Graph Filters for Signal Processing and Machine Learning on Graphs

Elvin Isufi, Fernando Gama, David I Shuman, and Santiago Segarra

Overview Article

***Abstract*—Filters are fundamental in extracting information from data. For time series and image data that reside on Euclidean domains, filters are the crux of many signal processing and machine learning techniques, including convolutional neural networks. Increasingly, modern data also reside on networks and other irregular domains whose structure is better captured by a graph. To process and learn from such data, graph filters account for the structure of the underlying data domain. In this article, we provide a comprehensive overview of graph filters, including the different filtering categories, design strategies for each type, and trade-offs between different types of graph filters. We discuss how to extend graph filters into filter banks and graph neural networks to enhance the representational power; that is, to model a broader variety of signal classes, data patterns, and relationships. We also showcase the fundamental role of graph filters in signal processing and machine learning applications. Our aim is that this article provides a unifying framework for both beginner and experienced researchers, as well as a common understanding that promotes collaborations at the intersections of signal processing, machine learning, and application domains.**

***Index Terms*—Graph signal processing, graph machine learning, graph convolution, filter identification, graph filter banks and wavelets, graph neural networks, distributed processing, collaborative filtering, graph-based image processing, mesh processing, point clouds, topology identification, spectral clustering, matrix completion, graph Gaussian processes.**

I. Introduction

Filters are information processing architectures that preserve only the relevant content of the input for the task at hand. In signal processing (SP), filtering preserves specific spectral content of input signals and is a common building block in domains including audio, speech, radar, communication, and multimedia [1]. In machine learning (ML), filtering is used to extract relevant patterns from the data or as an inductive bias for building neural networks [2]. For instance, principal component analysis (PCA) can be seen as a low-pass filter in the correlation matrix, where only the parts of the data contributing to the directions of the largest variance are preserved [3]. Likewise, the success of convolutional neural networks (CNNs) can be attributed to the convolutional filters used in each layer, allowing for easier training and scalability, as well as exploiting structural invariances in the data [2], [4].

Conventional filtering applies to signals defined on Euclidean domains, but cannot be directly applied to irregular data structures arising in biological, financial, social, economic, power, water, sensor, and multi-agent networks, among others [5], [6]. Graph filters are information processing architectures tailored to graph-structured data, generalizing the conventional Euclidean counterparts.

Graph filters have many similarities with conventional ones; they are linear, shift invariant, parametric functions of the input, they enjoy a spectral interpretation via the spectral graph theory [7], and their spectral design boils down to function fitting [8], [9]. However, striking differences also arise from the new graph medium; e.g., graph filters are equivariant to permutations in the support, can be implemented distributively, and can have more generalized forms such as node varying [8] or edge varying [10]. Due to the wide variety of network-based data and the flexibility of graphs to represent irregular structures, graph filters are used in myriad SP tasks (signal reconstruction, anomaly detection, image processing, distributed processing) and ML tasks (semi-supervised and unsupervised learning, matrix completion, Gaussian process regression), as well as robotics, point clouds, Internet of Things, biology, and vision applications.

Early formalisms of graph filters find their roots in the 1990's in mesh processing [11], [12]. In the 2000's, graph filters were used in ML applications, mostly as graph kernels [13], [14], and later on in graph-based image processing [15], [16]. From a SP viewpoint, graph wavelets [17], [18] can be considered as the first instances. The tutorial article [19] helped provide a unifying mathematical framework for many of the problem-specific efforts that were being carried out in different research communities, encouraging more signal processing researchers to readily dive into problems involving network-based data and develop new filter methods derived from first principles, inspired from the more familiar Euclidean setting. Simultaneously, a specialization of the algebraic signal processing framework [20] to the graph domain paved the way for a structured mathematical framework of graph filtering [21], [22]. More recently, with the advent of graph neural networks (GNNs), graph filters play a fundamental role as the key component to learn representations from graph-based data [23]–[27].

Despite the early roots of graph filtering, and its span across different applications – often developed in an interdisciplinary fashion – there is no comprehensive, point-of-entry reference for new researchers interested in either conducting fundamental research or exploring applications related to SP and ML, or both. This article has been designed to target this need, thus providing an extensive, principled overview of the fundamental aspects of graph filtering research, as well as highlighting the main application areas in both SP and ML. A number of valuable tutorials and overviews on graph signal processing (GSP) and GNNs that are worth discussing have been written since [19]. The survey in [5] and book [28] provide excellent starting points on graph convolutional filtering and its links to the broader field of GSP. Since the focus of these works is on the fundamentals of GSP, they only cover a single type of graph filter and only scratch the surface on design methods and applications. The book chapter [29] provides more detail on filter design strategies

E. Isufi is with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands. Email: e-isufi.1@tudelft.nl.

F. Gama was with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005 USA. Email: fgama@ieee.org

D. I Shuman is with Franklin W. Olin College of Engineering, Needham, MA 02492 USA. Email: dshuman@olin.edu

S. Segarra is with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77005 USA. Email: segarra@rice.edu

and their role in filter banks but leaves out a large portion of other filtering methods and their applications. Ref. [30] focuses specifically on how a single-level graph convolutional filter bank can be used to build dictionaries of atoms for linear wavelet and vertex-frequency transforms. The recent tutorial [23] discusses the role of graph filters in building GNNs, whereas [27] shows that different filtering solutions lead to fundamentally different GNN architectures, evidencing trade-offs in terms of inductive and transductive learning, locality, and representation capacity. Nevertheless, these two works focus explicitly on the role of graph filters in understanding GNNs. A recent survey on the application of GSP tools and GNNs in machine learning can be found in [6], and another one discussing the applicability of a particular class of graph filters in [31]. Both works review the applications where such tools can be used, but do not detail the particular contribution of the filters, nor the implications of the choice of graph filter type. Here, we aim to introduce and compare in detail the different filter types, and subsequently show their role in staple SP and ML applications.

In short, the above works discuss particular forms or properties of graph filters linked to specific case studies or applications. None of them, however, offers a comprehensive and unifying treatment of the role of graph filtering that showcases the design choices, trade-offs, and applications of graph-based data in both SP and ML . Our goal in this paper is to give the reader a detailed tour on the different graph filtering forms and their properties, and to show how they can be used to develop more expressive solutions via filter banks and neural networks. We also aim to provide details on filter design and learning strategies. Another key contribution of this unifying framework is to show that graph filters are crucial in myriad applications in both SP and ML. While these two fields remain somewhat different in their approach to solving problems, a comprehensive understanding of filtering as the fundamental tool in both fields promotes collaboration and facilitates new research developments.

A. Organization

Sec. II sets up the basic concepts about graphs, signals and embeddings, including also a list of landmark applications encountered in SP and ML. Sec. III introduces the central piece of this paper, the graph convolutional filter (GCF), which is viewed as a shift-and-sum operation of graph signals with respect to any graph representation matrix (see Figs. 2 and 3), thus generalizing the principle of convolution for discrete-time signals. Here, we also discuss several properties of this filter from a vertex perspective, such as its invariances and distributed implementation. In Sec. IV, we characterize the spectral response of the GCF using a notion of graph Fourier transform. We then discuss the spectral properties of GCFs, including the generalization of the convolution theorem to the graph setting. Sec. V is dedicated to filter design and identification strategies, either when a user-specific operator (e.g., a low-pass filter) is given or when the design is based on input-output data pairs. In Sec. VI, we discuss other forms of graph filters and their link with the GCF. Table I provides an overview of the different architectures, their properties, and a discussion about the advantages and limitations of each form. Sec. VII is dedicated to building graph filter banks and graph wavelet transforms. We review different structures for graph filter banks (see, e.g., Figs. 5 and 6) and examine important design considerations. Then, in Sec. VIII, we show how the popular graph convolutional neural networks (GCNNs) can be seen as no more than a nonlinear graph filter built by nesting a GCF into an activation function (see Fig. 7). We also discuss here how to build a non-convolutional GNN by changing the GCF with another filter type from Table I. The next two sections are dedicated to staple applications of graph filters in SP (Sec. IX) – signal interpolation, anomaly detection, image processing, and distributed signal processing – and in ML (Sec. X) – semi-supervised and unsupervised learning on graphs, matrix completion, Gaussian processes, point clouds, and computer vision. Fig. 1 illustrates how the filtering concepts in Sections III-VI relate to the filterbanks and GNNs, and how each of them has been used in select applications. In Sec. XI, we provide some suggestions for researchers and developers new to the area, including a suggested order in which to explore the graph filtering tools. Finally, in Sec. XII, we highlight some future directions.

II. Graphs and Signals

This section first reviews ways to construct graphs and the basic terminology for representing them (Sec. II-A). This will bridge the high-level discussion of the previous section with the more detailed mathematical formulations of graph filters in the succeeding sections. Next, it highlights some landmark tasks where graph filters are used (Sec. II-B).

A. Graph Terminology

We denote a weighted graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{W})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges such that $(i, j) \in \mathcal{E}$ if and only if there is an edge from node i to node j, and $\mathsf{W} : \mathcal{E} \to \mathbb{R}_+$ is a weight function. If all edge weights equal one, the graph is said to be *unweighted*. A graph is *undirected* if there is no orientation in the edges in $\mathcal{E}$. For an undirected graph, we denote the neighboring set for a node i by $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$. Instead, in a directed graph (or digraph), an edge $(i, j) \in \mathcal{E}$ has an orientation starting from node i and ending at node j. We say that node j is an out-neighbor of i (and i an in-neighbor of j). The out-neighboring set of node i is denoted by $\mathcal{N}_i^{\text{out}} = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$ and, likewise, the in-neighboring set by $\mathcal{N}_i^{\text{in}} = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$.

We represent graph $\mathcal{G}$ via the weighted adjacency matrix $\mathbf{A}$, which is an $N \times N$ sparse matrix with nonzero elements $[\mathbf{A}]_{ji} = a_{ji} > 0$ representing the strength of edge $(i, j) \in \mathcal{E}$. Matrix $\mathbf{A}$ is symmetric ($a_{ij} = a_{ji}$ for all i, j) for an undirected graph, but may be asymmetric for a directed one. For undirected graphs, another widely used matrix is the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where the diagonal matrix $\mathbf{D} = \mathsf{diag}(\mathbf{A}\mathbf{1})$ has as ith diagonal element the sum of all edge weights incident to node i.

Graph shift operator (GSO). We represent the structure of a graph $\mathcal{G}$ with a generic matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ called the graph shift operator matrix. The only requirement for $\mathbf{S}$ to be a valid GSO is that

$$[\mathbf{S}]_{ji} = s_{ji} = 0 \quad \text{whenever} \quad (i, j) \notin \mathcal{E} \quad \text{for} \quad i \neq j.$$

Both matrices $\mathbf{A}$ and $\mathbf{L}$ are special cases for $\mathbf{S}$ [5], [21]. Other examples include the normalized adjacency matrix $\mathbf{A}_{\text{n}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, the normalized Laplacian matrix $\mathbf{L}_{\text{n}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, and the random walk Laplacian $\mathbf{L}_{\text{rw}} = \mathbf{D}^{-1}\mathbf{L}$.

