf{W}^{(2)}$;
3. the step size ϵ for gradient descent in the PGD algorithm (4.41) for $\mathbf{W}^{(2)}$; and
4. the diagonal entries $Q_{i,i}$ in matrix $\mathbf{Q}$ of the fidelity term[10].

Note that these parameters are tuned per unrolled layer.

Based on the alternating optimization algorithm described earlier, the proposed unrolled module consists of two types of layers. The first layer type optimizes the

[10]With a constraint that the sum of the diagonal elements equals the number of nodes N, same as the sum of the diagonals in the identity matrix

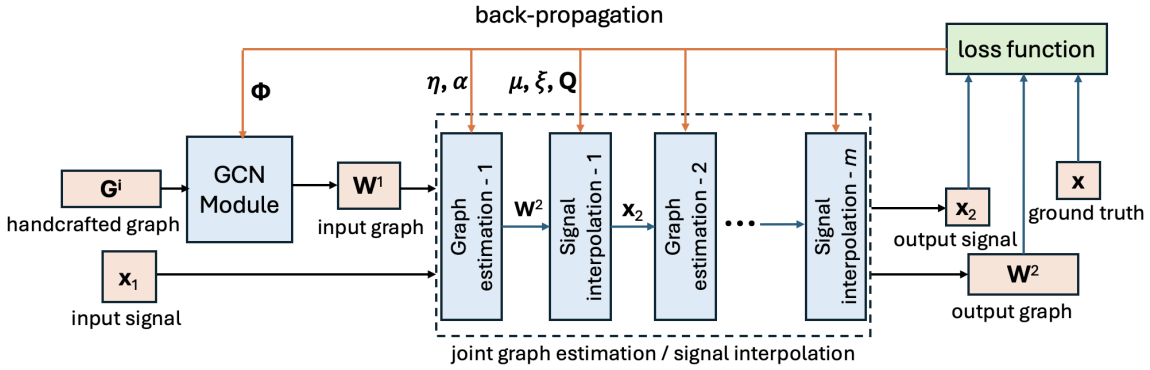

Figure 4.3: Neural Network Architecture with GCN and Unrolled Layer.

adjacency matrix $\mathbf{W}^{(2)}$ assuming the signal $\mathbf{x}_2$ is fixed, while the second type optimizes $\mathbf{x}_2$ assuming $\mathbf{W}^{(2)}$ is fixed. See Fig. 4.3 for an illustration.

C.5.b Learning an Initial $\mathbf{W}^{(1)}$ via GCN

To construct an initial adjacency matrix $\mathbf{W}^{(1)}$, handcrafted features $\mathbf{f}_i$ per node (*e.g.*, location and land value) are first used to build an initial graph $\mathcal{G}^i$ based on feature distance, *i.e.*, $w_{i,j} = \exp(-\|\mathbf{f}_i - \mathbf{f}_j\|_2^2)$. Then $\mathcal{G}^i$ and the node feature matrix $\mathbf{X}$ (which includes $\mathbf{f}_i$s) are feed into a shallow *Graph Convolutional Network* (GCN) [1, 107].

Fig. 4.4 shows the detailed architecture of the GCN. A GCNConv layer aggregates feature information from its neighbors. After feature aggregation, a non-linear transformation (ReLU) is applied to the resulting outputs. By stacking multiple layers, the final representation of each node receives relevant features from a larger neighborhood, and the output graph specified by $\mathbf{W}^{(1)}$ better captures the data's structure, providing more meaningful features. Subsequently, $\mathbf{W}^{(1)}$ is used as the initial graph for the unrolled alternating algorithm. Note that a very shallow GCN is used in this algorithm, so the number of parameters is small, which makes it suitable for

data-starved scenarios.

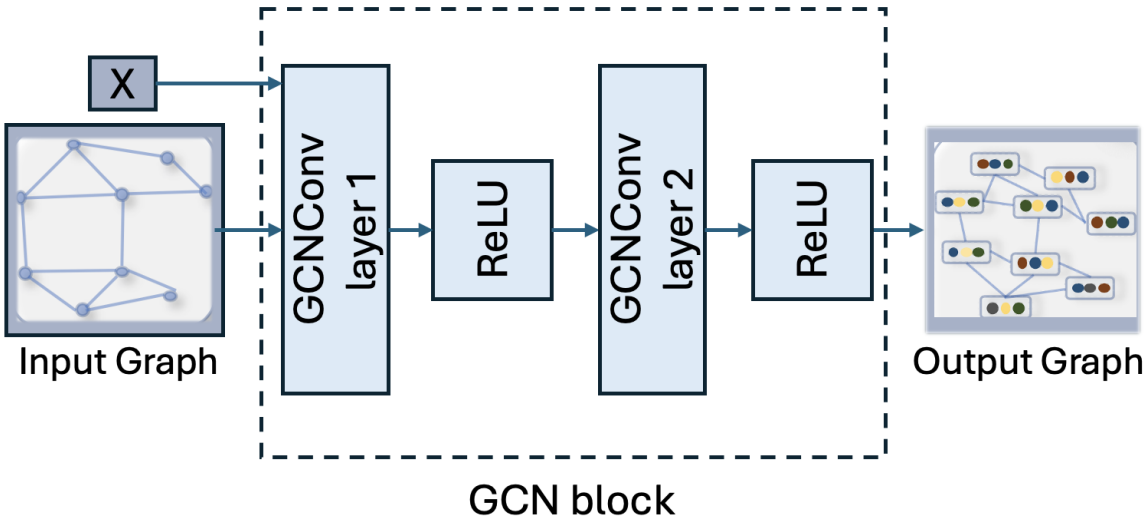

Figure 4.4: Shallow GCN Architecture

D Version 1 vs 2: Key Improvements

Compared to the conference version [34], the journal version of this algorithm makes several notable improvements:

- Instead of convexifying the rank of the adjacency difference $\Gamma(\mathbf{W}^{(1)} - \mathbf{W}^{(2)})$ to the nuclear norm $\|\mathbf{W}^{(1)} - \mathbf{W}^{(2)}\|_*$ with complexity $\mathcal{O}(N^3)$, the rank term is minimized directly and it is proved that the polynomial-time full greedy algorithm is globally optimal under an eigenvector assumption.

- A fast greedy approximation algorithm in version 2 is proposed that computes only B extreme eigenvectors and reduces complexity to $\mathcal{O}(N)$, where $B \ll N$. The low-rank prior is justified using the Perron vector, arguing that the rank-1 updates capture interactions among important nodes.

- A new diagonal matrix $\mathbf{Q}$ is incorporated into the fidelity term, as $(\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2)^\top \mathbf{Q}(\mathbf{y}_2 - \mathbf{H}^{(2)}\mathbf{x}_2)$, where the diagonal entries $Q_{i,i}$'s specify reliabilities of individual observed samples.

- Version 2 unrolls iterations of the algorithm into layers of a feed-forward network, enabling end-to-end learning of prior weights and reliability weights $Q_{i,i}$'s, as well as a shallow *graph convolutional network* (GCN) to initialize adjacency matrix $\mathbf{W}^{(1)}$.

E Experiments

E.1 Datasets

This section presents five datasets used to evaluate the proposed framework, each characterized by a distinct temporal scale and data structure. These diverse time frames demonstrate that the concept of "slowly time-varying" varies depending on the application and the inherent characteristics of the data.

E.1.a Synthetic Data

Consider a dataset comprising m vectors $Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)} \in \mathbb{R}^N$, where each vector $Y^{(i)}$ represents an observation or data point. These vectors are assumed to be independent and identically distributed (i.i.d.) as multivariate Gaussian random vectors:

$$Y^{(i)} \sim \mathcal{N}(\mu, \mathbf{\Sigma}), \tag{4.43}$$

where $\mu \in \mathbb{R}^N$ is the mean vector and $\mathbf{\Sigma} \in \mathbb{R}^{N \times N}$ is the positive definite covariance matrix, both of which are unknown. The maximum likelihood method is employed

to estimate μ and $\mathbf{\Sigma}$ by maximizing the log-likelihood function.

The log-likelihood function for the observations $y^{(i)}$ (realizations of $Y^{(i)}$) is:

$$\begin{aligned} l(\mu, \mathbf{\Sigma} \mid y^{(i)}) &= \log \prod_{i=1}^{m} f_{Y^{(i)}}(y^{(i)} \mid \mu, \mathbf{\Sigma}) \\ &= \sum_{i=1}^{m} \log \left(\frac{1}{(2\pi)^{N/2} |\mathbf{\Sigma}|^{1/2}} \exp \left(-\frac{1}{2} (y^{(i)} - \mu)^\top \mathbf{\Sigma}^{-1} (y^{(i)} - \mu) \right) \right) \\ &= -\frac{mN}{2} \log(2\pi) - \frac{m}{2} \log |\mathbf{\Sigma}| - \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \mu)^\top \mathbf{\Sigma}^{-1} (y^{(i)} - \mu). \end{aligned} \tag{4.44}$$

To estimate μ, the derivative of the log-likelihood with respect to μ is computed:

$$\frac{\partial}{\partial \mu} l(\mu, \mathbf{\Sigma} \mid y^{(i)}) = \sum_{i=1}^{m} \mathbf{\Sigma}^{-1} (y^{(i)} - \mu) = 0.$$

Since $\mathbf{\Sigma}$ is positive definite, $\mathbf{\Sigma}^{-1}$ is invertible, yielding:

$$\begin{aligned} 0 = \sum_{i=1}^{m} (y^{(i)} - \mu) \implies m\mu = \sum_{i=1}^{m} y^{(i)}, \\ \hat{\mu} = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} = \bar{y}. \end{aligned} \tag{4.45}$$

To estimate $\mathbf{\Sigma}$, the derivative of the log-likelihood with respect to $\mathbf{\Sigma}^{-1}$ is computed. Using the property $|\mathbf{\Sigma}^{-1}| = 1/|\mathbf{\Sigma}|$ and the trace trick, the log-likelihood is rewritten as:

$$l(\mu, \mathbf{\Sigma} \mid y^{(i)}) = C + \frac{m}{2} \log |\mathbf{\Sigma}^{-1}| - \frac{1}{2} \sum_{i=1}^{m} \operatorname{tr} \left[(y^{(i)} - \mu)(y^{(i)} - \mu)^\top \mathbf{\Sigma}^{-1} \right], \tag{4.46}$$

where $C = -\frac{mN}{2}\log(2\pi)$ is a constant. The derivative with respect to $\mathbf{\Sigma}^{-1}$ is:

$$\frac{\partial}{\partial \mathbf{\Sigma}^{-1}} l(\mu, \mathbf{\Sigma} \mid y^{(i)}) = \frac{m}{2}\mathbf{\Sigma} - \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \mu)(y^{(i)} - \mu)^{\top}. \tag{4.47}$$

Setting this derivative to zero and solving for $\mathbf{\Sigma}$ yields:

$$\begin{aligned} 0 &= m\mathbf{\Sigma} - \sum_{i=1}^{m}(y^{(i)} - \mu)(y^{(i)} - \mu)^{\top}, \\ \hat{\mathbf{\Sigma}} &= \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{\mu})(y^{(i)} - \hat{\mu})^{\top}. \end{aligned} \tag{4.48}$$

For synthetic data generation, the conditional distribution $\Pr(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is modeled as a product of two Gaussian distributions. The combined distribution is Gaussian with mean and covariance given by:

$$\mu_{12} = \frac{\mu_1\mathbf{\Sigma}_2 + \mu_2\mathbf{\Sigma}_1}{\mathbf{\Sigma}_1 + \mathbf{\Sigma}_2}, \tag{4.49}$$

$$\mathbf{\Sigma}_{12} = \frac{\mathbf{\Sigma}_1\mathbf{\Sigma}_2}{\mathbf{\Sigma}_1 + \mathbf{\Sigma}_2}. \tag{4.50}$$

Here, the first distribution is Gaussian with mean $\mu_1 = 0$ and covariance $\mathbf{\Sigma}_1 = (\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1}$, where $\mathbf{L}^{(t)}$ is a time-dependent matrix (e.g., a Laplacian) and $\alpha > 0$ is a regularization parameter. The second distribution is Gaussian with mean $\mu_2 = \mathbf{x}_{t-1}$ and covariance $\mathbf{\Sigma}_2 = \beta\mathbf{I}$, where $\beta > 0$ is a scalar and $\mathbf{x}_{t-1} \in \mathbb{R}^N$ represents the previous state.