Graphs and associated GSOs can represent:

1) *Physical networks:* Here, nodes and edges physically exist. For example, in a sensor network, nodes are sensors and

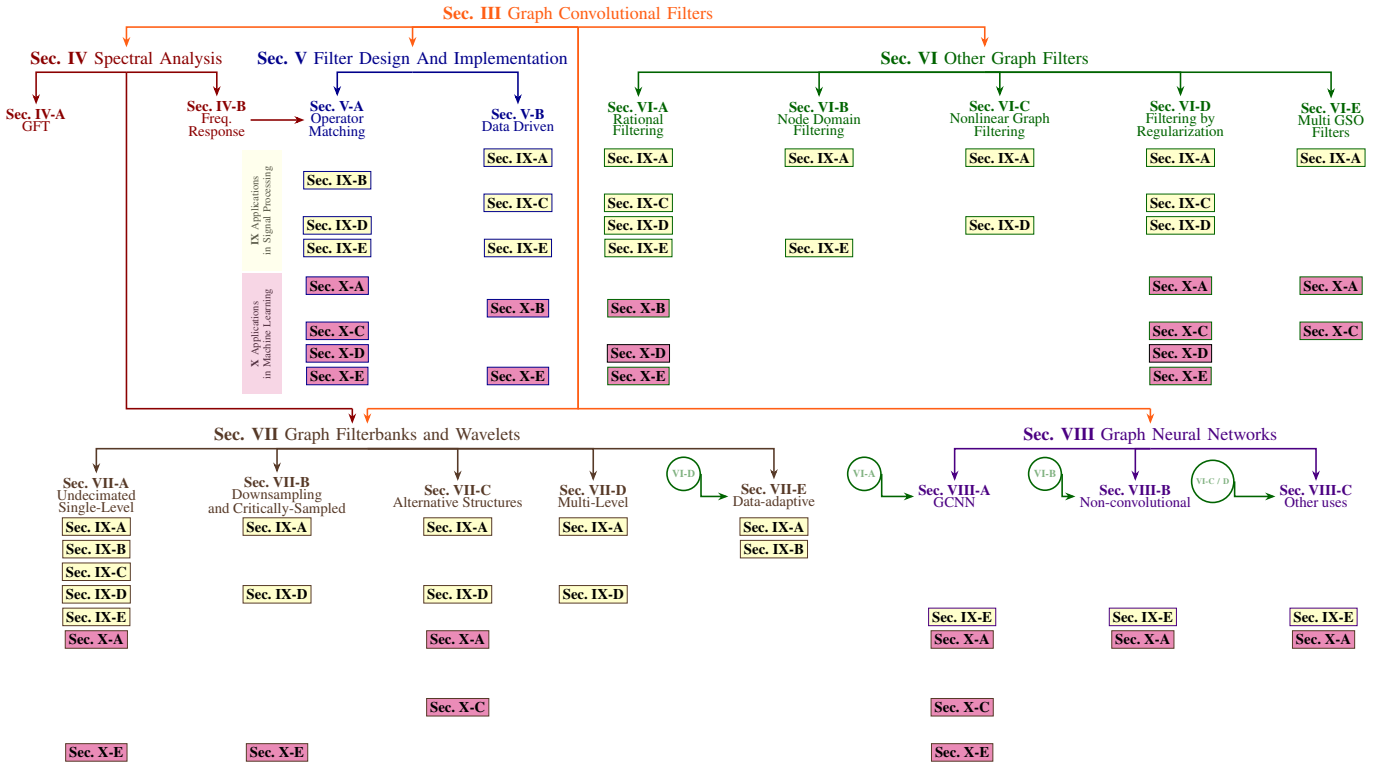


Fig. 1: A roadmap of this article. Solid arrows prerequisite relationships between sections. For example, Sec. VII can be mostly understood without reading Sec. VI, but not without reading Sec. IV. The boxes for applications in signal processing and machine learning correspond to specific application examples we discuss in this article, and are not meant to be a comprehensive representation of all work that has been done in the field.

edges are communication links [32]. A directed edge indicates the communication direction and the edge weight captures the communication channel properties. Other examples include: (i) multi-agent robot systems where nodes are robots and edges are communication channels [33]; (ii) power networks where nodes are buses and edges are power lines [34]; (iii) telecommunication networks where nodes are transceivers and edges are channels [35], [36]; (iv) water networks where nodes are junctions and edges are pipes [37], and; (v) road networks where nodes are intersections and edges are roads [38].

2) *Abstract networks:* These graphs typically represent dependencies between the data points. Consider N data points, each described by a feature vector $\mathbf{f}_i \in \mathbb{R}^F$, and let $\text{dist}(\mathbf{f}_i, \mathbf{f}_j)$ be a distance measure (e.g., Euclidean) between data points i and j. Each data point is considered as a node and two data points could be connected based on [39]: (i) $\varepsilon-$*neihborhood*, where the edge weight is

$$a_{ij} = \begin{cases} \mathsf{f}(\text{dist}(\mathbf{f}_i, \mathbf{f}_j); \theta) & \text{if } \text{dist}(\mathbf{f}_i, \mathbf{f}_j) \leq \varepsilon, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathsf{f}(\cdot;\theta)$ is a parametric function (e.g., a Gaussian kernel $\mathsf{f}(\text{dist}(\mathbf{f}_i, \mathbf{f}_j); \theta) = \exp\left(-\text{dist}(\mathbf{f}_i, \mathbf{f}_j)/2\theta^2\right)$ and $\varepsilon > 0$ is a constant controlling the edge sparsity; (ii) $k-$*nearest neighbor*, where each node is connected only to the k closest data points with respect to $\mathsf{f}(\text{dist}(\mathbf{f}_i, \mathbf{f}_j); \theta)$, which can again be a Gaussian kernel or a Pearson correlation. The covariance matrix or modifications thereof have also been used to build abstract networks as we elaborate on in Sec. IX-C; see also [40], [41].

The above approaches build undirected abstract networks but alternatives for directed or causal dependencies are also possible; see [40]–[42]. These abstract networks are useful, for example, in: (i) recommender systems, where two items are connected, e.g., if their Pearson correlation is greater than some value [43]; (ii) brain networks, where the nodes are brain regions and the edges are given, e.g., by the cross-correlation or mutual information of electroencephalography (EEG) time series in the different regions [44]; (iii) social networks, where nodes are users and edge weights may represent the number of interactions between them; (iv) economic networks, where nodes are different economic sectors and the edges may represent the input and output production going from one sector to another [45].

Since abstract networks represent dependencies between datapoints, they can be manipulated by recomputing edge weights, clustering, or pruning to facilitate representation. However, this is not typically the case for physical networks, as they often represent the medium with respect to which processing is performed. We shall see in Sec. IX-E that graph filters can leverage such structure for distributed processing.

B. Signals Defined on Graphs and Common Processing Tasks

We often encounter data that can be represented as a signal or set of features, with one value associated to each node.

Graph signal. A graph signal x is a function from the node set to the field of real numbers; i.e., $\mathsf{x} : \mathcal{V} \to \mathbb{R}$. We can represent a graph signal as a vector $\mathbf{x} \in \mathbb{R}^N$, where the ith entry $[\mathbf{x}]_i = x_i$ is the signal value at node i [19].

We denote the space of all graph signals defined on graph $\mathcal{G}$ with node set $\mathcal{V}$ as $\mathbb{X}^{\mathcal{V}} = \{\mathsf{x} : \mathcal{V} \to \mathbb{R}\}$. An example of a graph signal is a recording in a brain network, i.e., each brain area corresponds to a node, two nodes share a link based on structural connectivity, and the brain EEG measurement is the

signal of a particular node. We may want to process such a signal to understand, e.g., how different individuals have mastered a specific task [46].

Processing and learning tasks with graph signals include:

1) *Signal reconstruction, including interpolation and denoising:* We often observe a corrupted version of the graph signal, possibly at only a subset of nodes. Examples include noisy or subsampled measurements in sensor [32], power [34], and water networks [47]. The goal is to denoise the signal or interpolate the missing values by leveraging the neighboring signal values and the graph structure.
2) *Signal compression:* When graph signals have similar values at neighboring vertices, it is possible to *compress* the signal by developing representations that require fewer coefficients, and storing those coefficients rather than the original signal [30].
3) *Signal classification:* This task consists of classifying different graph signals observed over a common underlying graph. One such example is classifying patients based on their brain recordings, as discussed above [46].
4) *Node classification:* This task consists of classifying a subset of nodes in the graph given the class labels on another subset. When node features are available, we can treat them as a collection of graph signals and leverage their coupling with the underlying connectivity to infer the missing labels. The state-of-the-art for this task is achieved by GNNs, which, as we shall see in Section VIII, rely heavily on graph filters [23]. When node features are unavailable, we treat the available labels as graph signals and transform node classification into a label interpolation task that can be solved with graph filters [22].
5) *Graph classification / regression:* These tasks start with a collection of different graphs and (optionally) graph signals. The classification task assigns a label to the whole graph (e.g., classifying molecules into different categories such as soluble vs. non-soluble), whereas the regression task assigns a continuous number to each graph (e.g., the degree of solubility) [48].
6) *Link prediction:* Here, the goal is to infer if two nodes are connected given the current structure and the respective graph signals [49]. This is the case of friend recommendation in social networks, whereby leveraging the friendship connectivity between users based on their feature signals (e.g., geo-position, workplace) we can infer missing links.
7) *Graph identification:* This task extends the link prediction to that of inferring the whole graph structure given only the graph signals [41]. Graph filters play a role in modeling the relationships between candidate graph structures and the observed signals. We detail this problem in Sec. IX-C.
8) *Distributed processing:* Here the graph topology represents the structure of a sensor network and we want to distributively solve a task related to graph signals [9]. Graph filters lend themselves naturally to this setup because they rely only on local information. In Sec. IX-E, we discuss their use for different distributed tasks.

III. Graph Convolutional Filters

The convolution is a key operation in SP as it helps to define filtering operations and to understand linear, time-invariant systems. In ML, convolutional filters are the building block of CNNs, and their computational efficiency and parameter-sharing property tackle the curse of dimensionality. Convolutions also leverage the symmetries in the domain (such as translations in space) and allow for a degree of mathematical tractability with respect to domain perturbation [50]. We present here a now standard generalization of the convolutional filter to the graph domain, with the goal of inheriting the above properties. Then, in Sec. IV we analyze the filter behavior in the graph spectral domain, akin to the Fourier analysis for temporal filters, and in Sec. V we discuss strategies to design the filter parameters.

A. Definition

A convolutional filter is a *shift-and-sum* operation of the input signal [51]. While a shift in time implies a delay, a graph signal shift requires taking into account the topological structure.

Graph signal shift. A graph signal shift is a linear transformation $\mathsf{S}: \mathbb{X}^{\mathcal{V}} \to \mathbb{X}^{\mathcal{V}}$ obtained from applying a GSO $\mathbf{S}$ to a signal $\mathbf{x}$, i.e. $\mathsf{S}(\mathbf{x}) = \mathbf{S}\mathbf{x}$. The shifted signal at node i is computed as

$$[\mathbf{S}\mathbf{x}]_i = \sum_{j=1}^{N} [\mathbf{S}]_{ij}\mathbf{x} = \sum_{j \in \mathcal{N}_i^{\text{in}} \cup \{i\}} s_{ij} x_j, \tag{2}$$

which is a local linear combination of the signal values at neighboring nodes.

If the GSO is the adjacency matrix $\mathbf{A}$, the shifted signal represents a one-step propagation. Instead, if the GSO is the graph Laplcaian $\mathbf{L}$, the shifted signal is a weighted difference of the signals at neighboring nodes $[\mathbf{L}\mathbf{x}]_i = \sum_{j \in \mathcal{N}_i} a_{ij}(x_i - x_j)$.

Graph convolutional filter. Given a set of parameters $\mathbf{h} = [h_0, \ldots, h_K]^\top$, a graph convolutional filter of order K is a linear mapping $\mathsf{H}: \mathbb{X}^{\mathcal{V}} \to \mathbb{X}^{\mathcal{V}}$ comprising a linear combination of K shifted signals

$$\mathsf{H}(\mathbf{x}) = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x} \tag{3}$$

where $\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k$ is the $N \times N$ polynomial filtering matrix.

The output at node i is $y_i = h_0 x_i + h_1 [\mathbf{S}\mathbf{x}]_i + \ldots + h_K [\mathbf{S}^K \mathbf{x}]_i$, which is a linear combination of signal values located at most up to $K-$hops away. This is because $[\mathbf{S}^k]_{ji} \neq 0$ implies that there exists at least one path of length k between nodes i and j through which the signals can diffuse. These signals are shifted repeatedly over the graph as per (2); see also Fig. 2. The term *convolution* for (3) is rooted in the algebraic extension of the convolution operation [20] and the discrete-time counterpart can be seen as a particular case over a cyclic graph; see Box 1.

B. Properties

Graph convolutional filters satisfy the following properties.

Property 1 (Linearity)**.** For two inputs $\mathbf{x}_1$, $\mathbf{x}_2$, scalars α, β, and filter $\mathbf{H}(\mathbf{S})$, it holds that

$$\alpha \mathbf{H}(\mathbf{S})\mathbf{x}_1 + \beta \mathbf{H}(\mathbf{S})\mathbf{x}_2 = \mathbf{H}(\mathbf{S})(\alpha \mathbf{x}_1 + \beta \mathbf{x}_2).$$

Property 2 (Shift invariance)**.** The graph convolution is invariant to shifts, i.e., $\mathbf{S}\mathbf{H}(\mathbf{S}) = \mathbf{H}(\mathbf{S})\mathbf{S}$. This implies that given two

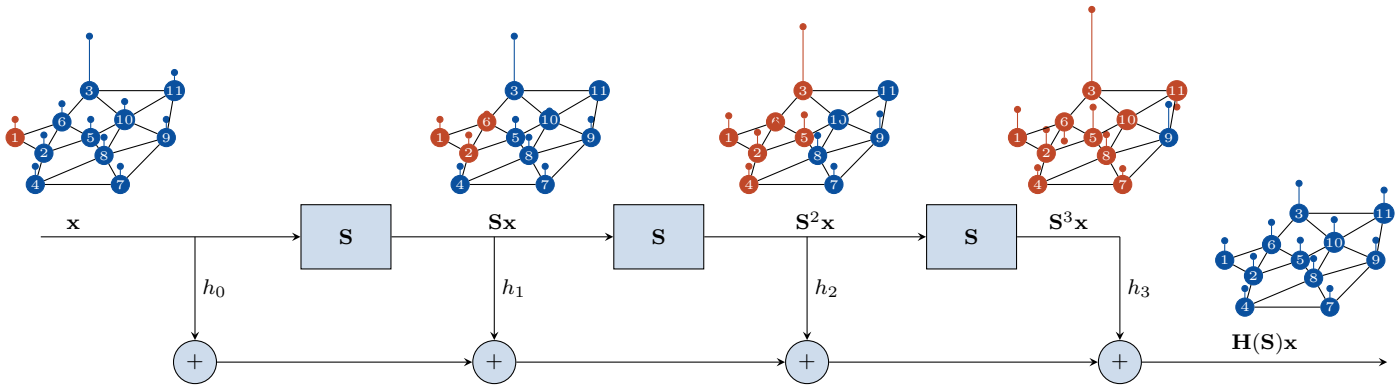


Fig. 2: The graph convolutional filter as a shift register. Highlighted are the nodes that reach node 1 on each consecutive shift; that is, the nodes j whose signal value x_j contributes to $[\mathbf{S}^k\mathbf{x}]_i$. The resulting summary of each communication $\mathbf{S}^k\mathbf{x}$ is correspondingly weighted by a filter parameter h_k. For each k, the parameter h_k is the same for all nodes. In this example, $\mathbf{S} = \mathbf{L}_n$ and $\mathbf{H}(\mathbf{S}) = 1\mathbf{L}_n^0 - 1.5\mathbf{L}_n^1 + 1\mathbf{L}_n^2 - 0.25\mathbf{L}_n^3$ is a lowpass filter that smooths the input signal.

filters $\mathbf{H}_1(\mathbf{S})$ and $\mathbf{H}_2(\mathbf{S})$ and an input signal $\mathbf{x}$, it holds that we can switch the order of the filters:

$$\mathbf{H}_1(\mathbf{S})\mathbf{H}_2(\mathbf{S})\mathbf{x} = \mathbf{H}_2(\mathbf{S})\mathbf{H}_1(\mathbf{S})\mathbf{x}.$$

Property 3 (Permutation equivariance)**.** Denote the set of permutation matrices by

$$\mathcal{P} = \{\mathbf{P} \in \{0,1\}^{N\times N} : \mathbf{P1} = \mathbf{1}\ ,\ \mathbf{P}^\mathsf{T}\mathbf{1} = \mathbf{1}\}.$$

Then for a graph with GSO $\mathbf{S}$ and $\mathbf{P} \in \mathcal{P}$, the permuted graph (the graph obtained by permuting the node indices by $\mathbf{P}^\mathsf{T}$) has the GSO $\hat{\mathbf{S}} = \mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}$, which describes the same topology but with a reordering of the nodes. Likewise, the permuted signal corresponding to the ordering in $\hat{\mathbf{S}}$ is $\hat{\mathbf{x}} = \mathbf{P}^\mathsf{T}\mathbf{x}$. Permutation equivariance for filter (3) implies

$$\mathbf{H}(\hat{\mathbf{S}})\hat{\mathbf{x}} = \mathbf{P}^\mathsf{T}\mathbf{H}(\mathbf{S})\mathbf{x};$$

i.e., the filter output operating on the permuted graph $\hat{\mathbf{S}}$ with the permuted signal $\hat{\mathbf{x}}$ is the permuted output of the same filter operating on the original graph $\mathbf{S}$ with the original signal $\mathbf{x}$.

Thus, graph convolutions are independent of the arbitrary ordering of the nodes. Moreover, the permutation equivariance shows that the graph convolutional filter exploits the signal patterns with respect to the underlying graph structure. This is a direct analogue of translation equivariance in temporal or spatial signals, where the respective convolutional filters are translation equivariant functions. This is key to their success in learning from a few training samples [4].

Property 4 (Parameter sharing)**.** All the nodes share the parameters among them. For two nodes i, j, the respective outputs are $y_i = h_0x_i + h_1[\mathbf{S}\mathbf{x}]_i + \ldots + h_K[\mathbf{S}^K\mathbf{x}]_i$ and $y_j = h_0x_j + h_1[\mathbf{S}\mathbf{x}]_j + \ldots + h_K[\mathbf{S}^K\mathbf{x}]_j$, which shows that the k-shifted signal $\mathbf{S}^k\mathbf{x}$ is weighted by the same parameter h_k.

Props. 3-4 imply that graph convolutions are inductive processing architectures. They can be designed or trained over a graph $\mathcal{G}$ and transferred to another graph $\widehat{\mathcal{G}}$ (with possibly a different number of nodes) without redesigning or retraining. This is particularly useful, e.g., when using graph filters for distributed SP tasks, as the physical channel graph may change. In Sec. IV (Prop. 8), we discuss the degree of transference.

Property 5 (Locality)**.** Graph convolutions are local architectures. To see this, set $\mathbf{z}^{(0)} = \mathbf{S}^0\mathbf{x}$. The one shifted signal $\mathbf{z}^{(1)} = \mathbf{S}\mathbf{x} = \mathbf{S}\mathbf{z}^{(0)}$ is local by definition. The $k > 1$ shift $\mathbf{z}^{(k)} = \mathbf{S}^k\mathbf{x}$ can be computed recursively as $\mathbf{z}^{(k)} = \mathbf{S}(\mathbf{S}^{(k-1)}\mathbf{x}) = \mathbf{S}\mathbf{z}^{(k-1)}$, which implies that the $(k-1)$st shift $\mathbf{z}^{(k-1)}$ needs to be shifted locally to the neighbors. Hence, to compute the output, each node exchanges locally with neighbors all K shifts $\mathbf{z}^{(0)}, \ldots, \mathbf{z}^{(K-1)}$.

Locality of computation makes the graph convolutional filters readily distributable, as we discuss in Sec. IX-E.

Property 6 (Linear computation cost)**.** Graph convolutions have a computational complexity of order $\mathsf{O}(K|\mathcal{E}|+KN)$; i.e., linear in the number of edges and filter order.

Props. 4-6 imply that graph convolutions tackle the curse of dimensionality in large graphs. The parameter sharing makes them suitable architectures to learn input-output mappings from a few training samples, irrespective of the graph dimensions (i.e., $\mathsf{O}(K)$ number of parameters); their locality allows them to extract patterns in the data in the surrounding of a node, and; their linear computational complexity facilitates scalability. In Sec. VIII, we discuss how to learn more expressive representations via neural networks while preserving these benefits in a form akin to CNNs for time series and images.

IV. Spectral Analysis

While in the previous section we discussed the graph convolutional filter in the vertex domain, we now shift attention to the graph spectral domain to characterize the frequency response of these filters. This frequency response is key to interpreting the filter behavior, and it facilitates design when we want to achieve a desired spectral response, as we detail in Sec. V.

A. Graph Fourier Transform

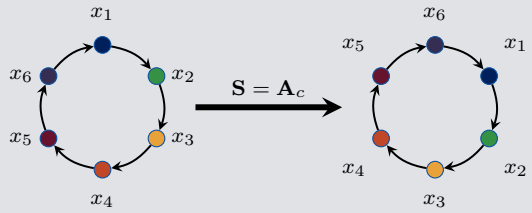
The DFT can be seen as the projection of a temporal signal onto the eigenvectors of the cyclic graph adjacency matrix (see Fig. 3). We define similarly the graph Fourier transform (GFT).

Graph Fourier transform (GFT). Consider the eigendecomposition of the diagonlizable GSO $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ with eigenvectors $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_N]$, and where $\mathbf{\Lambda} = \mathsf{diag}(\boldsymbol{\lambda})$ is a diagonal matrix with the corresponding eigenvalues $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_N]$. The GFT of a signal $\mathbf{x}$ is defined as the signal projection onto the GSO eigenspace

$$\tilde{\mathbf{x}} = \mathbf{V}^{-1}\mathbf{x}. \tag{4}$$

The inverse GFT is defined as $\mathbf{x} = \mathbf{V}\tilde{\mathbf{x}}$.

(Box 1) Discrete-time circular convolution. The graph signal shift (2), the graph convolutional filter (3), and their spectral equivalents in Sec. IV generalize the respective concepts developed for discrete-time periodic signals.



$$\mathbf{A}_c = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 3: Discrete-time periodic signals as graph signals over a directed cycle graph. Each node $\mathcal{V}_c = \{1, \ldots, 6\}$ is a time instant with adjacencies captured in the matrix $\mathbf{A}_c$. The temporal signal forms the graph signal $\mathbf{x} = [x_1, \ldots, x_6]^\mathsf{T}$ and the shift $\mathbf{A}_c\mathbf{x}$ acts as a delay operation that moves the signal to the next time instant node.

We can represent an arbitrary discrete-time periodic signal as a graph signal $\mathbf{x} = [x_1, \ldots, x_N]^\mathsf{T}$ residing over the vertices of a directed cyclic graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c)$ in which each node is a time instant and directed edges connect adjacent time instances of the form $(n, 1 + n \mod N)$, $n = 1, \ldots, N$, as shown in Fig. 3. The adjacency matrix of this graph is a cyclic matrix $\mathbf{A}_c$ such that $[\mathbf{A}_c]_{1+n \mod N, n} = 1$ and zeros everywhere else.

Signal shift: Setting the GSO $\mathbf{S} = \mathbf{A}_c$, operation (2) shifts the signal cyclically and acts as a delay operation, i.e., $[\mathbf{A}_c\mathbf{x}]_{1+n \mod N} = x_n$.

Convolutional filter: The graph convolutional filter (3) for graph $\mathcal{G}_c$ reduces to the circular convolution, i.e., the output at the temporal node i is $y_n = [\mathbf{H}(\mathbf{A}_c)\mathbf{x}]_n = \sum_{k=0}^{K} h_k [\mathbf{x}]_{1+(n-k+1) \mod N}$.

Signal variation: Using the total variation in (7), we measure how much the signal changes from its delayed version. This is a key building block for developing filters in standard signal processing [52].