Substituting these into the combined distribution, the estimated parameters are:

$$\hat{\mu} = \frac{\mathbf{x}_{t-1}\,\mathrm{diag}((\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1})}{\mathrm{diag}((\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1}) + \mathrm{diag}(\beta\mathbf{I})}, \tag{4.51}$$

$$\hat{\mathbf{\Sigma}} = \frac{\mathrm{diag}((\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1}) \cdot \mathrm{diag}(\beta\mathbf{I})}{\mathrm{diag}((\mathbf{L}^{(t)} + \alpha\mathbf{I})^{-1}) + \mathrm{diag}(\beta\mathbf{I})}. \tag{4.52}$$

These estimates enable the generation of synthetic data points that align with the observed temporal dynamics. Figure 4.5 illustrates the temporal similarity of synthetic data samples over time.

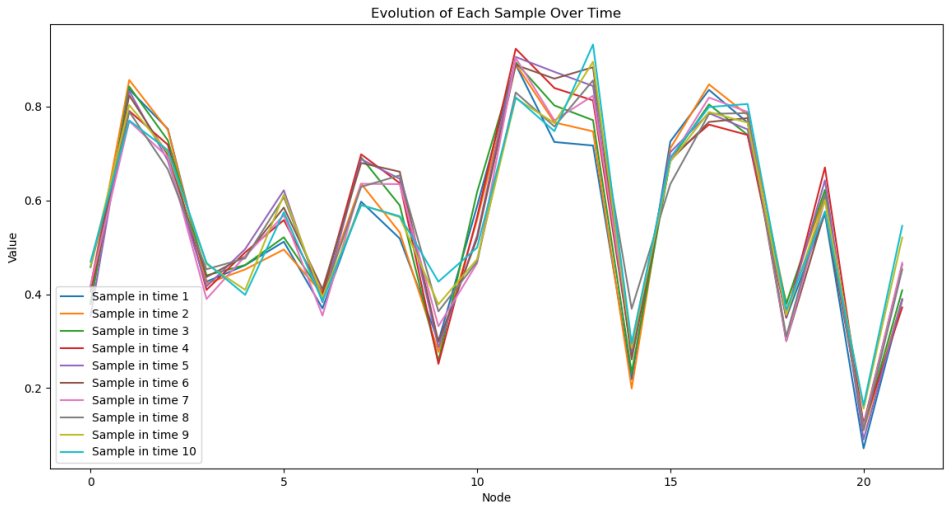

Figure 4.5: Evolution of each Synthetic sample over 10 time instants.

E.1.b Quarterly Farmland Sales Dataset

This dataset, obtained from the USDA's National Agricultural Statistics Service (NASS)[11], contains farmland sales records from various counties in Iowa, US, spanning the fiscal quarters from 2019 to 2023. It includes attributes such as gross price,

[11] `https://www.nass.usda.gov/`

gross acres, tillable acres, sale date, and soil rating.

Each data point represents a farmland sale in dollars per acre. Iowa was chosen due to its extensive availability of county-level data in every quarter, allowing for validation of the low-rank update assumption across consecutive quarters.

Given that soil rating significantly affects land value, an adjusted price metric is employed, derived through linear regression between the actual price and the soil rating. To illustrate the linear relationship more clearly, Figure 4.6 displays a scatter plot of farmland average price per acre versus soil rating, alongside the best-fit line obtained from the linear regression used to create the adjusted price metric.

Figure 4.7 presents a snapshot of this dataset as a data frame.

E.1.c Yearly Crop Yield Dataset

This second dataset aggregates yearly corn yield data (2010–2020) from the USDA's National Agricultural Statistics Service (NASS) for 169 counties across Iowa and Wisconsin. Each data point represents corn yield in bushels per acre, reflecting variations in crop production over time and across geographic regions. Additional features include clay percentage, an estimate of available water storage (AWS), an estimate of soil organic carbon stock (SOC), and a 2D location feature, providing a detailed characterization of the agricultural environments in each county.

These 169 counties were chosen to ensure comprehensive annual data throughout the entire time period, enabling consistent analysis of spatial and temporal patterns in corn yields. Similar to the farmland dataset, this crop yield dataset supports the exploration of how underlying factors—such as soil composition and location—affect key agricultural outcomes.

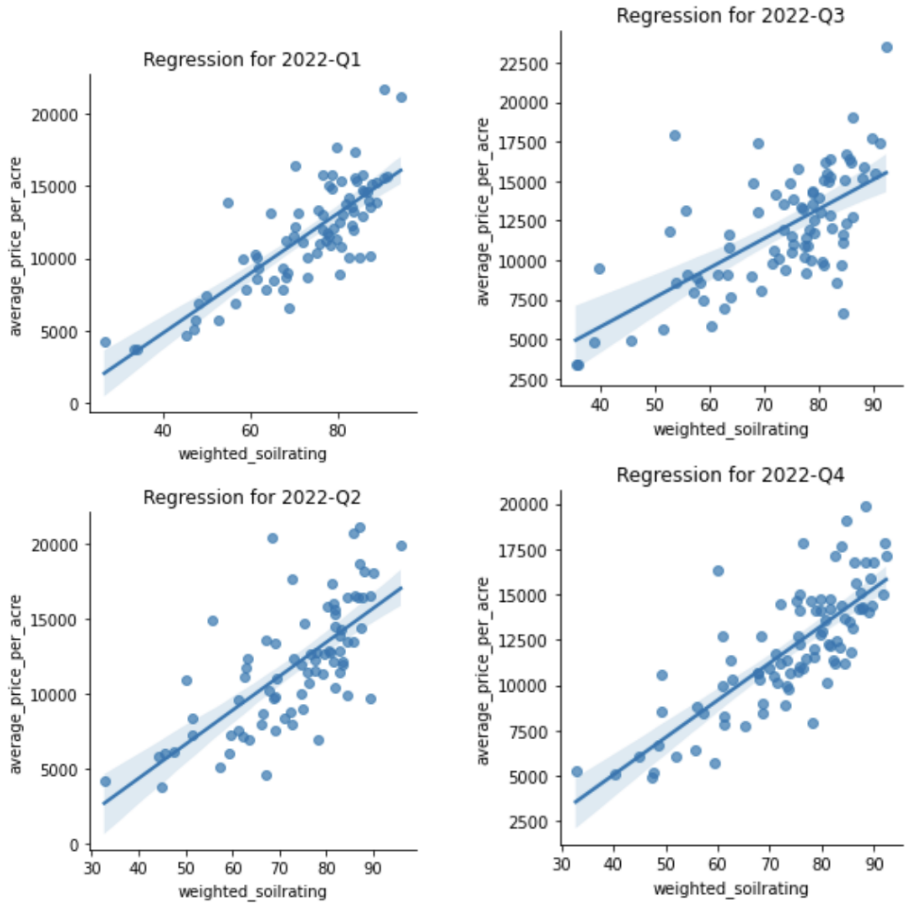

Figure 4.6: Scatter plot of farmland average price per acre vs. soil rating in 2022.

sale_quarter	county_fips	avg_lon	avg_lat	average_price_per_acre	avg_soilrating	weighted_soilrating	average_tillable_per_acre	price_per_acre_count
2020-Q1	1	-94.533859	41.369471	5785.984200	64.0800	64.415851	50.485000	2
2020-Q3	1	-94.597157	41.264009	5057.469787	61.8000	61.800000	37.910000	1
2020-Q4	1	-94.462842	41.292362	4193.910557	52.3700	52.880023	86.230000	2
2021-Q1	1	-94.385793	41.318491	7015.100215	68.0125	64.310241	111.540000	4
2021-Q3	1	-94.476842	41.321134	6820.509484	59.7800	61.790238	160.294000	5
2021-Q4	1	-94.498081	41.341360	7617.412735	62.5920	62.515041	78.845145	10
2022-Q1	1	-94.524218	41.296591	10039.104918	63.4800	61.547925	110.840000	3
2022-Q2	1	-94.602346	41.392455	8300.385547	51.4900	51.490000	73.150000	1
2022-Q3	1	-94.638204	41.335219	5827.432155	58.6350	60.193854	238.990000	2
2022-Q4	1	-94.406203	41.406294	8280.908724	61.9775	61.392323	74.825000	4

Figure 4.7: Snapshot of the farmland sales dataset.

E.1.d Daily Temperature Dataset

This dataset, obtained via the Meteostat Python library[12], comprises daily maximum temperature observations from 107 weather stations located in Ontario, Canada. Each entry contains a station ID, station name, geographic coordinates (latitude and longitude), and the recorded daily maximum temperature.

To capture short-term temperature trends, the dataset spans the period from June 1, 2020, to July 31, 2020. These observations originate from a combination of public and governmental weather data interfaces aggregated by Meteostat, providing a comprehensive source of daily temperature measurements.

E.1.e Hourly Windmill Dataset

This dataset, accessible through the PyTorch Geometric Temporal Datasets[13], contains hourly energy output observations from 26 windmills situated in a European country. Spanning more than two years, it includes details such as windmill identifiers and their corresponding hourly energy outputs. Each data point encapsulates the wind-driven power generation at a single windmill for a given hour, offering insights into temporal variations in energy production across diverse geographical locations.

E.2 Experiment Setup and Results for Version 1 (Conference)

E.2.a Setup

The first version of the algorithm published in [34] is evaluated using the Quarterly Farmland Sales Dataset (Section E.1.b) and the Yearly Crop Yield Dataset

[12] `https://dev.meteostat.net/`
[13] `WindmillOutputMediumDatasetLoader`

(Section E.1.c). To test the algorithm's performance, four alternative schemes are considered:

1. `SVD` [33]: This method makes structural assumptions in the spectral domain, assuming the first few singular vectors remain unchanged, thus implying minimal shifts in the underlying low-rank structure over time.

2. `Tikhonov` [32]: Here, the edges of the graph are assumed to change smoothly over time, based on the premise that the Frobenius norm of the adjacency matrix variation is small.

3. `sparse` [31]: This scheme extends `Tikhonov` by assuming that only a few edges require updates. A sparsity prior on the difference of successive adjacency matrices enforces this assumption, leading to a variant commonly referred to as TVGL-sparse.

4. `OTS`: An off-the-shelf solver from the SciPy optimization library[14] that directly solves (4.4) using the generic gradient-based method `SLSQP`, which accommodates constraints.

In `SVD`, `Tikhonov`, and `sparse`, the adjacency matrix is first learned independently for each time step. The signal interpolation is then performed by solving (3.8) under the learned adjacency. In contrast, the proposed approach jointly optimizes both the signal and the adjacency matrix.

Parameters for the proposed approach are set to $(\mu = 1 \times 10^{-5}, \alpha = 1 \times 10^{-5}, \eta = 10, \gamma = 0.03)$ for the farmland sales data, and $(\mu = 1 \times 10^{-3}, \alpha = 1 \times 10^{-4}, \eta = 10, \gamma = 0.08)$ for the crop yield data. For `Tikhonov` $(\alpha = 0.1, \gamma = 10)$ and $(\alpha = 0.8, \gamma = 10)$

[14] `https://docs.scipy.org/doc/scipy/reference/optimize.html`

are used; for `TVGL-sparse` ($\alpha = 0.1, \beta = 10, \eta = 10$) and ($\alpha = 0.5, \beta = 10, \eta = 10$) are used for sales and yield data, respectively. Meanwhile, `SVD` maintains $k = 3$ for the farmland sales data and $k = 10$ for the crop yield data.

To confirm the low-rank update assumption in the farmland sales dataset, graphs were constructed based on 22 counties with available data across fiscal quarters from 2019 to 2023. Specifically, a full graph was first built using the average adjusted price per acre for each county at various quarters. Weight $w_{i,j}$ of edge (i, j) connecting counties i and j is computed as

$$w_{i,j} = \exp\left(-\frac{|y_i - y_j|^2}{\Sigma_y^2}\right) \tag{4.53}$$

where y_i is the *adjusted price mean* of county i, and parameter $\sigma_y = 1.9$ was chosen for optimal performance. The resulting full graph was then sparsified using the kNN ($k = 10$). Using these graphs for each quarter, the difference matrix $\mathbf{W}_{t+1} - \mathbf{W}_t$ is observed to be approximately low-rank. Fig. 4.8 illustrates the distribution of eigenvalues for $\mathbf{W}_{t+1} - \mathbf{W}_t$ and $\mathbf{W}_t$ at $t = 2021\_Q1$ and $t + 1 = 2021\_Q2$. Only a few eigenvalues of $\mathbf{W}_{t+1} - \mathbf{W}_t$ have large magnitudes, whereas most are close to zero. In contrast, the majority of eigenvalues of $\mathbf{W}_t$ are larger in magnitude, highlighting that $\mathbf{W}_t$ itself is full-rank (or close to rank $N - 1$ in the case of a connected graph), while $\mathbf{W}_{t+1} - \mathbf{W}_t$ is nearly low-rank. Similar behavior was also observed in the crop yield dataset, empirically validating the assumption in (4.1).

E.2.b Results

Tables 4.2, 4.3, and 4.4 present the performance of various methods on the Quarterly Farmland Sales Dataset, the Crop Yield Dataset, and the Synthetic Dataset,

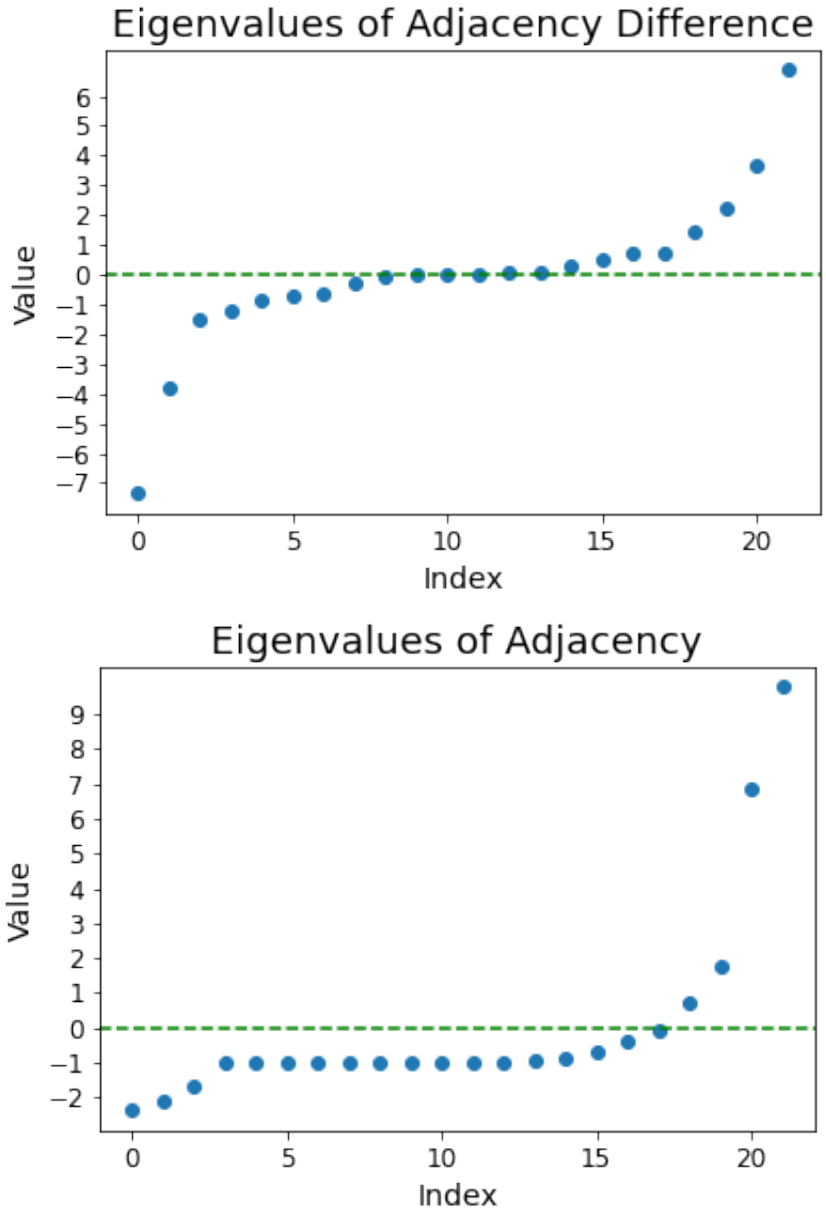

Figure 4.8: (a) Eigenvalues of $\mathbf{W}_{Q2} - \mathbf{W}_{Q1}$; (b) Eigenvalues of $\mathbf{W}_{Q1}$.

respectively.

For the Quarterly Farmland Sales Dataset, the experiment involved randomly omitting 11 observations from each of four quarterly time steps, with the objective of

recovering the missing entries. As detailed in Table 4.2, the proposed method outperforms `SVD`, `Tikhonov`, `sparse`, and `OTS`, achieving lower mean absolute error (MAE), root mean square error (RMSE), and mean absolute percentage error (MAPE).

Similarly, Table 4.3 summarizes results for the larger Crop Yield Dataset, comprising 169 nodes, where 70 observations were randomly removed at each of four yearly time steps (2015–2018). The proposed method consistently demonstrates superior performance across all metrics. Notably, `OTS` failed to converge within the two-hour runtime limit, highlighting the scalability limitations of generic solvers in high-dimensional settings.

For the Synthetic Dataset, the experiment randomly omitted 11 observations from each of four time steps. As shown in Table 4.4, the proposed method achieves lower errors compared to `SVD`, `Tikhonov`, and `sparse`, underscoring its effectiveness.

Fig. 4.9 compares the learned adjacency matrices, revealing that the difference between the estimated adjacency and the ground truth (in the Frobenius norm sense) is smallest for the proposed method. Meanwhile, Fig. 4.10 visualizes the interpolation error for each method across counties in Iowa and Wisconsin. The proposed method achieves fewer regions of significant error, and wherever such errors do occur, competing approaches exhibit even larger deviations. Since `sparse` and `Tikhonov` perform similarly, only one of these is shown for comparison on the map.

Table 4.2: Average performance of different methods for land sales dataset over 4 quarters

Metric	`OTS`	proposed	`SVD`	`Tikhonov`	`sparse`
MAE	877.61	**682.77**	1351.63	881.84	881.66
RMSE	1652.11	**1191.65**	1997.19	1544.48	1543.56
MAPE	0.084	**0.066**	0.141	0.086	0.085
Time	24.8 s	19.9 ms	142 ms	202 ms	189 ms

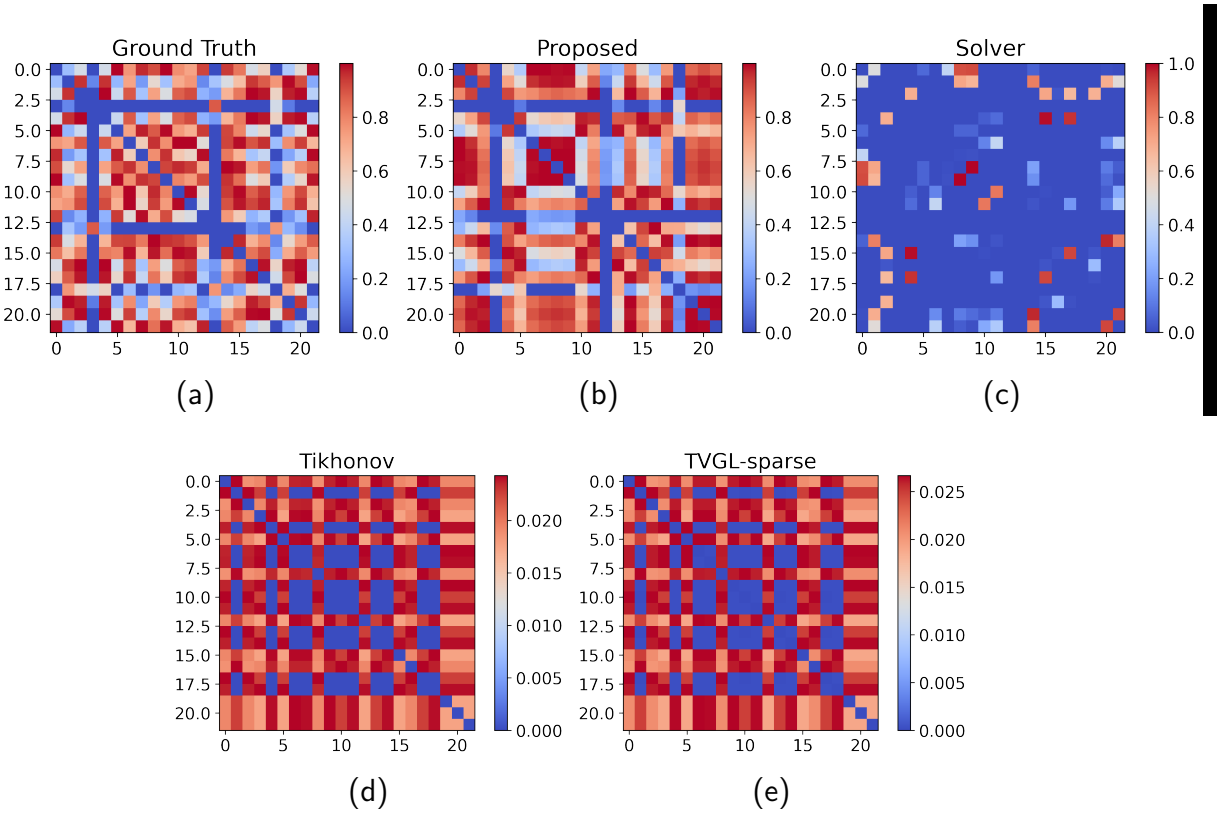

Figure 4.9: Comparison of ground truth and computed adjacency matrices. **(a)** Ground Truth for 2021-$Q2$, **(b)** Proposed (diff from GT = 8.51), **(c)** `OTS` (diff from GT = 13.38), **(d)** `Tikhonov` (diff from GT = 14.31), **(e)** `sparse` (diff from GT = 14.28).

Table 4.3: Average performance of different methods for Crop Yield dataset over 4 years

Metric	proposed	`SVD`	`Tikhonov`	`sparse`
MAE	**6.17**	8.46	10.64	10.61
RMSE	**11.14**	11.76	15.92	16.01
MAPE	**0.033**	0.046	0.058	0.057
Time	248 ms	1.13 s	1.24 s	1.41 s

E.3 Experiment Setup and Results for Version 2 (Journal)

E.3.a Setup

This experiment compares the updated journal approach against the original conference method [34] using three distinct datasets. As in the conference version, `Tikhonov`

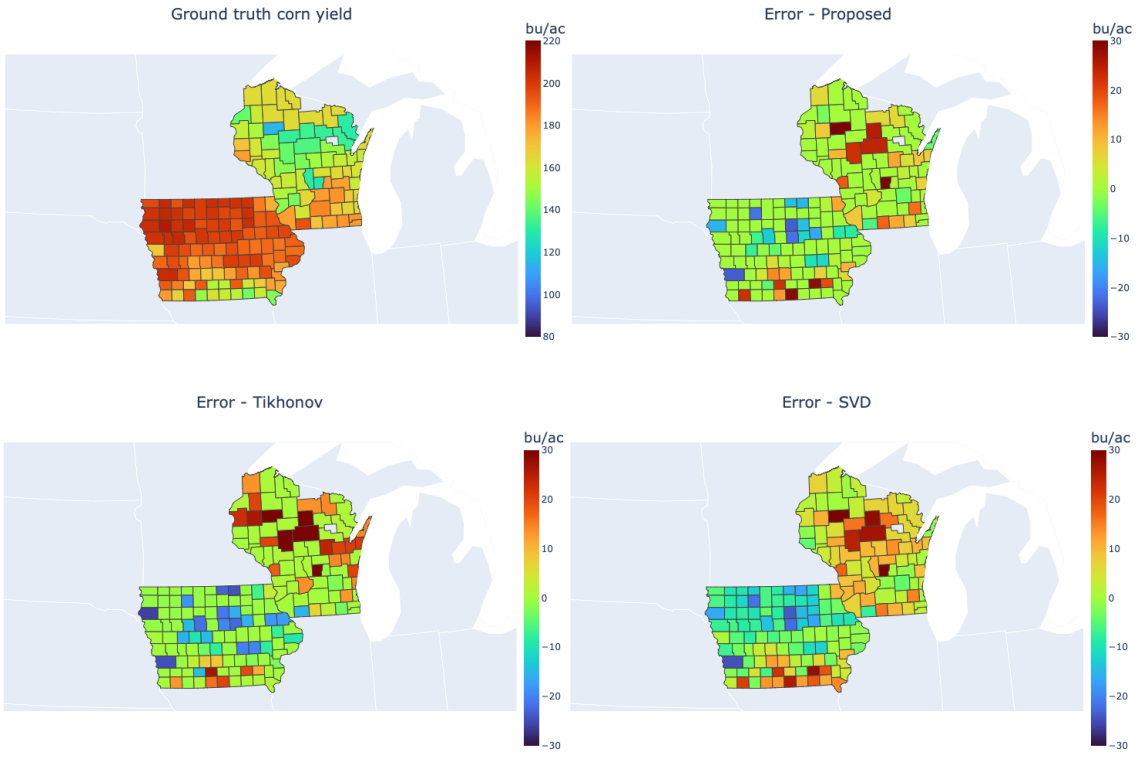

Figure 4.10: Map of interpolation error (in terms of bushel per acre) for all the counties in Iowa and Wisconsin.