Fourier transform: The cyclic adjacency matrix can be eigendecomposed as $\mathbf{A}_c = \mathbf{DFT}_N \text{diag}(\boldsymbol{\lambda}) \mathbf{DFT}_N^{-1}$ with eigenvectors $[\mathbf{DFT}_N]_{kn} = (1/\sqrt{N}) \exp^{\mathrm{j}2\pi(k-1)(n-1)/N}$ forming the discrete Fourier transform (DFT) matrix and eigenvalues $\boldsymbol{\lambda} = [\exp(-\mathrm{j}2\pi 0/N, \ldots, -\mathrm{j}2\pi(N-1)/N)]$ containing the frequencies. The DFT for signal $\mathbf{x}$ is $\tilde{\mathbf{x}} = \mathbf{DFT}_N \mathbf{x}$, which coincides with the graph Fourier transform (GFT) for this particular graph.

In the definition of the GFT, we are assuming the GSO $\mathbf{S}$ is diagonalizable. While definitions of GFT for nondiagonalizable GSOs exist [22], [53], [54], we hold to the diagonalizability assumption for a consistent and simple exposition. Furthermore, in some specified examples – particularly those with a spectral interpretation – we additionally assume that $\mathbf{S}$ is Hermitian; i.e., $\mathbf{S}$ is equal to $\mathbf{S}^\mathsf{H}$, its conjugate transpose. Common choices of shift operators such as the combinatorial and normalized graph Laplacians on undirected graphs satisfy this condition. Such operators have the nice property that $\mathbf{S} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{H}$, where the entries of the diagonal matrix $\boldsymbol{\Lambda}$ are real, and $\mathbf{V}$ is a unitary matrix satisfying $\mathbf{V}^\mathsf{H}\mathbf{V} = \mathbf{I}$. Refer, for example, to [55] for more details on the choice of the GSO and its spectral consequences.

The eigenvectors $\mathbf{V}$ serve as the basis expansion for the GFT. In the discrete-time case, the complex exponentials fulfill this role. The GFT coefficients $\tilde{\mathbf{x}}$ are the weights each of these eigenvectors contribute to represent the signal. Following again this analogy, the vector $\boldsymbol{\lambda}$ contains the so-called graph frequencies. Interpreting these graph frequencies $\boldsymbol{\lambda}$ and the respective GFT coefficients $\tilde{\mathbf{x}}$ requires understanding how the signal varies over the graph. In turn, measuring the signal variability requires accounting for the graph structure. We review two basic criteria used for undirected [19] and directed graphs [21], [22].

Undirected graphs. The variability of a signal over an undirected graph is measured via the quadratic variation (QV)

$$\mathsf{QV}(\mathbf{x}) := \mathbf{x}^\mathsf{H}\mathbf{L}\mathbf{x} = \frac{1}{2} \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} a_{ij}(x_i - x_j)^2, \tag{5}$$

which quantifies how much a signal at a node is different from that of the strong connected ones [19]. The lower $\mathsf{QV}(\mathbf{x})$, the smoother signal $\mathbf{x}$ is with respect to the underlying graph. In fact, the constant graph signal $\mathbf{x} = c\mathbf{1}$ has a zero variability.

Using (5), we can interpret the variability of the Laplacian eigenvectors $\mathbf{L} = \mathbf{V}\text{diag}(\boldsymbol{\lambda})\mathbf{V}^\mathsf{H}$ so as to provide a Fourier basis. Treating each eigenvector $\mathbf{v}_i$ as a graph signal, we have

$$\mathsf{QV}(\mathbf{v}_i) = \mathbf{v}_i^\mathsf{H}\mathbf{L}\mathbf{v}_i = \lambda_i. \tag{6}$$

Thus, we can sort eigenvectors based on their variability $0 = \mathsf{QV}(\mathbf{v}_1) \leq \mathsf{QV}(\mathbf{v}_2) \leq \ldots \leq \mathsf{QV}(\mathbf{v}_N)$, which implies that the respective eigenvalues $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$ carry the notion of frequency in the graph setting. We refer to the eigenvalues λ_i close to 0 as low frequencies and to those $\lambda_i \gg 0$ as high frequencies. The lowest graph frequency is $\lambda_1 = 0$ which corresponds to a constant eigenvector for a connected graph. Accordingly, when $\mathbf{S} = \mathbf{L}$, the GFT coefficient $\tilde{x}_i$ indicates how much eigenvector $\mathbf{v}_i$ contributes to the variability of signal $\mathbf{x}$ over $\mathcal{G}$.

Directed graphs. To measure the signal variability for directed graphs, we use again the analogy with the cyclic graph representing time signals, shown in Fig. 3. We measure how much the diffused signal $\mathbf{S}\mathbf{x}$ changes from the signal $\mathbf{x}$ via the total variation (TV)

$$\mathsf{TV}(\mathbf{x}) = \|\mathbf{x} - \mathbf{S}\mathbf{x}\|_1, \tag{7}$$

Expression (7) attains a high value if the shifted signal differs more from the original one. However, unlike the quadratic measure for undirected graphs (5), the total variation in (7) may be non-zero for constant signals.

If the shift operator is $\mathbf{S} = |\lambda_{\max}|^{-1}\mathbf{A}$ and $\lambda_{\max}$ is the eigenvalue of maximum amplitude, then $\mathsf{TV}(\mathbf{x}) = \|\mathbf{x} - |\lambda_{\max}|^{-1}\mathbf{A}\mathbf{x}\|_1$. In this case, we can measure the variability of the adjacency matrix eigenvectors $\mathbf{A} = \mathbf{V}\text{diag}(\boldsymbol{\lambda})\mathbf{V}^{-1}$; we have that $\mathsf{TV}(\mathbf{v}_i) \leq \mathsf{TV}(\mathbf{v}_j)$ iff $|\lambda_{\max} - \lambda_i| < |\lambda_{\max} - \lambda_j|$. That is, the eigenvector associated with the largest eigenvalue has the lowest variability, while the eigenvector associated with the eigenvalue

farthest from $\lambda_{\max}$ has the highest variability. Since the eigenvalues may be complex, the distances have to be computed in the complex plane. The order of the eigenvalues according to increasing variability is $\mathfrak{Re}\{\lambda_1\} \geq \mathfrak{Re}\{\lambda_2\} \geq \cdots \geq \mathfrak{Re}\{\lambda_N\}$, see [22, Figs. 2, 3]. In this case, the eigenvalues located (in a complex-plane sense) closest to the largest real eigenvalue are the ones corresponding to lower frequencies, while the eigenvalues located farthest from it correspond to the highest frequency.

Either on a directed or an undirected graph, the variability of a graph signal $\mathbf{x}$ can often be expressed by only a few $N' \ll N$ GFT basis vectors. In this case, we say the graph signal is $N'-$bandlimited and expand it as

$$\mathbf{x} = \mathbf{V}_{N'} \tilde{\mathbf{x}}_{N'}, \tag{8}$$

with $\mathbf{V}_{N'} = [\mathbf{v}_1, \ldots, \mathbf{v}_{N'}]$ and $\tilde{\mathbf{x}}_{N'} \in \mathbb{R}^{N'}$.

B. Frequency Response

By subsituting the eigendecomposition $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ into (3), we can write the filter output as

$$\mathbf{y} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x} = \sum_{k=0}^{K} h_k \mathbf{V}\mathbf{\Lambda}^k \mathbf{V}^{-1}\mathbf{x}. \tag{9}$$

Using (4) and defining the GFT of the output $\tilde{\mathbf{y}} := \mathbf{V}^{-1}\mathbf{y}$, we can write the filter input-output spectral relation as

$$\tilde{\mathbf{y}} = \sum_{k=0}^{K} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}}. \tag{10}$$

Convolution theorem for graph filters. It follows from (10) that a *shift-and-sum graph convolutional filter* of the form (3) operates in the spectral domain as a *pointwise multiplication* $\tilde{y}_i = \tilde{\mathsf{h}}(\lambda_i)\tilde{x}_i$ between the input signal GFT $\tilde{\mathbf{x}} = \mathbf{V}^{-1}\mathbf{x}$ and the filter frequency response

$$\tilde{\mathsf{h}}(\lambda) = \sum_{k=0}^{K} h_k \lambda^k. \tag{11}$$

Such a result is reminiscent of the convolution theorem [52], whereby the convolution in the graph domain corresponds to multiplication in the frequency domain. The filter frequency response is an analytic polynomial in λ and it is independent of the graph. The specific filter effect on a given graph is on the positions where the frequency response is instantiated; see Fig. 4.

In this context, graph convolutional filters satisfy the following spectral properties.

Property 7 (GFT of the filter)**.** Eq. (10) can be rewritten as

$$\tilde{\mathbf{y}} = \mathsf{diag}(\tilde{\mathbf{h}})\tilde{\mathbf{x}} \text{ with } \tilde{\mathbf{h}} = \mathbf{\Psi}\mathbf{h}, \tag{12}$$

where $\mathbf{\Psi} \in \mathbb{C}^{N\times(K+1)}$ is a Vandermonde matrix such that $[\mathbf{\Psi}]_{ik} = \lambda_i^{k-1}$. The vector $\tilde{\mathbf{h}} \in \mathbb{C}^N$ is known as the GFT of the filter parameters, which depends on the eigenvalues of the GSO (cf. (12)), unlike that of the signal, which depends on the eigenvectors (cf. (4)). Refer to [8] for more detail.

From (12), it follows that a graph convolutional filter defined as in (3) has the same frequency response for two frequencies with the same eigenvalues, as this results in $\mathbf{\Psi}$ having repeated rows. Alternatively, one can define graph convolutional filters through their action in the frequency domain, potentially accommodating different responses for repeated eigenvalues.

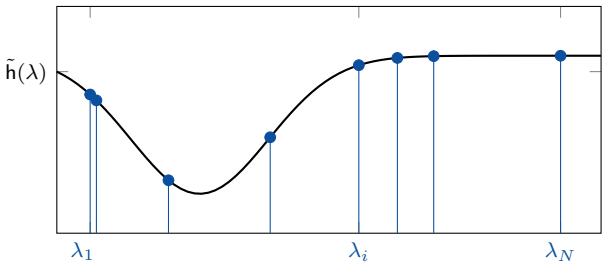


Fig. 4: The frequency response of the filter (11), given in the black solid line, is completely characterized by the values of the filter parameters $\mathbf{h}$. Given a graph, this frequency response gets instantiated on the specific eigenvalues of that graph, determining the effect the filter will have on an input (10).

Property 8 (Lipschitz continuity to changes in $\mathbf{S}$)**.** Let $\mathbf{S}, \hat{\mathbf{S}} \in \mathbb{R}^{N\times N}$ be two GSOs, potentially corresponding to different graphs with the same number of nodes N. Define the relative difference of $\hat{\mathbf{S}}$ with respect to $\mathbf{S}$ as

$$\mathsf{d}(\hat{\mathbf{S}}; \mathbf{S}) = \min_{\mathbf{E}\in\mathcal{R}(\hat{\mathbf{S}};\mathbf{S})} \|\mathbf{E}\| \tag{13}$$

for $\mathcal{R}(\hat{\mathbf{S}}; \mathbf{S}) = \{\mathbf{E} : \mathbf{P}^{\mathsf{T}}\hat{\mathbf{S}}\mathbf{P} = \mathbf{S} + (\mathbf{E}\mathbf{S} + \mathbf{S}\mathbf{E}) \ , \ \mathbf{P} \in \mathcal{P}\}$, the set containing all the relative difference matrices $\mathbf{E}$. Let the frequency response of the filter (cf. (11)) satisfy $|\lambda\tilde{\mathsf{h}}'(\lambda)| \leq C$ for some $C < \infty$ and where $\tilde{\mathsf{h}}'(\lambda)$ is the derivative of (11). Then, it holds that

$$\left\|\mathbf{H}(\hat{\mathbf{S}})\mathbf{x} - \mathbf{H}(\mathbf{S})\mathbf{x}\right\|_2 \leq \mathsf{d}(\hat{\mathbf{S}}; \mathbf{S})(1+8\sqrt{N})C\|\mathbf{x}\|_2 + \mathsf{O}(\mathsf{d}^2(\hat{\mathbf{S}}; \mathbf{S})). \tag{14}$$

Thus, if the relative difference between two GSOs is small, the outputs of the filters with the same input signal will also be small. This is a scenario that arises often when learning on graph-structured data. The graph observed at training time is often different than the graph observed at testing time. Thus, we would like to obtain certain guarantees that the learned filters will still be useful at inference time. Property 8 provides one type of guarantee. Refer to [56] for an extensive discussion on the choice of (13) as the function to capture differences between $\mathbf{S}$ and $\hat{\mathbf{S}}$.