Table 4.4: Average performance of different methods for Synthetic dataset over 4 time instants

Metric	proposed	`SVD`	`Tikhonov`	`sparse`
MAE	**0.045**	0.139	0.093	0.104
RMSE	**0.070**	0.152	0.128	0.138
MAPE	**0.483**	0.601	0.585	0.547

and `sparse` are included, while `SVD` and `OTS` are omitted due to their slower performance in earlier tests. An additional deep learning-based approach, `NestedDAU` [108], is also tested to strengthen the comparisons.

`NestedDAU` unrolls Plug-and-Play ADMM (PnP-ADMM) [109–112] into a trainable neural network. Each unrolled layer separates the computation into an "inverse"

step for partially observed signal nodes and a "denoising" step that imposes a graph-structured prior. The internal denoiser is itself an unrolled ADMM procedure specialized to graph signals, yielding a nested design. By end-to-end training of the layerwise parameters, `NestedDAU` adaptively balances fidelity and regularization while requiring relatively few training samples. However, unlike the time-varying model proposed here, `NestedDAU` does not learn or update the graph topology across multiple time intervals; it relies on a single, fixed adjacency structure.

For the farmland sales dataset (2021-Q2 to 2023-Q2), data from 22 Iowa counties are selected across eight consecutive quarters. The signals are min-max normalized, and 11 out of 22 observations per quarter are randomly omitted (50% missing rate). Gaussian noise with mean 0 and standard deviation 0.9 is then added, with the goal of recovering the complete signals for all these quarters.

For the daily temperature dataset (June 2–21, 2020), daily maximum temperature readings from 107 weather stations in Ontario are used over 20 consecutive days. Each day's measurements are normalized, 40 out of 107 observations are randomly removed ($\sim 35\%$ missing rate), and the same Gaussian noise is introduced. The task is to reconstruct the full daily temperature signals.

Finally, the hourly windmill dataset originally contains 17,472 hourly measurements from 26 windmills; the first 10 hours are used here. Each hour's signals are normalized, 13 out of 26 observations are omitted (50% missing rate), and Gaussian noise is added. The aim is to estimate the complete windmill output signals over this 10-hour window.

For the farmland sales dataset, the conference method uses parameters ($\mu = 0.1, \alpha = 0.1, \eta = 0.1, \gamma = 0.001$). Both `Tikhonov` and `TVGL-sparse`, use ($\alpha = 0.1, \gamma =$

0.001). For `NestDAU`, the number of unrolled layers P is set to 5, and the number of channels is 1. In the journal version, initial parameters are set to ($\mu = 0.1$, $\xi = 10^{-5}$, and $\eta = 0.1$) in objective (4.4). The step size ϵ for gradient descent in the PGD algorithm (4.41) is 0.1, and the diagonal entries $Q_{i,i}$ in matrix $\mathbf{Q}$ are initialized based on the number of observations. The same initial settings apply to all three datasets for both the journal method and `NestDAU`, since they both learn parameters via algorithm unrolling for each dataset, thereby further improving interpolation performance.

For windmill dataset The conference method uses parameters ($\mu = 0.5, \alpha = 0.5, \eta = 2, \gamma = 0.01$). For `Tikhonov` and `TVGL-sparse` , the settings are ($\alpha = 0.5, \gamma = 0.001$.

E.3.b Results

Figures 4.11 and 4.12 summarize the performance of five competing methods on the farmland sales dataset across eight quarters. The updated journal method achieves the lowest error metrics in nearly every quarter, demonstrating its consistent improvements over the other approaches.

Figures 4.13 and 4.14 illustrate the performance of five competing methods on the daily temperature dataset from 2020-06-02 to 2020-06-21. The updated journal method generally achieves the lowest errors on most days, though the conference method and `NestDAU` surpass it between 2020-06-10 and 2020-06-14.

Figures 4.15 and 4.16 illustrate the performance of the five competing methods on the hourly windmill dataset. The updated journal method generally achieves the lowest error metrics across most time intervals, although `NestDAU` surpasses it at the 6th and 8th time points.

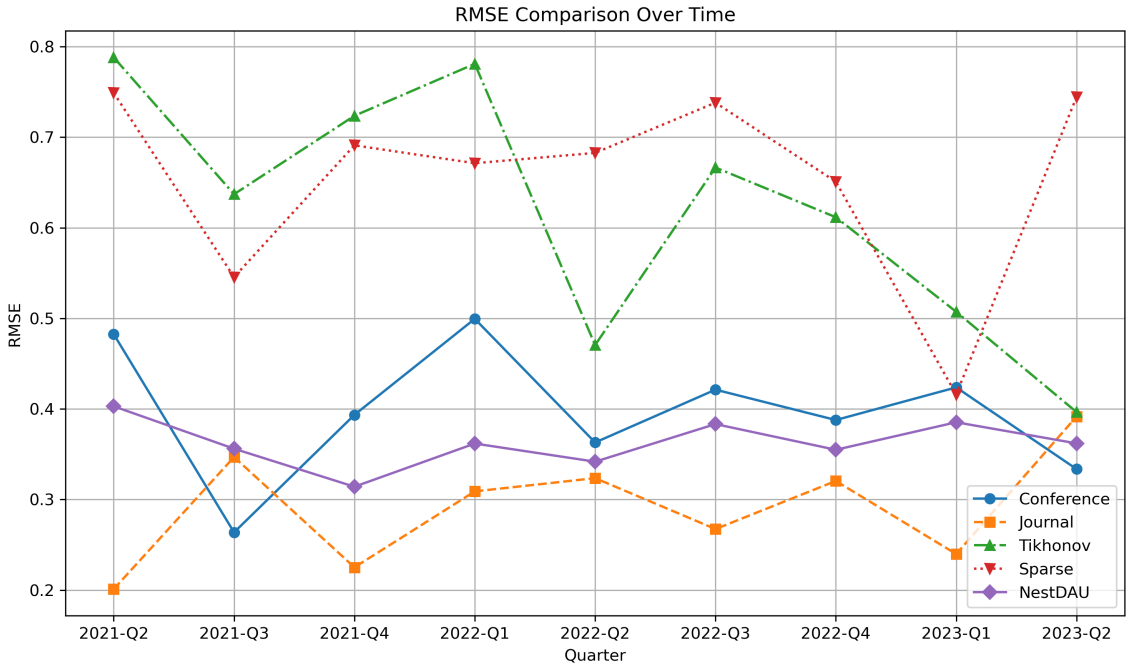

Figure 4.11: RMSE Comparison of Five Interpolation Methods by Quarter (50% Missing Rate) - Farmland Sales Dataset

Collectively, the results across the three datasets suggest that the updated journal method consistently delivers either superior or comparable performance compared to alternative approaches.

Finally, Table 4.5 summarizes the average RMSE interpolation error for all methods across all datasets at a 50% missing rate. The averages were computed over 10 quarters for the farmland sales dataset, 20 days for the temperature dataset, and 12 hours for the windmill dataset. Overall, the updated journal method exhibits the lowest average error, highlighting its effectiveness in adapting to diverse data characteristics and temporal scales while maintaining high interpolation accuracy.

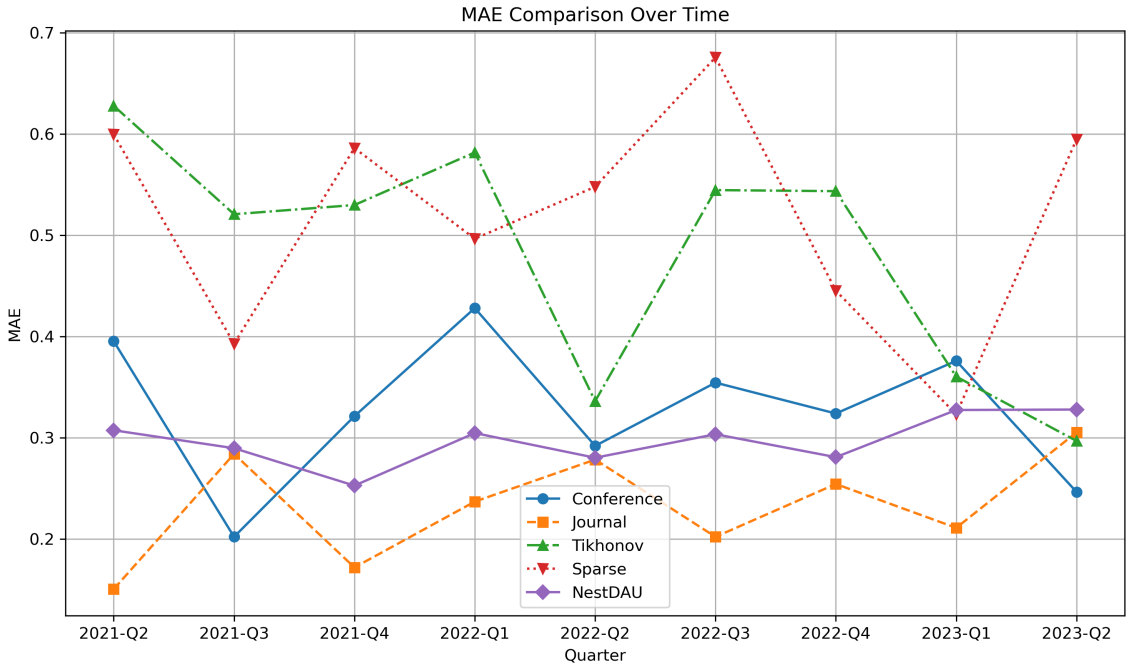

Figure 4.12: MAE Comparison of Five Interpolation Methods by Quarter (50% Missing Rate) - Farmland Sales Dataset

Table 4.5: Comparison of Average RMSE Errors for Graph Signal Interpolation Methods across Datasets (Lowest Errors in Bold).

Dataset	Conference	Journal	Tikhonov	Sparse	NestDAU
Land Sales	0.4538	**0.3081**	0.5810	0.6200	0.4023
Weather	0.3495	**0.2339**	0.3794	0.5003	0.3862
Windmill	0.4061	**0.2237**	0.5787	0.6542	0.3887

E.4 Ablation Study

To assess the impact of key hyperparameters, ablation studies were conducted on the Quarterly Farmland Sales Dataset, reporting the average root mean square error

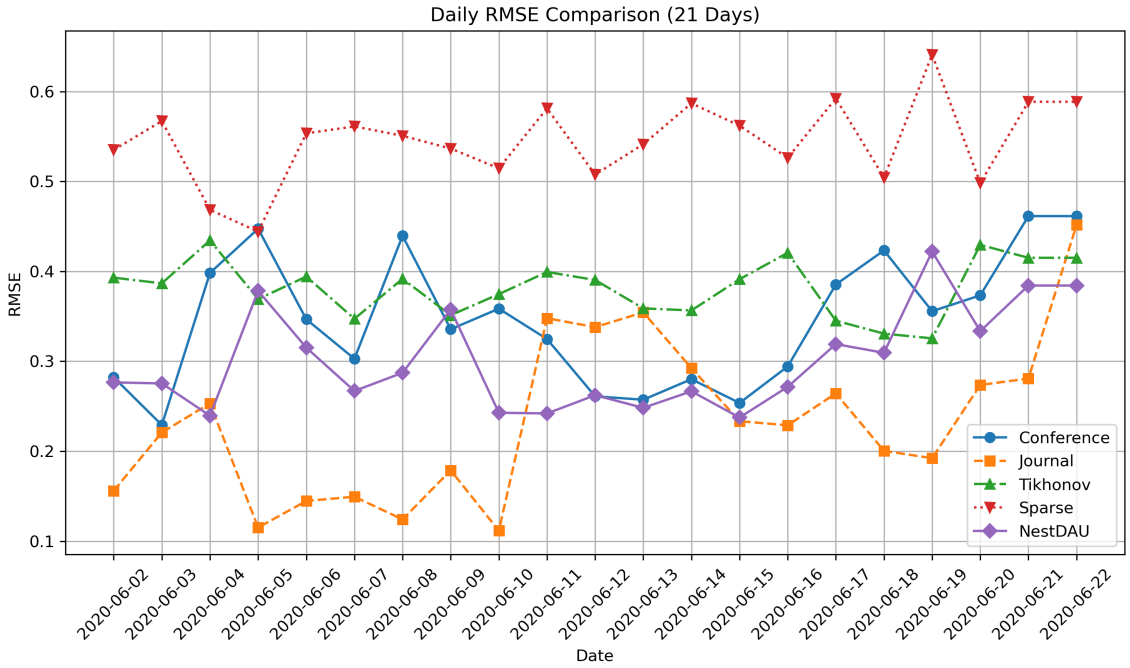

Figure 4.13: RMSE Comparison of Five Interpolation Methods by Quarter ($\sim 35\%$ Missing Rate) -Temperature Data

(RMSE) over eight quarters. Unless otherwise specified, the default configuration employs a learning rate of 0.01, 400 training epochs, 4 unrolled layers in the interpolation network, and 8 hidden layers in the graph convolutional network (GCN) module.

The first experiment evaluated the benefit of updating the graph at each time step (dynamic approach) compared to using a single static graph learned from the second quarter of 2021. The static graph was trained on all available observations from the first quarter and then applied unchanged for interpolation across all eight quarters. In contrast, the dynamic approach updates the adjacency matrix at each quarter using the unrolled algorithm. The static-graph approach yielded an average RMSE of 0.3999, whereas the dynamic-graph approach reduced the RMSE to 0.3081, demonstrating the advantage of modeling time-varying connectivity.

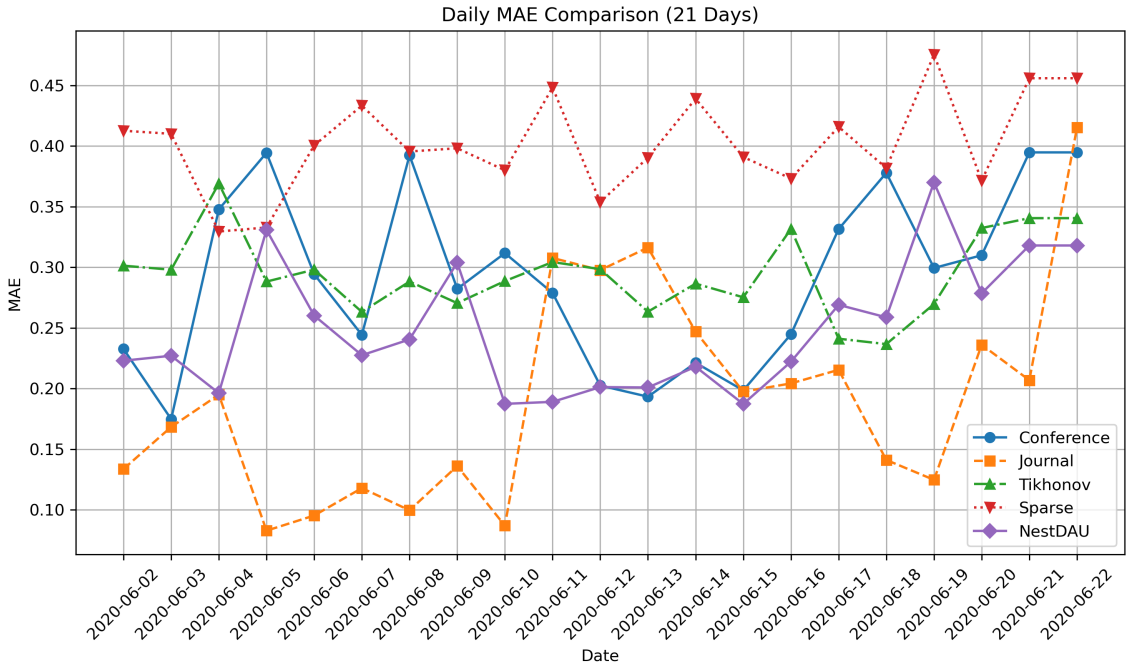

Figure 4.14: MAE Comparison of Five Interpolation Methods by Quarter ($\sim$ 35% Missing Rate) -Temperature Data

The second experiment varied the learning rate while keeping other parameters fixed. Table 4.6 presents the RMSE after 400 epochs. A learning rate of 0.01 achieved the lowest error, while both smaller and larger rates resulted in slower convergence or slight over- or under-fitting.

Table 4.6: Mean RMSE for Different Learning Rates

Learning Rate (lr)	Mean RMSE
0.0001	0.3512
0.01	**0.3081**
0.1	0.3375
2	0.4193

The third experiment varied the number of unrolled layers L while holding other settings constant. Table 4.7 reports both the RMSE and the average wall-clock time

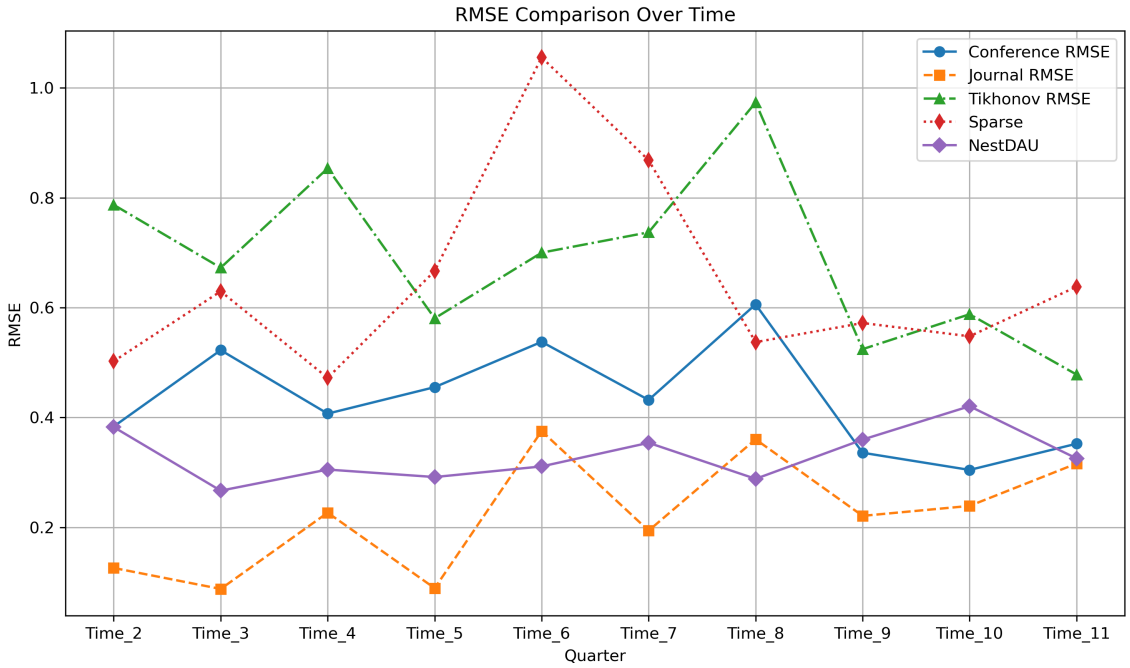

Figure 4.15: RMSE Comparison of Five Interpolation Methods by Quarter (50% Missing Rate) - Windmill Data

per run. Although $L = 10$ layers produced the lowest RMSE (0.3062), it approximately doubled the runtime compared to $L = 4$. Increasing to $L = 20$ resulted in a slight RMSE increase, likely due to overfitting. Thus, $L = 4$ offers the optimal balance between accuracy and computational efficiency.

Table 4.7: Performance for Different Numbers of Unrolled Layers (400 epochs)

Unrolled Layers	Time (s)	RMSE
2	15.96	0.3517
4	27.71	0.3081
10	62.13	**0.3062**
20	114.98	0.3349

The final experiment examined the depth of the GCN feature extractor by varying the number of hidden layers H. Table 4.8 presents the RMSE results. Shallower

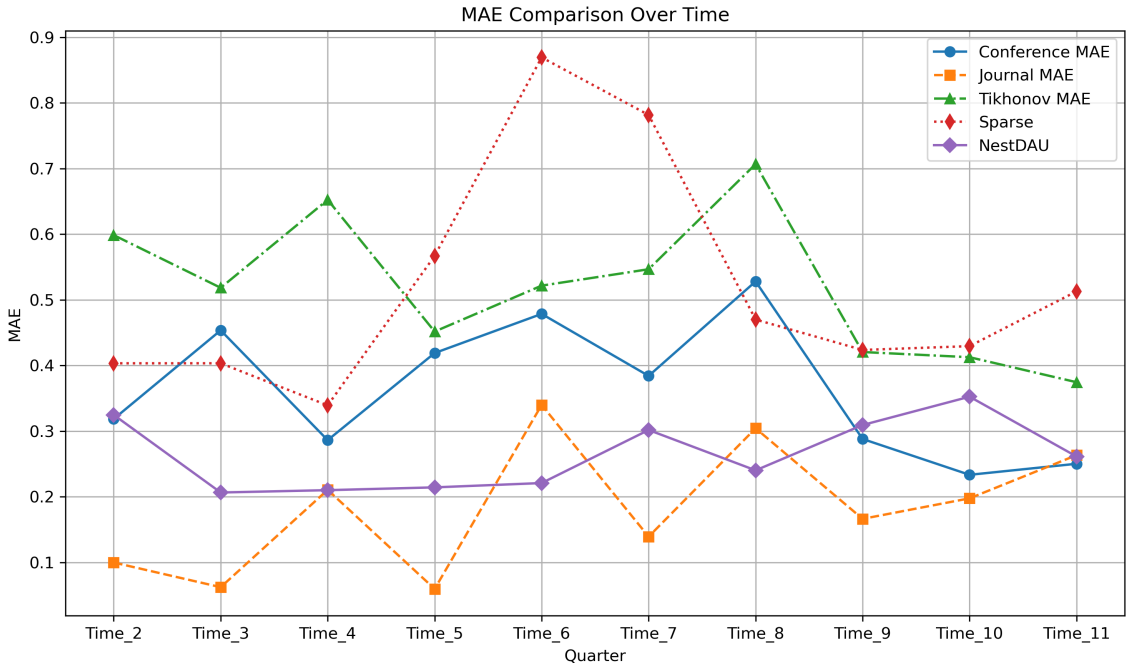

Figure 4.16: MAE Comparison of Five Interpolation Methods by Quarter (50% Missing Rate) - Windmill Data

networks ($H = 4$) exhibited underfitting, while deeper networks ($H \geq 16$) showed increased error, likely due to overfitting. A depth of $H = 8$ achieves the best balance between model capacity and generalization.

Table 4.8: RMSE for Different Numbers of GCN Hidden Layers

GCN Hidden Layers	RMSE
4	0.3486
8	**0.3081**
16	0.3372
32	0.3973

Chapter 5

Methodology: Fast Graph Sparsification

This chapter presents a fast graph sparsification technique developed to enhance the efficiency of graph-based data processing. The proposed method constructs a sparse graph from a complete graph by utilizing the Fiedler number—the second smallest eigenvalue of the graph Laplacian—to inform edge removal decisions.

A Fiedler Number

As discussed in Section E of Chapter 2, various graph sparsification techniques exist, each with its own disadvantages. An alternative way to quantify a graph's connectedness is through the second smallest eigenvalue λ_2 of a combinatorial graph Laplacian $\mathbf{L}$. This metric is known as the *Fiedler Number* or algebraic connectivity. Note that the first eigenvalue λ_1 of $\mathbf{L}$ is always 0, while λ_2 provides insight into the graph's connectivity. The second eigen-pair $(\lambda_2, \mathbf{v}_2)$ of $\mathbf{L}$ can be efficiently computed using algorithms such as the *Locally Optimal Block Preconditioned Conjugate Gradient* (LOBPCG) [103] in linear time $\mathcal{O}(N)$, assuming a sparse, symmetric and PD matrix (One can compute eigen-pair $(\mu_2, \mathbf{v}_2)$ for a shifted PD matrix $\mathbf{L} + \delta\mathbf{I}$ and then shift down to obtain $(\lambda_2 = \mu_2 - \delta, \mathbf{v}_2)$ for $\mathbf{L}$). For a complete graph $\mathcal{G}$, one can iteratively

remove edges that lead to the smallest change in λ_2. This iterative process is known as the *Fiedler method*.

A.1 Comparison of Graph Sparsification Methods

This section compares the performance of several graph sparsification schemes described in Section E in Chapter 2, based on the cost function values obtained during GCN training.

The experimental setup is detailed in Section C. Due to the high complexity associated with APSP, graphs are constructed using data from 11 counties in Iowa. The resulting graphs $\mathcal{G}^*$ together with 11 features (from the target year) related to crop yields were inputted (such as *normalized difference vegetation index* (NDVI) and *enhanced vegetation index* (EVI) [113]) for GCN training.

Figure 5.1 presents the mean squared error (MSE) outcomes for six sparsification methods evaluated across four different edge counts. The results clearly indicate that the Fiedler sparsification method consistently achieves the best prediction accuracy across all levels of graph sparsity.