Filters whose frequency response satisfies $|\lambda\tilde{\mathsf{h}}'(\lambda)| \leq C$ are known as integral Lipschitz filters. These filters may exhibit high variability for low values of λ (because its derivative can be high), but they have to be approximately constant for high values of λ (because its derivative has to be small). An example is shown in Fig. 4. For finite graphs with finite edge weights, all convolutional filters (cf. (3)) are integral Lipschitz within the spectrum interval of interest, but the constant C may be large. This constant depends only on the filter parameters and, thus, filters can be designed or learned to have a small value of C, guaranteeing a tighter bound; see [56] for details.

V. Filter Design and Identification

In this section, we discuss strategies to find the graph filter parameters to solve a specific task. We split our discussion into two common scenarios, each of which arises in the applications in Sec. IX and Sec. X. First, we consider designing the filter (3) to match (or approximately match) a given operator $\mathbf{B} \in \mathbb{R}^{N\times N}$; i.e., find $\mathbf{H}$ such that $\mathbf{H}(\mathbf{S}) = \mathbf{B}$ (or $\mathbf{H}(\mathbf{S}) \approx \mathbf{B}$) (Sec. V-A). Second, we seek a filter (3) that represents a data-driven mapping between input-output signal pairs (Sec. V-B).

A. Operator Matching

Many SP applications on graphs can be formulated as a linear operator $\mathbf{B}$ on the signal $\mathbf{x}$. Such an operator may arise from the solution to a denoising problem [57], be the consensus operator [58], or implement a specific spectral response that can be useful for graph wavelets (Sec. VII) or spectral clustering (Sec. X-B). We want to represent the operator as a graph filter to reduce the computational cost (Property 6) if the matrix $\mathbf{B}$ is dense, or to implement $\mathbf{B}$ distributively over a sensor network (Property 5). In the following, we distinguish between exactly and approximately matching the operator $\mathbf{B}$ with a graph filter.

Exact match. Denote by $\mathbf{H}(\mathbf{h}, \mathbf{S})$ the convolutional filtering matrix in (3), where we make explicit the dependency on parameters $\mathbf{h}$. The following holds.

Proposition 1 ([8]). *Given the three following conditions:*

1) *Matrices* $\mathbf{B}$ *and* $\mathbf{S}$ *are simultaneously diagonalizable; i.e.,* $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ *and* $\mathbf{B} = \mathbf{V}\mathsf{diag}(\boldsymbol{\beta})\mathbf{V}^{-1}$ *with eigenvalues* $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_N]^\top$.
2) *For all* (k_1, k_2) *such that* $\lambda_{k_1} = \lambda_{k_2}$ *, it holds* $\beta_{k_1} = \beta_{k_2}$.
3) *The order of* $\mathbf{H}(\mathbf{h}, \mathbf{S})$ *is such that* $K \geq D$*, where* D *denotes the number of distinct eigenvalues of* $\mathbf{S}$.

Then, $\mathbf{B} = \mathbf{H}(\mathbf{h}^*, \mathbf{S})$ *where* $\mathbf{h}^* = \mathbf{\Psi}^\dagger \boldsymbol{\beta}$*,* $\mathbf{\Psi}$ *is the Vandermonde matrix defined after* (12)*, and* $(\cdot)^\dagger$ *denotes the pseudo-inverse.*

Condition 1 implies that transformation $\mathbf{B}$ is diagonalized by the GFT matrix. This implies that we can specify the spectral response of the operation and implement it via the filter in (3). Since obtaining the eigendecomposition of $\mathbf{S}$ has a cost $\mathsf{O}(N^3)$, such an operation is not important for a centralized solution, as we could perfectly filter the signal in the spectral domain. However, it is important for distributed processing, because of the filter locality (Property 5). We shall detail this in Sec. IX-E.

Approximate match. When the conditions of Proposition 1 are too stringent (especially the first one), we resort to optimally approximating the desired operator. If Condition 1 holds, it might be that we want to approximate $\mathbf{B}$ with a low-degree polynomial (violating Condition 3) or that we do not have access to the specific eigenvalues of $\mathbf{B}$ due to the computational cost of obtaining them. In either case, we can perform the spectral approximation described below. If Condition 1 does not hold (i.e., $\mathbf{B}$ does not have the same eigenvectors as the shift operator), we can approximate $\mathbf{B}$ directly in the vertex domain via the non-spectral approximation described below.

Spectral approximation: Consider the common situation where $\mathbf{S}$ is a real, symmetric GSO, the desired operator $\mathbf{B}$ is jointly diagonalizable with $\mathbf{S}$, and the spectral response $\tilde{\beta}(\boldsymbol{\lambda})$ of $\mathbf{B}$ is a real-valued function. The goal is to find a low-order filter $\tilde{\mathsf{h}}(\lambda)$ that approximates this spectral response. If we have access to the specific eigenvalues of $\mathbf{S}$, we can easily obtain the spectral response $[\tilde{\beta}(\lambda_1), \tilde{\beta}(\lambda_2), \ldots, \tilde{\beta}(\lambda_N)]$ on those eigenvalues. Then, Proposition 1 offers the least squares approximate solution [22]. If, on the other hand, we only know the analytic expression of $\tilde{\beta}(\lambda)$ on a graph frequency interval $[\lambda_{\min}, \lambda_{\max}]$, the problem reduces to a one-dimensional polynomial approximation problem, such as the least squares problem

$$\tilde{\mathsf{h}} = \underset{\mathbf{h}}{\operatorname{argmin}} \int_{\lambda_{\min}}^{\lambda_{\max}} \left| \tilde{\beta}(\lambda) - \sum_{k=0}^{K} h_k \lambda^k \right|^2 d\lambda, \qquad (15)$$

or the minimax problem

$$\tilde{\mathsf{h}} = \underset{\mathbf{h}}{\operatorname{argmin}} \sup_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} \left\{ \left| \tilde{\beta}(\lambda) - \sum_{k=0}^{K} h_k \lambda^k \right| \right\}. \qquad (16)$$

The approaches in (15) and (16) are referred to as *universal design*, because the approximating filters are designed over the interval $[\lambda_{\min}, \lambda_{\max}]$, as opposed to specific eigenvalues. Thus, if the same filter is used on a different graph with the same spectral bounds, the approximating filter will also be the same. The solution to (15) can be found by orthogonally projecting $\tilde{\beta}(\lambda)$ onto the span of the first $K+1$ Legendre polynomials. A near-optimal solution to (16) can be found by truncating the expansion of $\tilde{\beta}(\lambda)$ into shifted Chebyshev polynomials [59] (see Box 2). Alternative minimax approximations are investigated in [60], [61]. For more details on the trade-offs involved in polynomial approximations, see, e.g, [9, Sec. V.C] and [62].

While these approximations are constructed for the entire interval $[\lambda_{\min}, \lambda_{\max}]$, it is only the approximation error at the (unknown) eigenvalues of $\mathbf{S}$ that affects the spectral approximation error. Thus, additional partial knowledge of the spectrum can be leveraged to improve the polynomial approximation. In this regard, [63] proposes a fast spectrum approximation method for specific families of graphs, whereas [64] leverages a fast estimation of the eigenvalue density for any graph shift operator to achieve a lower error in the high density regions of the spectrum. Alternatively, [58], [65] estimate the spectral distribution of frequencies via random matrix theory for random graphs (e.g., Erdős-Rényi). All these approaches are developed for symmetric GSOs with real eigenvalues, while extensions for directed graphs with complex eigenvalues are discussed in [66], [67].

Non-spectral approximation: When the desired operation $\mathbf{B}$ is not jointly diagonalized with $\mathbf{S}$, we can instead approximate it directly in the vertex domain, as stated by the following result.

Proposition 2 ([8]). *Define the* $N^2 \times (K+1)$ *matrix* $\mathbf{\Theta} = [\mathrm{vec}(\mathbf{I}), \mathrm{vec}(\mathbf{S}), \ldots, \mathrm{vec}(\mathbf{S}^K)]$*. The optimal filter parameters* $\mathbf{h}^* = \arg\min_{\mathbf{h}} \|\mathbf{B} - \mathbf{H}(\mathbf{h}, \mathbf{S})\|_{\mathrm{F}}$ *are* $\mathbf{h}^* = \mathbf{\Theta}^\dagger \mathrm{vec}(\mathbf{B})$.

B. Data-driven

In many cases, we do not know the exact operator but rather have input-output realizations of a graph-based system. This is for instance the case of opinion formation and source identification in social networks, biological signals supported on graphs, and modeling and estimation of diffusion processes in multi-agent networks. The assumption here is that the data input-output relation can be modeled as a graph filter map and our goal is to identify the filter parameters. Formally, consider the input $\mathbf{x}$ and output $\mathbf{y}$ satisfy

$$\mathbf{y} = \mathbf{H}(\mathbf{h}, \mathbf{S})\mathbf{x} + \mathbf{n}, \qquad (21)$$

where $\mathbf{n}$ is a zero-mean measurement noise. Different variants of the problem have been studied depending on whether $\mathbf{x}$ and $\mathbf{y}$ are fully observed or not, and whether we have additional side information (statistical or structural) about the input [68], [69]. Particularly, we distinguish between (i) *system identification*, where we estimate the parameters $\mathbf{h}$ from $\mathbf{S}$, $\mathbf{x}$, and a partial or complete observation of $\mathbf{y}$; and (ii) *blind deconvolution*, where we jointly estimate $\mathbf{h}$ and $\mathbf{x}$ from $\mathbf{y}$ and $\mathbf{S}$.

System identification. The goal is to use the (partial) observation of $\mathbf{y}$ to recover the unobserved elements of $\mathbf{y}$ and the filter

(Box 2) Chebyshev polynomial approximation. Stretched and shifted Chebyshev polynomials offer an orthogonal basis for approximating a desired frequency response $\beta(\boldsymbol{\lambda})$ via a low-order polynomial graph convolutional filter [9], [62]. Moreover, they offer a closed-form solution for the approximating polynomial filter coefficients. Formally, our goal is to approximate $\mathbf{B}\mathbf{x} = \mathbf{V}\mathsf{diag}(\tilde{\beta}(\lambda))\mathbf{V}^{\mathsf{H}}\mathbf{x}$ by $\mathbf{V}\mathsf{diag}(\tilde{\mathsf{h}}(\boldsymbol{\lambda}))\mathbf{V}^{\mathsf{H}}\mathbf{x}$, where $\tilde{\mathsf{h}}$ is a degree K polynomial. Let $\{\mathsf{T}_k(x)\}_{k=0,1,\ldots}$ be Chebyshev polynomials of the first kind, which form an orthogonal basis for the function space $L^2\left([-1,1], \frac{dx}{\sqrt{1-x^2}}\right)$. Since, our frequency response $\tilde{\beta}(\lambda)$ is defined on the interval $[0, \lambda_{\max}]$ (for positive semidefinite GSOs), we consider the change of variable $\lambda = \frac{1}{2}\lambda_{\max}(x+1)$. This leads to the stretched and shifted Chebyshev polynomials

$$\overline{\mathsf{T}}_k(\lambda) := \mathsf{T}_k\left(\frac{\lambda-\gamma}{\gamma}\right) \quad \text{and} \quad \gamma := \frac{\lambda_{\max}}{2}, \tag{17}$$

which can be used to expand the desired frequency response as

$$\tilde{\beta}(\lambda) = \frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k \overline{\mathsf{T}}_k(\lambda), \quad \forall \lambda \in [0, \lambda_{\max}], \tag{18}$$

where each parameter c_k can be found in closed-form by solving the integral

$$c_k := \frac{2}{\pi}\int_0^{\pi} \cos(k\theta)\tilde{\beta}\big(\gamma(\cos(\theta)+1)\big)d\theta. \tag{19}$$