B Fast Graph Sparsification

Although the Fiedler method yields high-quality sparsification, its computational cost is substantial.

For each candidate edge removal from a graph $\mathcal{G}$ with M edges, the second eigenvalue λ_2 must be computed using LOBPCG for the resulting Laplacian $\mathbf{L}$ corresponding to each one of M candidate edges with a complexity of $\mathcal{O}(N)$. Consequently, removing a single edge requires $\mathcal{O}(MN)$ operations. To remove K edges, the total

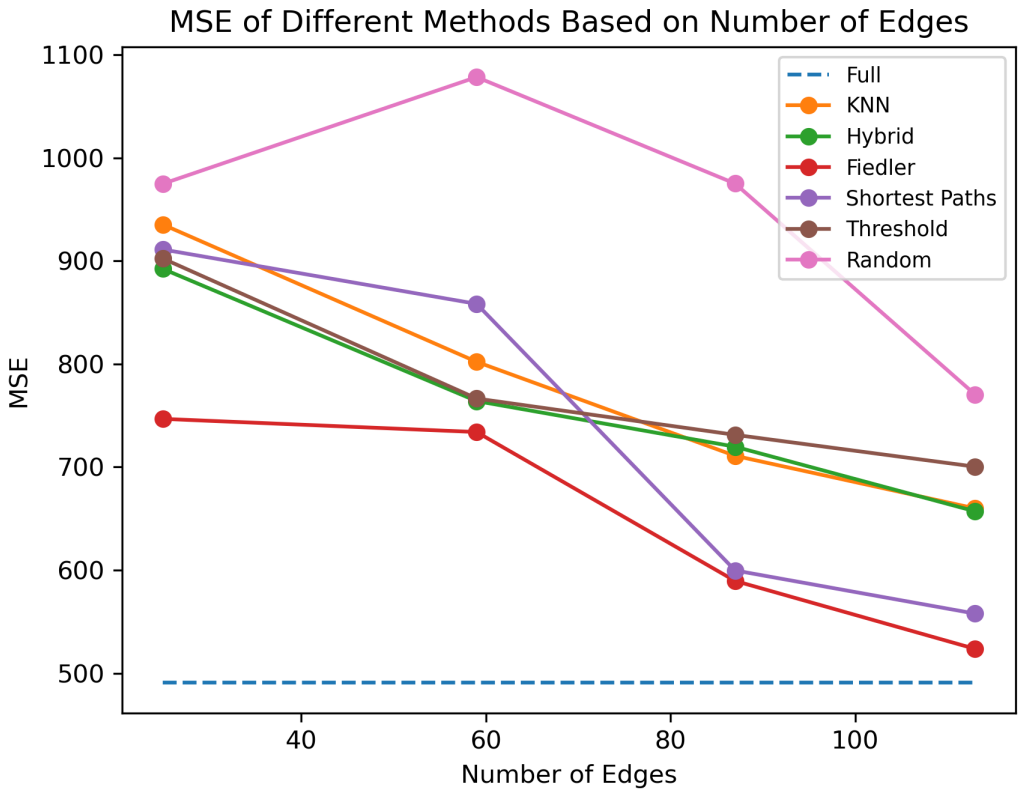

Figure 5.1: Average Crop Yield Prediction Error (MSE) Over 20 Runs: Comparison of Sparsification Methods to Complete Graph with 11 Nodes.

complexity is $\mathcal{O}(KMN)$.

To reduce this computational burden, a fast approximation scheme is introduced.

Consider a modified Laplacian matrix $\tilde{\mathbf{L}} = \mathbf{L} + \mathbf{E}^{m,n}$ with an edge (m, n) removed from original graph $\mathcal{G}$ specified by Laplacian $\mathbf{L}$, where *perturbation matrix* $\mathbf{E}^{m,n} \in \mathbb{R}^{N \times N}$ is defined as

$$E_{i,j}^{m,n} = \begin{cases} w_{m,n} & \text{if } \; i = m, j = n \;\; \text{or} \;\; i = n, j = m \\ -w_{m,n} & \text{if } \; i = j = m \;\; \text{or} \;\; i = j = n \\ 0 & \text{o.w.} \end{cases} . \tag{5.1}$$

Denote by $(\lambda_2, \mathbf{v}_2)$ the second eigen-pair for matrix $\mathbf{L}$, *i.e.*, $\mathbf{L}\mathbf{v}_2 = \lambda_2 \mathbf{v}_2$. A known eigenvalue perturbation theorem [114] states that there exists an eigenvalue $\tilde{\lambda}$ of $\tilde{\mathbf{L}}$ in the range:

$$\tilde{\lambda} \in [\; \lambda_2 - \|\mathbf{E}^{m,n}\mathbf{v}_2\|_2, \;\; \lambda_2 + \|\mathbf{E}^{m,n}\mathbf{v}_2\|_2 \;] \, . \tag{5.2}$$

Recall that the Fiedler method removes an edge whose removal induces the minimum change in λ_2. Thus, from (5.2), a criterion can be derived to choose an edge (m, n) for removal so that $\tilde{\lambda}$ differs from λ_2 minimally.

Denote by $\mathcal{E}$ the edge set in graph $\mathcal{G}$ specified by Laplacian $\mathbf{L}$. An edge $(m^*, n^*) \in \mathcal{E}$ is chosen such that

$$(m^*, n^*) = \arg \min_{(m,n) \in \mathcal{E}} \; \|\mathbf{E}^{m,n}\mathbf{v}_2\|_2 \, . \tag{5.3}$$

The complexity of computing $\|\mathbf{E}^{m,n}\mathbf{v}_2\|_2$ for a given $\mathbf{E}^{m,n}$ is $\mathcal{O}(1)$; by (5.1), there are only four non-zero entries in $\mathbf{E}^{m,n}$. Thus, the complexity of computing (m^*, n^*) in (5.3) is $\mathcal{O}(M)$. If K edges are removed, then the complexity is $\mathcal{O}(KM)$.

Figure 5.2 compares the edges removed by the full Fiedler method (48 out of 121 edges in a complete graph with 11 nodes) against those identified by the proposed fast approximation. Both methods predominantly target edges with smaller weights,

demonstrating a close correspondence in the selected edge set.

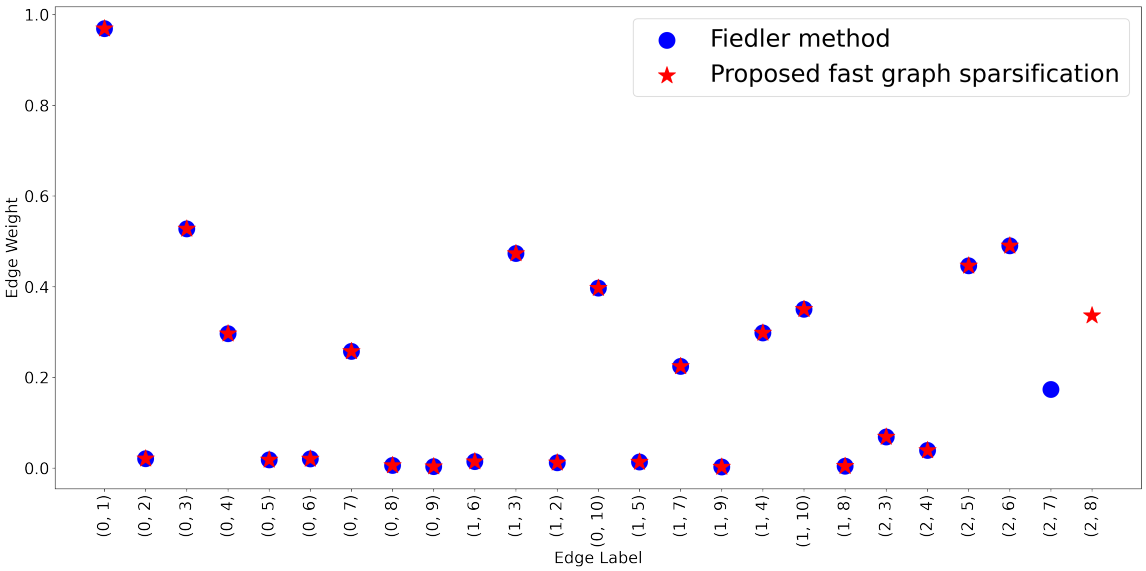

Figure 5.2: Comparison of Edges Removed: Fiedler Method vs. Proposed Fast Approximation.

C Experiment Setup and Results

C.1 Setup

To evaluate the effectiveness of the proposed graph sparsification method, an experiment is conducted using the county-level corn yield data described in Section E.1.c. The data from 2010 to 2020 are employed to predict yields in 2021, focusing on 99 counties in Iowa.

A full graph is constructed based on the average yield from the previous four years and the counties' locations. Specifically, weight $w_{i,j}$ of edge (i, j) connecting counties

i and j is computed as

$$w_{i,j} = \exp\left(-\frac{\|\mathbf{l}_i - \mathbf{l}_j\|_2^2}{\sigma_l^2} - \frac{|y_i - y_j|^2}{\sigma_y^2}\right) \tag{5.4}$$

$$\lambda_2 = \frac{\mathbf{v}_2^\top \mathbf{L}\mathbf{v}_2}{\mathbf{v}_2^\top \mathbf{v}_2} = \sum_{i,j=0}^{N} w_{i,j}(\mathbf{v}_{2i} - \mathbf{v}_{2j})^2 \tag{5.5}$$

where $\mathbf{l}_i \in \mathbb{R}^2$ represents the 2D geographic coordinates and $\mathbf{y}_i$ is the *crop yield mean* of county i. The parameters are set to $\sigma_l = 5.625 \times 10^9$ and $\sigma_y = 0.5$ for optimal performance.

Subsequently, five different sparsification methods are applied to the full graph: kNN, ϵ-thresholding, hybrid kNN / ϵ-thresholding, random pruning, and our proposed fast Fiedler method. Each method is evaluated across five different sparsity levels (*i.e.*, varying numbers of graph edges).

The sparsified graphs denoted as $\mathcal{G}^*$, are then combined with 41 yield-related features from the target year—such as precipitation, soil composition, and sunlight conditions—to form the input for a yield prediction network. Figure 5.3 presents a schematic illustration of the prediction algorithm.

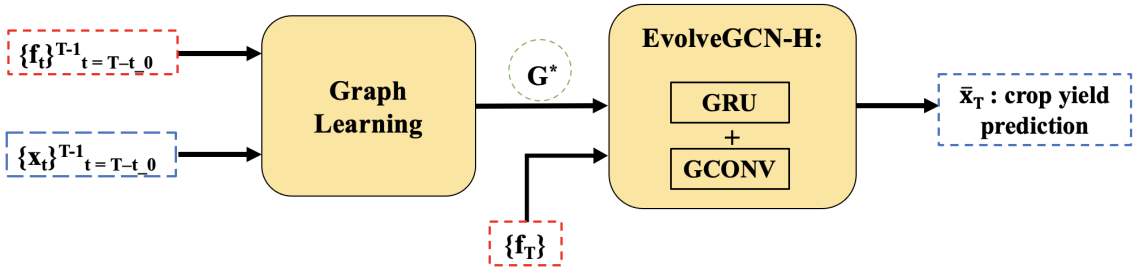

Figure 5.3: Crop Yield Prediction Algorithm Illustration.

A GCN model, specifically EvolveGCN-H [115], which integrates a simple GCN [1] with a gated recurrent unit (GRU) [116], is trained for yield prediction. To mitigate

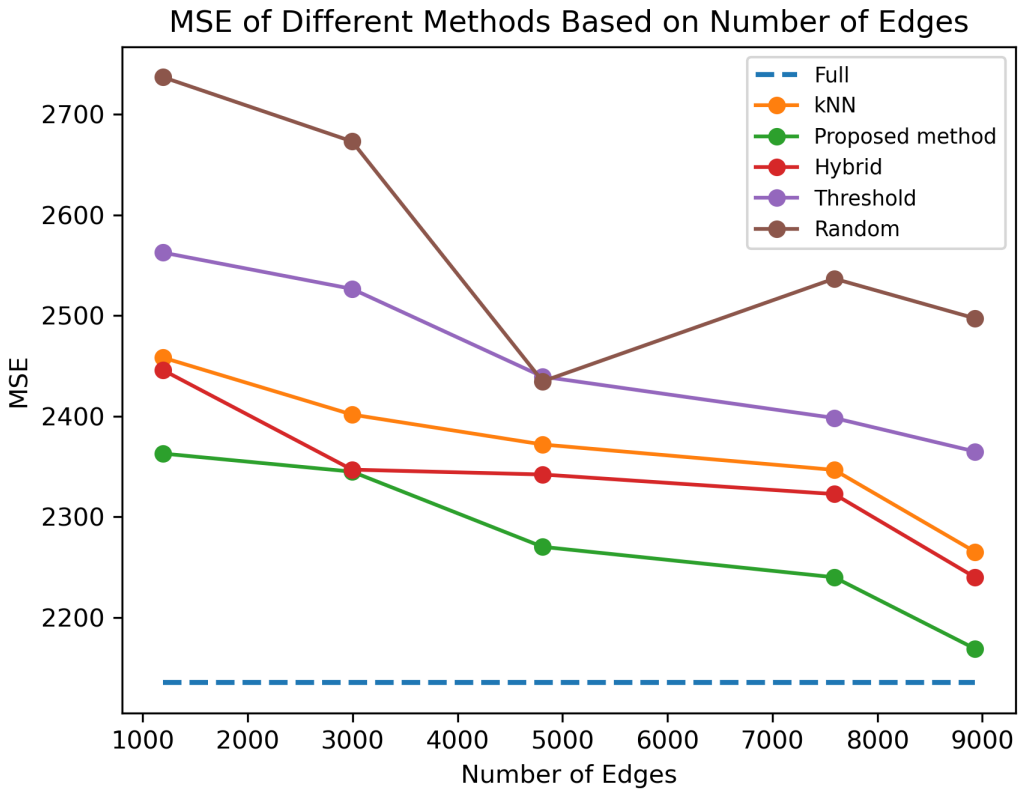

Figure 5.4: Crop Yield Prediction: Average MSE over 20 Runs for Five Graph Sparsification Methods versus the Full Graph (99 Nodes).

overfitting, only one layer of EvolveGCN-H, along with a single linear layer, is employed. The average error is calculated over 20 runs, with each run consisting of 1000 training epochs and early stopping activated using patience of 20 epochs. A learning rate of 0.01 is maintained throughout the training process.

C.2 Results

Figure 5.4 presents the mean squared error (MSE) results for all five sparsification methods across various sparsity levels. The proposed fast Fiedler method consistently

achieves the lowest prediction errors for nearly all edge counts. In general, the MSE for most methods decreases as the number of edges increases—indicating that sparser graphs with additional edges carry more informative weights and thus approximate the performance of the full graph—except in the case of random sparsification.

Table 5.1 further highlights the computational advantages of sparse graphs during GCN training. In this experiment, sparse graphs with 1191 edges were used. The results show that graph sparsification enables faster GCN convergence, requiring fewer training epochs and reducing overall training time, a benefit that is particularly significant when working with large-scale graphs.

Table 5.1: Average Training Time and Epochs over 20 Runs for Four Graph Sparsification Methods versus the Full Graph (99 Nodes).

Graph	Full	kNN	Hybrid	Fast Fiedler	ϵ-Threshold
Time	52.70s	17.33s	18.08s	15.01s	15.70s
Epochs	765	256	383	258	304

Chapter 6

Conclusion and Future Work

This dissertation advances graph learning techniques for irregular-structured data by addressing two complementary challenges: modeling slowly time-varying graphs and sparsifying dense graph kernels. These approaches tackle critical issues in graph signal processing, namely dynamic adaptation to evolving data relationships and the mitigation of computational burdens associated with dense graph representations.

A Conclusion: Modeling Slowly Time-Varying Graphs

To address scenarios where pairwise similarities between nodes evolve over time, this dissertation proposes a low-rank model, wherein the difference between consecutive adjacency matrices $\mathbf{W}^{(2)} - \mathbf{W}^{(1)}$ is assumed to be a low-rank matrix. The proposed algorithm formulates a joint signal interpolation and graph learning problem, alternately optimizing the signal $\mathbf{x}_2$ and the adjacency matrix $\mathbf{W}^{(2)}$ at time $t = 2$, given $\mathbf{x}_1$ and $\mathbf{W}^{(1)}$ at $t = 1$. When $\mathbf{x}_2$ is fixed, a proximal gradient descent (PGD) algorithm computes $\mathbf{W}^{(2)}$. Under an eigenvector assumption, the proximal mapping for the rank term $\Gamma(\mathbf{W}^{(2)} - \mathbf{W}^{(1)})$ can be optimally computed using a full greedy algorithm in polynomial time. A more recent version of this algorithm introduces a

fast linear-time greedy approach that computes only B extreme eigenvectors of $\mathbf{W}^{(1)}$, where the Perron vector $\mathbf{v}_1$, corresponding to the largest eigenvalue, captures relative node importance. Consequently, the rank-1 matrix $\mathbf{v}_1\mathbf{v}_1^\top$ models interactions between significant nodes. The algorithm is unrolled into layers to form a lightweight feed-forward network for data-driven parameter learning. Experimental results demonstrate that this joint optimization outperforms existing time-varying graph models in signal interpolation tasks.

B Conclusion: Fast Graph Sparsification

Graph convolutional networks (GCNs) excel in tasks such as crop yield prediction by leveraging historical data and diverse features. However, dense graph kernels can incur substantial computational overhead, increasing training and inference costs. This dissertation introduces a fast graph sparsification method to address this challenge while preserving essential structural information for accurate predictions. The proposed approach employs a sparsification algorithm guided by the Fiedler number—the second smallest eigenvalue of the graph Laplacian. In each iteration, the algorithm selects an edge for removal that minimally alters this eigenvalue, thereby maintaining the graph's core connectivity. A fast implementation leverages an eigenvalue perturbation theorem, reducing the complexity of edge selection per iteration to $\mathcal{O}(M)$ for M edges. Experimental results confirm that this Fiedler-based sparsification method significantly reduces GCN training time while maintaining robust crop yield prediction performance.

C Overall Conclusion and Future Directions

Collectively, these methodologies establish a comprehensive framework for graph learning across diverse structured datasets. The dynamic graph learning approach enables adaptive modeling of evolving node similarities, while the fast graph sparsification technique reduces computational costs without sacrificing critical connectivity. This dual framework reflects the dissertation's broader objective: to develop efficient, interpretable, and robust graph learning algorithms capable of adapting to varied data characteristics and temporal scales.

The contributions of this dissertation enhance the accuracy of applications such as crop yield prediction, temperature forecasting, and energy output estimation, while providing a foundation for future research. Potential extensions include exploring learning frameworks for more complex graph structures, such as time-varying directed graphs, signed graphs, and hypergraphs.

Directed graph learning presents opportunities to model asymmetrical relationships, such as those observed in rumor propagation on social networks [5, 117]. However, this approach poses challenges, as non-symmetric adjacency or Laplacian matrices may not be eigen-diagonalizable, and inferring directionality often requires additional contextual data.

Hypergraphs offer another promising direction, as they can capture high-order interactions where each hyberedge connect more than two nodes. Recent work in hypergraph signal processing (HGSP) [118] employs tensor representations to extend traditional graph signal processing concepts into a hypergraph Fourier space, enabling novel sampling theories and robust filter designs. This framework is particularly

suited for applications in the Internet of Things, where data exhibits complex multi-way relationships beyond pairwise interactions.

Signed graphs, incorporating both positive and negative edge weights, allow modeling of anti-correlations alongside positive similarities [14, 60]. While conventional graph signal processing often focuses on positive graphs, many real-world datasets exhibit significant negative relationships, necessitating specialized techniques to capture the interplay between cooperative and antagonistic interactions.

These extensions—directed graphs, hypergraphs, and signed graphs—represent compelling avenues for future research. They broaden the scope of graph learning techniques and deepen the understanding of complex data structures. Additionally, future work could explore integrating unsupervised parameter learning or advanced deep learning architectures, such as transformers or diffusion models, while preserving the interpretability of the current framework. Ultimately, this dissertation lays a robust foundation for developing efficient and adaptable graph learning algorithms tailored to real-world applications.

Bibliography

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

[2] 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.

[3] G. B. Giannakis, Y. Shen, and G. V. Karanikolas, "Topology identification and learning over graphs: Accounting for nonlinearities and dynamics," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 787–807, 2018.

[4] I. Jabloński, "Graph signal processing in applications to sensor networks, smart grids, and smart cities," *IEEE Sensors Journal*, vol. 17, no. 23, pp. 7659–7666, 2017.

[5] I. Anger and C. Kittl, "Measuring influence on twitter," in *Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '11, (New York, NY, USA), Association for Computing Machinery, 2011.

[6] H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos, "Colibri: Fast mining of large static and dynamic graphs," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, (New York, NY, USA), p. 686–694, Association for Computing Machinery, 2008.

[7] C. Hu, L. Cheng, J. Sepulcre, K. Johnson, G. Fakhri, Y. Lu, and Q. Li, "A spectral graph regression model for learning brain connectivity of alzheimer's disease," *PLOS ONE*, vol. 10, p. e0128136, 05 2015.

[8] K. J. Friston, "Functional and effective connectivity in neuroimaging: A synthesis," *Human Brain Mapping*, vol. 2, no. 1-2, pp. 56–78, 1994.

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

[10] G. Cheung, E. Magli, Y. Tanaka, and M. Ng, "Graph spectral image processing," in *Proceedings of the IEEE*, vol. 106, no.5, pp. 907–930, May 2018.

[11] A. Ortega, *Introduction to Graph Signal Processing*. Cambridge University Press, 2022.

[12] C. Yang, G. Cheung, and W. Hu, "Signed graph metric learning via Gershgorin disc perfect alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 7219–7234, 2022.

[13] C. Dinesh, G. Cheung, and I. V. Bajic, "Point cloud sampling via graph balancing and Gershgorin disc alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2022.

[14] C. Dinesh, G. Cheung, S. Bagheri, and I. V. Bajic, "Efficient signed graph sampling via balancing Gershgorin disc perfect alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–18, 2024.

[15] W. Hu, X. Li, G. Cheung, and O. Au, "Depth map denoising using graph-based transform and group sparsity," in *IEEE International Workshop on Multimedia Signal Processing*, (Pula, Italy), October 2013.

[16] J. Pang, G. Cheung, A. Ortega, and O. C. Au, "Optimal graph Laplacian regularization for natural image denoising," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, (Brisbane, Australia), April 2015.

[17] J. Pang and G. Cheung, "Graph Laplacian regularization for inverse imaging: Analysis in the continuous domain," in *IEEE Transactions on Image Processing*, vol. 26, no.4, pp. 1770–1785, April 2017.

[18] W.-t. Su, G. Cheung, R. Wildes, and C.-W. Lin, "Graph neural net using analytical graph filters and topology optimization for image denoising," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8464–8468, 2020.

[19] X. Liu, G. Cheung, X. Wu, and D. Zhao, "Random walk graph laplacian based smoothness prior for soft decoding of JPEG images," in *IEEE Transactions on Image Processing*, vol. 26, no.2, pp. 509–524, February 2017.

[20] W. Hu, G. Cheung, and A. Ortega, "Intra-prediction and generalized graph Fourier transform for image coding," in *IEEE Signal Processing Letters*, vol. 22, no.11, pp. 1913–1917, November 2015.

[21] W. Hu, G. Cheung, A. Ortega, and O. Au, "Multi-resolution graph Fourier transform for compression of piecewise smooth images," in *IEEE Transactions on Image Processing*, vol. 24, no.1, pp. 419–433, January 2015.

[22] J. Zeng, G. Cheung, Y. Chao, I. Blanes, J. Serra-Sagristà, and A. Ortega, "Hyperspectral image coding using graph wavelets," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 1672–1676, 2017.

[23] X. Su, M. Rizkallah, T. Maugey, and C. Guillemot, "Graph-based light fields representation and coding using geometry information," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 4023–4027, 2017.

[24] Y. Bai, G. Cheung, X. Liu, and W. Gao, "Graph-based blind image deblurring from a single photograph," in *IEEE Transactions on Image Processing*, vol. 28, no.3, pp. 1404–1418, March 2019.

[25] F. Chen, G. Cheung, and X. Zhang, "Manifold graph signal restoration using gradient graph Laplacian regularizer," *IEEE Transactions on Signal Processing*, vol. 72, pp. 744–761, 2024.

[26] C. Dinesh, G. Cheung, and I. V. Bajić, "Point cloud denoising via feature graph Laplacian regularization," *IEEE Transactions on Image Processing*, vol. 29, pp. 4143–4158, 2020.

[27] F. Wang, Y. Wang, G. Cheung, and C. Yang, "Graph sampling for matrix completion using recurrent Gershgorin disc shift," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2814–2829, 2020.

[28] W. Cai, G. Cheung, T. Kwon, and S.-J. Lee, "Optimized frame structure for interactive light field streaming with cooperative cache," in *IEEE International Conference on Multimedia and Expo*, (Barcelona, Spain), July 2011.