For computational efficiency, we truncate the summation in (18) to a finite K, resulting in an approximation [59]

$$\tilde{\beta}(\lambda) \approx \tilde{\mathsf{h}}(\lambda) := \frac{1}{2}c_0 + \sum_{k=1}^{K} c_k \overline{\mathsf{T}}_k(\lambda), \quad \forall \lambda \in [0, \lambda_{\max}].$$

This Chebyshev polynomial approximation yields a Kth order graph convolutional filter (3); i.e., for any graph signal $\mathbf{x}$, we have

$$\mathbf{B}\mathbf{x} \approx \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} c_k \overline{\mathsf{T}}_k(\mathbf{S})\mathbf{x}, \tag{20}$$

where we can compute the kth term recursively as $\overline{\mathsf{T}}_k(\mathbf{S})\mathbf{x} = \frac{2}{\gamma}(\mathbf{S}-\gamma\mathbf{I})\overline{\mathsf{T}}_{k-1}(\mathbf{S})\mathbf{x} - \overline{\mathsf{T}}_{k-2}(\mathbf{S})\mathbf{x}$, with initial values $\overline{\mathsf{T}}_0(\mathbf{S})\mathbf{x} = \mathbf{x}$ and $\overline{\mathsf{T}}_1(\mathbf{S})\mathbf{x} = \frac{1}{\gamma}\mathbf{S}\mathbf{x} - \mathbf{x}$ [59]. The closed-form parameters can therefore be computed offline ahead of time and because of the recursive implementation, these filters can also be implemented distributively (cf. Property 5).

parameters $\mathbf{h}$. Consider only $M \leq N$ nodes are observed and define the sampling matrix $\mathbf{M} \in \{0,1\}^{M\times N}$, which has one 1 in each row m corresponding to the mth observed node and zero elsewhere. The filter identification problem comprises solving

$$\mathbf{h}^* = \arg\min_{\mathbf{h}} \|\mathbf{M}\big(\mathbf{y} - \mathbf{H}(\mathbf{h},\mathbf{S})\mathbf{x}\big)\|_2^2 + \gamma\|\mathrm{diag}(\boldsymbol{\omega})\mathbf{h}\|_1, \tag{22}$$

where $\boldsymbol{\omega} \in \mathbb{R}_+^{K+1}$ is a weighting vector [69]. The first term quantifies the fitting loss between the observed $\mathbf{M}\mathbf{y}$ and its prediction generated by $\mathbf{h}$. The second term is a sparsity-promoting regularizer on $\mathbf{h}$. Since we often do now know the filter degree, we can overestimate it and then penalize higher degrees to promote simpler filters. Consequently, we can select the weights in $\boldsymbol{\omega}$ to increase with the entry index. So, the parameters associated with higher powers of $\mathbf{S}$ are more heavily penalized, thus promoting a low-complexity and numerically stable model. The scalar $\gamma > 0$ is the regularizer weight relative to the fitting loss. Extended versions of problem (22) consider also estimating the topology in addition to the filter parameters; see e.g., [70]–[72].

Blind deconvolution. This problem arises when both the input $\mathbf{x}$ and the filter parameters $\mathbf{h}$ are unknown. To formalize this problem, for a given $\mathbf{S}$, $\mathbf{y}$ is a bilinear function of $\mathbf{h}$ and $\mathbf{x}$, which we can denote as $\mathbf{y} = \mathsf{A}(\mathbf{x}\mathbf{h}^\top)$. The linear operator $\mathsf{A}(\cdot)$ depends on the eigenvalues and eigenvectors of $\mathbf{S}$ and acts on the outer product of the sought vectors. More precisely, $\mathsf{A}(\mathbf{x}\mathbf{h}^\top) = (\boldsymbol{\Lambda}^\top \otimes (\mathbf{V}^\top)^{-1})^\top \mathrm{vec}(\mathbf{x}\mathbf{h}^\top)$, where $\otimes$ denotes the Khatri-Rao (i.e., columnwise Kronecker) product. While in principle we can jointly estimate $\mathbf{x}$ and $\mathbf{h}$, this leads to a non-convex problem with few theoretical guarantees. Instead, using the classical idea of lifting [73], we can derive a convex relaxation of the blind deconvolution problem by noting that $\mathbf{y}$ is a linear function of the entries of the rank one matrix $\mathbf{Z} = \mathbf{x}\mathbf{h}^\top$. Assuming further that $\mathbf{x}$ is a sparse vector (only a few nodes inject a signal into the filter), [68] proposes solving the convex problem

$$\mathbf{Z}^* = \arg\min_{\mathbf{Z}} \|\mathbf{y} - \mathsf{A}(\mathbf{Z})\|_2^2 + \gamma_1\|\mathbf{Z}\|_* + \gamma_2\|\mathbf{Z}\|_{2,1}. \tag{23}$$

The nuclear norm regularizer $\|\cdot\|_*$ promotes a low-rank solution since the true $\mathbf{Z}$ has rank one. The mixed norm $\|\mathbf{Z}\|_{2,1} = \sum_{i=1}^N \|\mathbf{z}_i\|_2$ is the sum of the ℓ_2-norms of the rows of $\mathbf{Z}$, thus promoting a row-sparse structure in $\mathbf{Z}$. This is aligned with the sparse assumption on $\mathbf{x}$, since each zero entry in $\mathbf{x}$ generates a whole row of zeros in the outer product $\mathbf{Z} = \mathbf{x}\mathbf{h}^\top$. Upon solving for $\mathbf{Z}^*$, we can recover $\mathbf{x}$ and $\mathbf{h}$ from, e.g., a rank one decomposition of $\mathbf{Z}^*$. Alternative relaxations to the row-sparsity and rank minimization have been proposed. For instance, [69] proposes a majorization-minimization procedure that yields lower-rank solutions compared to the convex relaxation in (23), whereas [74] provides a handle to control the row-sparsity of the recovered matrix.

Extensions to multiple input-output pairs (with a common filter) along with theoretical guarantees when the GSO $\mathbf{S}$ is normal (i.e., $\mathbf{S}\mathbf{S}^{\mathsf{H}} = \mathbf{S}^{\mathsf{H}}\mathbf{S}$) are investigated in [68]. The related case of a single graph signal as input to multiple filters (generating multiple outputs) is studied in [75], thus generalizing the classical blind multi-channel identification problem in DSP. This setting is relevant when studying a common stimulus to several

systems (e.g., the same image shown to several patients while their brain activity is recorded) or the effect of a single stimulus measured at different points in time (e.g., several snapshots of the spread of a rumor). The work in [76] addresses blind demixing when a single observation formed by the sum of multiple outputs is available, and it is assumed that these outputs are generated by different sparse inputs diffused through different graph filters. This setting is relevant when the observations are given by the superposition of several concurrent processes. For example, we can model a brain state as the result of the simultaneous reaction to several stimuli that we want to separate.

VI. Other Graph Filters

Graph convolutional filters implement a polynomial frequency response. Their descriptive power increases as we grow the filter order K. However, using higher orders implies handling higher matrix powers $\mathbf{S}^k$, which introduces numerical instabilities and in turn leads to poor interpolatory and extrapolatory performance [62]. While orthogonal polynomials (e.g., Chebyshev polynomials) can alleviate this issue, they still require a high number of parameters to implement the desired filtering function. Another limiting aspect of convolutional filtering is that its functions lie in the graph spectrum, meaning that there may not exist a GCF that is a sufficiently good approximation to a general operator.

In this section, we look at alternative graph filters to overcome these issues. We start with the family of filters that implement a rational response in Sec. VI-A. Then, in Sec. VI-B we discuss linear filters that go beyond the spectral analogy, a.k.a. node domain filtering, and in Sec. VI-C we discuss nonlinear graph filtering forms. Sec. VI-D shows how graph-based regularization techniques behave as graph filters, and Sec. VI-E discusses filtering with multiple graph shift operators. Table I provides a more extensive discussion of the properties in Sec. III-B and Sec. IV-B, as well as recommendations as to where to use them.

A. Rational Graph Filters

A rational graph filter implements the frequency response

$$\tilde{\mathsf{h}}(\lambda) = \left(\sum_{q=0}^{Q} b_q \lambda^q \right) \Big/ \left(1 + \sum_{p=1}^{P} a_p \lambda^p \right), \tag{24}$$

which is the ratio of two polynomials of orders Q and P that control the number of zeros and poles, respectively. This form achieves similar frequency responses as the convolutional filters but with fewer parameters; because rational functions have better interpolatory and extrapolatory properties than polynomials and require a lower order to achieve a similar approximation [62]. However, rational filters have stability issues. A rational graph filter is stable if the roots of its denominator are different from the GSO eigenvalues, i.e.,

$$\tilde{\mathsf{p}}(\lambda) := 1 + \sum_{p=1}^{P} a_p \lambda^p \neq 0, \ \forall \lambda \in \{\lambda_1, \ldots, \lambda_N\}. \tag{25}$$

If we do not have access to the specific eigenvalues, we can also impose *universal stability* by requiring condition (25) to hold for all potential eigenvalues in the interval $[\lambda_{\min}, \lambda_{\max}]$ [57], [77].

Given a stable filter and defining $\tilde{\mathsf{q}}(\lambda) = \sum_{q=0}^{Q} b_q \lambda^q$, we can write (24) as $\tilde{\mathsf{h}}(\lambda) = \tilde{\mathsf{q}}(\lambda)/\tilde{\mathsf{p}}(\lambda)$. Then the filter input-output relation in the spectral domain has the form $\tilde{y}_i = \tilde{\mathsf{q}}(\lambda_i)/\tilde{\mathsf{p}}(\lambda_i)\tilde{x}_i$ at each graph frequency λ_i. In the node domain, the rational filtering matrix has the form

$$\mathbf{H}(\mathbf{S}) = \left(\mathbf{I} + \sum_{p=1}^{P} a_p \mathbf{S}^p \right)^{-1} \left(\sum_{q=0}^{Q} b_p \mathbf{S}^q \right) := \mathbf{P}^{-1}(\mathbf{S}) \mathbf{Q}(\mathbf{S}), \tag{26}$$

where we define $\mathbf{P}(\mathbf{S}) := \mathbf{I} + \sum_{p=1}^{P} a_p \mathbf{S}^p$ and $\mathbf{Q}(\mathbf{S}) := \sum_{q=0}^{Q} b_p \mathbf{S}^q$ with respective frequency responses $\tilde{\mathsf{p}}(\lambda)$ and $\tilde{\mathsf{q}}(\lambda)$. When applied to a graph signal $\mathbf{x}$, we get the input-output relationship in the vertex domain

$$\mathbf{y} = \mathbf{P}^{-1}(\mathbf{S}) \mathbf{Q}(\mathbf{S}) \mathbf{x} \iff \mathbf{P}(\mathbf{S}) \mathbf{y} = \mathbf{Q}(\mathbf{S}) \mathbf{x}. \tag{27}$$

Expressions (24) and (27) show the two main challenges of rational graph filters. First, obtaining the output $\mathbf{y}$ from (27) requires solving a system of equations, which has a cubic order computational complexity $\mathsf{O}(N^3)$, making the filter impractical.[1] Second, designing a rational filter is more challenging than fitting a polynomial filter because of the nonlinear nature of the problem and the stability issues. In the remainder of this section, we discuss strategies to approach the latter.