[29] A. K. Prasad, L. B. Chai, R. P. Singh, and M. C. Kafatos, "Crop yield estimation model for iowa using remote sensing and surface parameters," *International Journal of Applied Earth Observation and Geoinformation*, vol. 8, pp. 26–33, 2006.

[30] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?," *J. ACM*, vol. 58, jun 2011.

[31] K. Yamada, Y. Tanaka, and A. Ortega, "Time-varying graph learning based on sparseness of temporal variation," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5411–5415, 2019.

[32] V. Kalofolias, A. Loukas, D. Thanou, and P. Frossard, "Learning time varying graphs," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2826–2830, 2017.

[33] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Incremental singular value decomposition algorithms for highly scalable recommender systems," 2002.

[34] S. Bagheri, G. Cheung, T. Eadie, and A. Ortega, "Joint signal interpolation / time-varying graph estimation via smoothness and low-rank priors," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 9646–9650, 2024.

[35] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," tech. rep., USA, 1994.

[36] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[37] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," *CoRR*, vol. abs/2007.02133, 2020.

[38] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," *CoRR*, vol. abs/2002.05287, 2020.

[39] F. R. K. Chung, *Spectral Graph Theory*. Providence, RI: American Mathematical Society, 1997.

[40] W. Hu, X. Gao, G. Cheung, and Z. Guo, "Feature graph learning for 3D point cloud denoising," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2841–2856, 2020.

[41] F. Chen, G. Cheung, and X. Zhang, "Fast & robust image interpolation using gradient graph Laplacian regularizer," in *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 1964–1968, 2021.

[42] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. 01 2009.

[43] O. Banerjee and L. Ghaoui, "Model selection through sparse max likelihood estimation," *Journal of Machine Learning Research*, vol. 9, 08 2007.

[44] N. Meinshausen and P. Bühlmann, "High-dimensional graphs and variable selection with the Lasso," *The Annals of Statistics*, vol. 34, no. 3, pp. 1436 – 1462, 2006.

[45] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," in *Biostatistics*, vol. 9, no.3, pp. 432–441, 2008.

[46] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 2005.

[47] H. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under Laplacian and structural constraints," in *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no.6, pp. 825–841, July 2017.

[48] S. Kumar, J. Ying, J. V. de Miranda Cardoso, and D. Palomar, "Structured graph learning via laplacian spectral constraints," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[49] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[50] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM Journal on Computing*, vol. 40, no. 4, pp. 981–1025, 2011.

[51] 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," in *IEEE Signal Processing Magazine*, vol. 30, no.3, pp. 83–98, May 2013.

[52] M. Vetterli, J. Kovavcević, and V. Goyal, *Foundations of Signal Processing*. Cambridge University Press, 2014.

[53] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," in *Advances in Neural Information Processing Systems* (S. Thrun, L. Saul, and B. Schölkopf, eds.), vol. 16, MIT Press, 2003.

[54] A. Globerson and S. Roweis, "Metric learning by collapsing classes," in *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), vol. 18, MIT Press, 2005.

[55] P. Moutafis, M. Leng, and I. A. Kakadiaris, "An overview and empirical comparison of distance metric learning methods," *IEEE Transactions on Cybernetics*, vol. 47, no. 3, pp. 612–625, 2017.

[56] G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang, "An efficient sparse metric learning in high-dimensional space via l1-penalized log-determinant regularization," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), p. 841–848, Association for Computing Machinery, 2009.

[57] C. Yang, G. Cheung, and W. Hu, "Signed graph metric learning via Gershgorin disc perfect alignment," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2021.

[58] R. S. Varga, *Gershgorin and his circles*. Springer, 2004.

[59] W.-T. Su, G. Cheung, and C.-W. Lin, "Robust graph-based image classifier learning with negative edge weights," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 397–402, 2017.

[60] G. Cheung, W.-T. Su, Y. Mao, and C.-W. Lin, "Robust semisupervised graph classifier learning with negative edge weights," in *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no.4, pp. 712–726, December 2018.

[61] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from filtered signals: Graph system and diffusion kernel identification," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 2, pp. 360–374, 2019.

[62] C. Yang, G. Cheung, and W. Hu, "Graph metric learning via gershgorin disc alignment," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5530–5534, 2020.

[63] 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.

[64] J. Sun, L. Di, Z. Sun, Y. Shen, and Z. Lai, "County-level soybean yield prediction using deep cnn-lstm model," *Sensors*, vol. 19, no. 20, 2019.

[65] M. Shahhosseini, G. Hu, S. Khaki, and S. V. Archontoulis, "Corn yield prediction with ensemble CNN-DNN," *Frontiers in Plant Science*, vol. 12, aug 2021.

[66] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proceedings of the IEEE International Conference on Computer Vision*, (Bombay, India), 1998.

[67] A. Gadde, S. K. Narang, and A. Ortega, "Bilateral filter: Graph spectral interpretation and extensions," in *IEEE International Conference on Image Processing*, (Melbourne, Australia), September 2013.

[68] J. Pang and G. Cheung, "Graph laplacian regularization for image denoising: Analysis in the continuous domain," *IEEE Transactions on Image Processing*, vol. 26, p. 1770–1785, Apr 2017.

[69] A. Natali, E. Isufi, M. Coutino, and G. Leus, "Learning time-varying graphs from online data," 2022.

[70] J. L. Ying, X. Han, R. Zhou, X. Wang, and H. C. So, "Network topology inference with sparsity and laplacian constraints," 2023.

[71] C. Chen and H. Tong, "On the eigen-functions of dynamic graphs: Fast tracking and attribution algorithms: Eigen-functions of dynamic graphs," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 10, 06 2016.

[72] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ACM, nov 2017.

[73] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, p. 3634–3640, AAAI Press, 2018.

[74] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3848–3858, 2020.

[75] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," 2019.

[76] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107000, Jan. 2020.

[77] J. Li, Z. Han, H. Cheng, J. Su, P. Wang, J. Zhang, and L. Pan, "Predicting path failure in time-evolving graphs," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, (New York, NY, USA), p. 1279–1289, Association for Computing Machinery, 2019.

[78] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," 05 2018.

[79] C. Wang, Y. Qiu, D. Gao, and S. Scherer, "Lifelong graph learning," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13709–13718, 2022.

[80] F. Zhou and C. Cao, "Overcoming catastrophic forgetting in graph neural networks with experience replay," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4714–4722, 05 2021.

[81] S. Sun, Q. Sun, K. Zhou, and T. Lv, "Hierarchical attention prototypical networks for few-shot text classification," pp. 476–485, 01 2019.

[82] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, (New York, NY, USA), p. 891–900, Association for Computing Machinery, 2015.

[83] S. Chen, A. Sandryhaila, J. Moura, and J. Kovacevic, "Signal recovery on graphs: Variation minimization," in *IEEE Transactions on Signal Processing*, vol. 63, no.17, pp. 4609–4624, September 2015.

[84] C. Couprie, L. Grady, L. Najman, J.-C. Pesquet, and H. Talbot, "Dual constrained tv-based regularization on graphs," vol. 6, no. 3, 2013.

[85] P. Berger, G. Hannak, and G. Matz, "Graph signal recovery via primal-dual algorithms for total variation minimization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 842–855, 2017.

[86] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Processing Magazine*, vol. 38, no. 2, pp. 18–44, 2021.

[87] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th International Conference on International Conference*

on Machine Learning, ICML'10, (Madison, WI, USA), p. 399–406, Omnipress, 2010.

[88] Y. Yu, S. Buchanan, D. Pai, T. Chu, Z. Wu, S. Tong, B. Haeffele, and Y. Ma, "White-box transformers via sparse rate reduction," in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 9422–9457, Curran Associates, Inc., 2023.

[89] J. Zeng, J. Pang, W. Sun, and G. Cheung, "Deep graph laplacian regularization for robust denoising of real images," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1759–1768, 2019.

[90] H. Vu *et al.*, "Unrolling of deep graph total variation for image denoising," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2050–2054, 2021.

[91] R. Yoshida, K. Kodama, H. Vu, G. Cheung, and T. Hamamoto, "Unrolling graph total variation for light field image denoising," in *2022 IEEE International Conference on Image Processing (ICIP)*, pp. 2162–2166, 2022.

[92] M. Nagahama, K. Yamada, Y. Tanaka, S. H. Chan, and Y. C. Eldar, "Graph signal restoration using nested deep algorithm unrolling," *IEEE Transactions on Signal Processing*, vol. 70, pp. 3296–3311, 2022.

[93] T. T. Do, P. Eftekhar, S. A. Hosseini, G. Cheung, and P. Chou, "Interpretable lightweight transformer via unrolling of learned graph smoothness priors," 2024.

[94] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, p. 345, jun 1962.

[95] 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, p. 83–98, May 2013.

[96] G. Strang, "The discrete cosine transform," *SIAM Review*, vol. 41, no. 1, pp. 135–147, 1999.

[97] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *ArXiv*, vol. abs/1506.05163, 2015.

[98] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (Red Hook, NY, USA), p. 3844–3852, Curran Associates Inc., 2016.

[99] W. L. Tony Cai and X. Luo, "A constrained 1 minimization approach to sparse precision matrix estimation," *Journal of the American Statistical Association*, vol. 106, no. 494, pp. 594–607, 2011.

[100] O. C. Ibe, "Chapter 6 - functions of random variables," in *Fundamentals of Applied Probability and Random Processes (Second Edition)* (O. C. Ibe, ed.), pp. 185–223, Boston: Academic Press, second edition ed., 2014.

[101] N. Parikh and S. Boyd, "Proximal algorithms," in *Foundations and Trends in Optimization*, vol. 1, no.3, pp. 123–231, 2013.

[102] T. Hawkins, "Cauchy and the spectral theory of matrices," *Historia Mathematica*, vol. 2, no. 1, pp. 1–29, 1975.

[103] A. V. Knyazev, "Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517–541, 2001.

[104] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 2 ed., 2012.

[105] P. Bonacich, "Power and centrality: A family of measures," *American Journal of Sociology*, vol. 92, no. 5, pp. 1170–1182, 1987.

[106] P. L. Combettes and J.-C. Pesquet, *Proximal Splitting Methods in Signal Processing*, pp. 185–212. New York, NY: Springer New York, 2011.

[107] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, p. 4–24, Jan. 2021.

[108] M. Nagahama, K. Yamada, Y. Tanaka, S. H. Chan, and Y. C. Eldar, "Graph signal restoration using nested deep algorithm unrolling," *IEEE Transactions on Signal Processing*, vol. 70, pp. 3296–3311, 2022.

[109] S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg, "Plug-and-play priors for model based reconstruction," in *2013 IEEE Global Conference on Signal and Information Processing*, pp. 945–948, 2013.

[110] S. H. Chan, X. Wang, and O. A. Elgendy, "Plug-and-play admm for image restoration: Fixed-point convergence and applications," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 84–98, 2017.

[111] R. Ahmad, C. Bouman, G. Buzzard, S. Chan, S. Liu, E. Reehorst, and P. Schniter, "Plug-and-play methods for magnetic resonance imaging: Using denoisers for image recovery," *IEEE Signal Processing Magazine*, vol. 37, pp. 105–116, 01 2020.

[112] S. Shoushtari, J. Liu, E. P. Chandler, M. S. Asif, and U. S. Kamilov, "Prior mismatch and adaptation in pnp-admm with a nonconvex convergence analysis," in *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, JMLR.org, 2024.

[113] B. Matsushita, W. Yang, J. Chen, Y. Onda, and G. Qiu, "Sensitivity of the enhanced vegetation index (evi) and normalized difference vegetation index (ndvi) to topographic effects: A case study in high-density cypress forest," *Sensors*, vol. 7, no. 11, pp. 2636–2651, 2007.

[114] I. Ipsen and B. Nadler, "Refined perturbation bounds for eigenvalues of hermitian and non-hermitian matrices," *SIAM J. Matrix Analysis Applications*, vol. 31, pp. 40–53, 01 2009.

[115] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. B. Schardl, and C. E. Leiserson, "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[116] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, (Doha, Qatar), pp. 103–111, Association for Computational Linguistics, Oct. 2014.

[117] A. G. Marques, S. Segarra, and G. Mateos, "Signal processing on directed graphs: The role of edge directionality when processing and learning from network data," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 99–116, 2020.

[118] S. Zhang, Z. Ding, and S. Cui, "Introducing hypergraph signal processing: Theoretical foundation and practical applications," *IEEE Internet of Things Journal*, vol. 7, p. 639–660, Jan. 2020.