Implementation. To reduce the computational cost of solving (27), we resort to iterative solvers that are fast and computationally efficient. If a centralized implementation is targeted, conjugate gradient approaches that exploit the graph structure are of interest [66]. They have a computational cost of $\mathsf{O}((PT+Q)|\mathcal{E}|)$, where T is the number of iterations. Because of the fast convergence of the conjugate gradient, we can stop it in a few tens of iterations. And since a rational function achieves good approximation with low orders P and Q, we expect the cost of obtaining the rational filter output to be low and comparable with that of graph convolutional filters (cf. (3)). Other works have considered the Jacobi method [27], [78], quasi-Newton methods [79], or pre-conditioned gradient descent [80] to speed-up the computation in particular cases or have lighter per-iteration computation cost.

Instead, if a distributed implementation is needed, the algorithm for solving (27) need also enjoy a local computation. Most strategies rely on first-order methods based either on ARMA-like recursions [57], [81], [82] or gradient-descent [77]. When the graph has a small diameter, the quasi-Newton method in [79] or the pre-conditioned gradient descent [80] could be a choice since they can be implemented locally with little effect on the performance.

Design. There are two streams of rational filter design, both reminiscent of rational fitting and filter design in DSP: *optimization-based* approaches and *change of variable* approaches.

Optimization-based: These methods can be cast as solving the constrained optimization problem

$$\begin{aligned} \underset{\{a_p, b_q\}}{\text{minimize}} \quad & \int_{\lambda_{\min}}^{\lambda_{\max}} \left| \tilde{\beta}(\lambda) - \frac{\tilde{\mathsf{q}}(\lambda)}{\tilde{\mathsf{p}}(\lambda)} \right|^2 d\lambda \\ \text{subject to} \quad & \tilde{\mathsf{p}}(\lambda) \neq 0, \ \forall \lambda \in [\lambda_{\min}, \lambda_{\max}], \end{aligned} \tag{28}$$

where $\tilde{\beta}(\lambda)$ is the desired response. Driven by their success in DSP [83], simple design approaches such as Prony's and Shanks' methods have been extended to the graph setting in [57], [66], [81], [82]. Such methods focus on the modified error $\tilde{\mathsf{e}}(\lambda) =$

[1] The inversion order cost matches that of the eigendecomposition of the shift operator. Consequently, there is no need to design a rational filter and apply it in the vertex domain as once we get the GFT of a signal we can implement exactly any desired spectral response.

$\tilde{\beta}(\lambda)\tilde{\mathsf{p}}(\lambda) - \tilde{\mathsf{q}}(\lambda)$ and ignore the stability constraint. It has been consistently observed that these simple approaches offer good fitting and stable filters. Instead, stability-enforcing solutions are devised in [84]–[86], which use, respectively, a sum-of-squares, partial factorization, and constrained weighted least squares.

Change of variable: The above strategies require solving an optimization problem that may be computationally demanding. To overcome this, some works extend the techniques that utilize Chebyshev polynomials to design convolutional filters to the rational filter design setting, ultimately yielding closed-form solutions and stable filters. Essentially, these methods: i) map graph frequencies $\lambda \in [0, \lambda_{\max}]$ into angular frequencies $\omega \in [0, \pi]$ via the transformation variable $\omega = \pi\lambda/\lambda_{\max}$; ii) design the filter for ω via standard DSP techniques, and; iii) generate the graph counterpart as $\tilde{\mathsf{h}}(\lambda) = |\tilde{\mathsf{h}}(\omega)|^2\big|_{\omega=\pi\lambda/\lambda_{\max}}$. References [87] and [77] use the Butterworth method for the design, while [88] considers rational Chebyshev design of the first kind. A link with the rational filtering design in DSP is also discussed in [89], while [90] proposes an iterative design via Chebyshev polynomials (cf. (17)) to approximate the inverse response. While having closed-form design, these approaches are often limited to ideal step responses in contrast to the optimization-based methods, which can be used for any response.

B. Node Domain Filtering

Convolutional and rational filters implement locally an operator that has a spectral response. However, in many cases, the desired operator or the data input-output mapping is more complex than a spectral response, which makes these solutions suboptimal (see also Sec. V). Thus, it is of interest to develop filters from a node domain perspective and potentially go beyond the spectral duality. Generally speaking, a graph filter of order K computes the output $[\mathbf{y}]_i$ at node i as a linear combination of the input signal localized in the $K-$hop neighbors $\mathcal{N}(i, K)$, i.e.,

$$[\mathbf{y}]_i = h_{ii}[\mathbf{x}]_i + \sum_{j \in \mathcal{N}(i,K)} h_{ij}[\mathbf{x}]_j, \tag{29}$$

where $\{h_{ij}\}$ are the parameters. These parameters account also for the graph structure (edge weights) underlying the signal $[\mathbf{x}]_j$, seen locally from node i. Here, we first relate the convolutional filtering (3) with operation (29) and then discuss extensions to node varying [8] and edge varying versions [10].

Convolutional filtering. Leveraging locality (Property 5), the convolutional filter obtains the information from k-hop neighbors as $\mathbf{z}^{(k)} = \mathbf{S}^k\mathbf{x}$. This can be written as

$$\begin{aligned}[\mathbf{y}]_i = [\mathbf{H}(\mathbf{S})\mathbf{x}]_i &= h_0[\mathbf{x}]_i + \sum_{k=1}^{K} h_k[\mathbf{z}^{(k)}]_i \\ &= h_0[\mathbf{x}]_i + \sum_{k=1}^{K} h_k[\mathbf{S}^k\mathbf{x}]_i. \end{aligned} \tag{30}$$

That is, the same parameter h_0 is applied to $[\mathbf{x}]_i$ by each node $i \in \mathcal{V}$, and the same parameters $\{h_k\}$ weight the $k-$hop neighboring signal locally percolated via the GSO, $[\mathbf{S}^k\mathbf{x}]_i$.

Node varying filtering. A *node varying graph filter* applies node-specific parameters h_{ki} to $[\mathbf{x}^{(k)}]_i$ and each $[\mathbf{S}^k\mathbf{x}]_i$; i.e.,

$$[\mathbf{y}]_i = h_{0i}[\mathbf{x}]_i + \sum_{k=1}^{K} h_{ki}[\mathbf{S}^k\mathbf{x}]_i. \tag{31}$$

Collecting the different parameters applied at shift k into the vector $\mathbf{h}_k = [h_{k1}, \ldots, h_{kN}]^\top$, we can write such a filter as

$$\mathsf{H}(\mathbf{x}) = \sum_{k=0}^{K} \mathsf{diag}(\mathbf{h}_k)\mathbf{S}^k\mathbf{x}. \tag{32}$$

This increased flexibility allows implementing more general operators than the convolutional filter, while still maintaining the local implementation. Results akin to those in Section V-A for exact and approximate operator matching have also been derived in [8] for node varying filters. To illustrate the result for exactly matching an operator $\mathbf{B}$, let us define $\boldsymbol{\delta}_i$ as the $N \times 1$ canonical vector with a 1 in position i and 0 elsewhere, and $\mathbf{u}_i = \mathbf{V}^\top\boldsymbol{\delta}_i$, $\mathbf{b}_i = \mathbf{B}^\top\boldsymbol{\delta}_i$, and $\bar{\mathbf{b}}_i = \mathbf{V}^\top\mathbf{b}_i$. With this notation in place, the following holds.

Proposition 3 ([8]). *If the following conditions hold for all i*

1) $[\bar{\mathbf{b}}_i]_j = 0$ *for those j such that* $[\mathbf{u}_i]_j = 0$
2) *For all (j_1, j_2) such that $\lambda_{j_1} = \lambda_{j_2}$, it holds that* $[\bar{\mathbf{b}}_i]_{j_1}/[\mathbf{u}_i]_{j_1} = [\bar{\mathbf{b}}_i]_{j_2}/[\mathbf{u}_i]_{j_2}$
3) *The degree of $\mathbf{H}(\mathbf{S})$ is such that $K \geq D$, where D denotes the number of distinct eigenvalues in $\mathbf{S}$*

then $\mathbf{B}$ can be perfectly implemented using a node varying graph filter as defined in (32)*.*

A direct comparison of Propositions 1 and 3 reveals the added expressivity of node varying graph filters since the stringent requirement of simultaneous diagonalization in Proposition 1 is replaced by the milder Condition 1 in Proposition 3. Similarly, data-driven design, modifications to the adaptive methodologies presented in Section V-B have also been extended to node varying filters in [91], [92].

Due to the node-specific nature of node varying filters, we can find frequency representations for every row of $\mathbf{H}(\mathbf{S})$ as

$$\boldsymbol{\delta}_i^\top\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} [\mathbf{h}_k]_i\mathbf{u}_i^\top\boldsymbol{\Lambda}^k\mathbf{V}^{-1} = \mathbf{u}_i^\top\mathsf{diag}(\tilde{\mathbf{h}}^{(i)})\mathbf{V}^{-1}, \tag{33}$$

where $\tilde{\mathbf{h}}^{(i)} = \boldsymbol{\Lambda}\mathbf{h}^{(i)}$ and $\mathbf{h}^{(i)} = [[\mathbf{h}_0]_i, [\mathbf{h}_1]_i, \ldots, [\mathbf{h}_K]_i]$ collects the filter parameters associated to node i. The output at node i is the elementwise product of the input Fourier transform $\mathbf{V}^{-1}\mathbf{x}$ and the filter being implemented at i, $\mathsf{diag}(\tilde{\mathbf{h}}^{(i)})$, and then combined with node-specific weights $\mathbf{u}_i^\top$ that encode how strong each frequency is represented by node i.

Edge varying filtering. We can further improve the filter flexibility by allowing each node to weight differently the information of its different neighbors. Then the diagonal parameter matrix in (32) becomes a matrix $\mathbf{H}_k$ with the same support as the GSO $\mathbf{S}$, leading to the *edge varying graph filter*:

$$\mathsf{H}(\mathbf{x}) = \sum_{k=0}^{K} \mathbf{H}_k\mathbf{S}^k\mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x}, \tag{34}$$

where $[\mathbf{H}_k]_{ij} = h_{kij}$ is the parameter node i applies to the signal of neighbor j at iteration k. For $k = 0$, we have $\mathbf{H}_0 := \mathsf{diag}(\mathbf{h}_0)$ since each node only weights its own signal [10]. By weighting differently the shifted information of the neighbors, the edge varying graph filter has an even higher flexibility and still preserves the local implementation. The latter is because the original signal values from the neighbors up to k hops away are still propagated through the graph via the GSO $\mathbf{S}^k\mathbf{x}$ and only then weighted locally by $\mathbf{H}_k$. It is shown in [10] that filter (34) can better approximate a defined operator compared with

the convolutional and rational filters. In addition, it also enjoys a spectral representation, but that requires long derivations and we refer the reader to [10]. Other forms of edge varying filters are developed in [10], [93], which use a parametric GSO and a cascaded form, respectively.

One of the main advantages of node domain filters is their increased degrees of freedom (DoFs) while preserving linearity and locality of implementation (Properties 1 and 5). As a result, both filters have a computational complexity of order $\mathsf{O}(K|\mathcal{E}|)$, even though the node varying filter (32) has $N(K+1)$ parameters and the edge varying filter (34) has $N+(N+|\mathcal{E}|)K$ parameters. The node varying graph filter is also proven Lipschitz stable (Property 8) in [94].

Since these filters are neither shift invariant nor permutation equivariant (i.e., Properties 2-3 do not hold), we need to account for the order in which we cascade them, as well as the node labeling. The lack of permutation equivariance also implies that we cannot transfer the learned filters across different graphs. Another challenge is that there may be too many DoFs (parameters) to estimate from limited data. In these instances, we can regularize the problem to penalize some norm of the parameters or develop a hybrid filter where edge varying parameters are applied only for a few representative nodes [27]. But when operating on a fixed graph and with a reasonable amount of data or a fixed operator, node domain filters can substantially improve the performance, especially in a distributed implementation.

C. Nonlinear Graph Filtering

Nonlinear filters have been proposed to overcome the limitations of node domain filters (large DoFs and lack of transferability across graphs), but still be more flexible than GCFs. To introduce nonlinear filters, we first focus on the graph convolutional filter (3) output $\mathbf{y}_i$ at node i. Specifically, we collect the $K-$shifted signals at node i in the vector $\mathbf{x}_i^{(K)} = \big[[\mathbf{x}]_i, [\mathbf{S}\mathbf{x}]_i, \ldots, [\mathbf{S}^K\mathbf{x}]_i\big]^\top$ and the filter parameters in the set $\mathcal{H} = \{\mathbf{h} = [h_0, \ldots, h_K]^\top\}$. Then, we can write the filter output at node i as

$$y_i = \mathsf{f}(\mathbf{x}_i^{(K)}; \mathcal{H}) := \mathbf{h}^\top \mathbf{x}_i^{(K)}, \ \forall\ i = 1, \ldots, N; \tag{35}$$

i.e., it is a multivariate linear regression in $\mathbf{x}_i^{(K)}$ with parameters $\mathbf{h}$ that are shared among the nodes. We also see from (31) and (34) that node varying filtering is a linear variation of (35) but with different parameters $\mathbf{h}_i$ for each node. In contrast, nonlinear graph filters can be built by considering a nonlinear function $\mathsf{f}(\mathbf{x}_i^{(K)}; \mathcal{H})$ in (35) with the same parameters $\mathcal{H}$ for all nodes. While the function $\mathsf{f}(\cdot)$ can be arbitrary, it has been studied for two models inspired by traditional signal processing: the Volterra graph filter [95] and the median graph filter [96], [97].

Volterra filter. The natural generalization of (35) is to consider a multivariate polynomial regressor in variables $\mathbf{x}_i^{(K)}$:

$$y_i = \mathsf{f}(\mathbf{x}_i^{(K)}; \mathcal{H}) := \text{poly}_{L_0,\ldots,L_K}(\mathbf{x}_i^{(K)}; \mathcal{H}), \tag{36}$$

where $\text{poly}_{L_0,\ldots,L_K}(\cdot)$ denotes a multivariate polynomial of orders $L_0, \ldots, L_K$ in $[\mathbf{x}]_i, [\mathbf{S}\mathbf{x}]_i, \ldots, [\mathbf{S}^K\mathbf{x}]_i$, respectively, and the set $\mathcal{H}$ collects the respective parameters. For instance, for a shift order $K = 1$ and polynomial orders $L_0 = 2, L_1 = 3$, we have $\mathbf{x}_i^{(1)} = \big[[\mathbf{x}]_i, [\mathbf{S}\mathbf{x}]_i\big]^\top$ and (36) becomes

$$y_i = \sum_{l_0=0}^{L_0=2} \sum_{l_1=0}^{L_1=3} h_{l_0 l_1} [\mathbf{x}]_i^{l_0} [\mathbf{S}\mathbf{x}]_i^{l_1} \text{ and } \mathcal{H} = \{h_{l_0 l_1}\}. \tag{37}$$

Expressed compactly, this filter has the input-output relation

$$\mathbf{y} = \sum_{l_0=0}^{L_0=2} \sum_{l_1=0}^{L_1=3} h_{l_0,l_1} \big[\mathbf{x}^{\odot l_0} \odot (\mathbf{S}\mathbf{x})^{\odot l_1}\big] \text{ and } \mathcal{H} = \{h_{l_0 l_1}\}, \tag{38}$$

where $\mathbf{x}^{\odot a} = \mathbf{x} \odot \ldots \odot \mathbf{x}$ is the element-wise $a-$th power of $\mathbf{x}$. Then, a nonlinear graph filter of order K has the form

$$\mathbf{y} = \sum_{l_0=0}^{L_0} \cdots \sum_{l_K=0}^{L_K} h_{l_0 \ldots l_K} \big[\mathbf{x}^{\odot l_0} \odot (\mathbf{S}\mathbf{x})^{\odot l_1} \odot \ldots \odot (\mathbf{S}^K\mathbf{x})^{\odot l_K}\big], \tag{39}$$

where set $\mathcal{H} = \{h_{l_0 \ldots l_K}\}$ collects all the parameters of order $\mathsf{O}(KL_{\max})$ with $L_{\max} = \max\{L_0, \ldots, L_K\}$. Because data are gathered locally, these filters generate the output with the same order of computational complexity. Such an increased flexibility allows us to represent more complex nonlinear relationships in graph input-output data. But, at the same time, multivariate polynomial regression can overfit the data and may suffer from ill-conditioning. Differently from the node or edge varying filter, the nonlinear filter in (39) shares the parameters across nodes, which allows transferring it across graphs. The Volterra graph filter is the particular case of (39) with reduced DoFs ($L_0 \leq L_1 \leq \ldots \leq L_K$) and it has been shown that even if the input is a bandlimited signal (cf. (8)), the output can have frequency content in the entire graph spectrum [95].

Median filter. All the above filters rely on signal propagation over the graph. When a particular node is anomalous and has, e.g., a large signal value, it will affect all the neighbors and the filter output. Median graph filters have been proposed as robust alternatives that can tackle such an issue.

Consider an integer $h \geq 0$ and a real scalar x. We define the replication operation $h \diamond x = [x, \ldots, x]^\top \in \mathbb{R}^h$. Then, the median graph filter output at node i can be obtained as

$$y_i = \mathsf{f}(\mathbf{x}_i^{(K)}; \mathcal{H}) := \text{Med}(h_0 \diamond [\mathbf{x}]_i; \ldots; h_K \diamond [\mathbf{S}^K\mathbf{x}]_i), \tag{40}$$

where the median operation $\text{Med}(\cdot)$ sorts its arguments in ascending order and outputs the middle one [97]. In obtaining $[\mathbf{S}\mathbf{x}]_i$, we compute a weighted linear combination of the entries in $\mathbf{x}$, where the weights are given by the values in the ith row of $\mathbf{S}$. If a node has a particularly high value, it can be amplified via the shift operator $\mathbf{S}$ and be present in $\mathbf{x}_i^{(K)}$ for almost all nodes. The median operator attenuates such influence. The data-driven design of the parameters $\mathbf{h}$ is discussed in [97]. Cases where the weights in $\mathbf{S}$ can be designed are also studied in [97]. As with the Volterra filter, the median filter is nonlinear and local; however, it does not enjoy a spectral equivalence. Alternative expressions to (40) are also proposed in [96], [97]. They differ in how the data are gathered at the nodes (either linearly via shifting or nonlinearly via the median operator) and how these gathered data are processed (again linearly or via a median operator). Lastly, we remark that the local median operator in (40) is only one choice and other nonlinear functions such as max or min can be used [98], [99].

D. Filtering by Regularization

The graph filters discussed above can be seen as graph-based parametric functions to model input-output mappings. When the spectral specifications of this mapping are unclear or when the amount of data is limited, these parametric filters can be difficult to design or can easily overfit the data. In these cases, we may

want to implement graph filtering via regularization, leveraging prior information about particular properties that graph signals exhibit. For simplicity, we consider graph regularization to denoise graph signals, which is crucial in data processing; however, similar observations extend also to interpolating missing values, as we shall see in Sec. IX-A.

Consider the task of recovering a graph signal $\mathbf{z}$ from a single noisy obervation $\mathbf{x} = \mathbf{z} + \mathbf{n}$, with $\mathbf{n}$ being additive Gaussian noise. This can be addressed by solving

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \mathsf{f}(\mathbf{x},\mathbf{y}) + \gamma \mathsf{r}(\mathbf{y},\mathcal{G}), \tag{41}$$

where $\mathsf{f}(\mathbf{x},\mathbf{y})$ is the fitting-term, typically $\mathsf{f}(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2^2$, and $\mathsf{r}(\mathbf{y},\mathcal{G})$ imposes a graph-based prior about the true signal. Depending on the signal behavior with respect to the underlying graph, we discuss three regularization techniques: (i) smooth filtering; (ii) sparsity filtering; and (iii) Wiener filtering.

Smooth filtering. These approaches consider a regularizer that imposes a low signal variation between adjacent nodes. For undirected graphs, two popular approaches are the Tikhonov regularizer and the Sobolev regularizer, while for directed graphs, the total variation regularizer is commonly applied.

Tikhonov [14], [19]: A smooth signal $\mathbf{y}$ over an undirected graph has a low quadratic form $\mathsf{LQ}(\mathbf{y}) = \mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y}$ (cf. (5)). Considering LQ as a regularizer, we obtain

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{y}\|_2^2 + \gamma \mathbf{y}^\top \mathbf{L}\mathbf{y}. \tag{42}$$

The more we increase $\gamma \gg 0$, the more we prioritise smoothness on the solution. Problem (42) is a quadratic convex problem and has the closed-form solution

$$\mathbf{y}^* = (\mathbf{I} + \gamma \mathbf{L})^{-1}\mathbf{x}. \tag{43}$$

Comparing (43) with (27), we see that the Tikhonov filter is an order one rational filter with frequency response $\tilde{\mathsf{h}}(\lambda) = (1+\gamma\lambda)^{-1}$. This frequency response also helps understanding the role of parameter γ; the optimal solution in (43) is a low-pass graph filter and the higher γ, the more low-pass the filter.

Sobolev [100]: The Sobolev regularizer increases the flexibility by allowing for a more expressive rational frequency response. Specifically, it focuses on solving

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{y}\|_2^2 + \gamma \mathbf{y}^\top (\mathbf{L}+\epsilon\mathbf{I})^\beta \mathbf{y}, \tag{44}$$

with $\epsilon \geq 0$ and $\beta \in \mathbb{R}_+$. The closed-form solution is

$$\mathbf{y}^* = \left(\mathbf{I} + \gamma(\mathbf{L}+\epsilon\mathbf{I})^\beta\right)^{-1}\mathbf{x}, \tag{45}$$

which corresponds to a rational filter with the frequency response $\tilde{\mathsf{h}}(\lambda) = (1+\gamma(\lambda+\epsilon)^\beta)^{-1}$. Here, the scalar β controls the expressivity order of this function (cf. P in (24)) and γ, ϵ are the parameters of such a rational response.

Quadratic shift variation [22]: When the graph is directed, we measure the variability as the change between the signal $\mathbf{y}$ and its shifted version $\mathbf{S}\mathbf{y}$. Thus, we can recover a smooth signal over a directed graph by solving

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{y}\|_2^2 + \gamma \|\mathbf{y}-\mathbf{S}\mathbf{y}\|_2^2, \tag{46}$$

which has a closed-form solution

$$\mathbf{y}^* = \left(\mathbf{I} + \gamma(\mathbf{I}-\mathbf{S}-\mathbf{S}^\top+\mathbf{S}^\top\mathbf{S})\right)^{-1}\mathbf{x}. \tag{47}$$

This is again an inverse graph filtering, but it does not admit a straightforward spectral analogy as (43) and (45). The role of the regularization parameter γ is discussed from a bias-variance perspective in [101] and from a graph-kernel perspective in [102]. Differently, [103] generalizes (43) to the case where each node has its own regularization parameters (i.e. a vector of parameters $\boldsymbol{\gamma}$), yielding a rational node varying filter (cf. (32)).

Sparsity filtering. These approaches leverage the prior that the signal has discontinuities across neighbors or shifts; e.g., a piecewise smooth signal that has homogeneous values within a group of nodes but can have arbitrarily large variations between groups. For undirected graphs, sparsity filtering is implemented via graph trend filtering (GTF), while for directed graphs it is implemented via the total variation in (7).

Trend filtering [104]: Let $\boldsymbol{\Delta} \in \mathbb{R}^{N\times|\mathcal{E}|}$ be the oriented incidence matrix of an undirected graph $\mathcal{G}$, whose rows are indexed by the nodes and columns by the edges. The operation $\boldsymbol{\Delta}^\top\mathbf{x}$ computes the pairwise difference between signal values on each edge; hence, $\boldsymbol{\Delta}^\top$ can be interpreted as a graph difference operator. In fact, since $\mathbf{L} = \boldsymbol{\Delta}\boldsymbol{\Delta}^\top$, the regularizer in (42) can be written as $\mathsf{LQ}(\mathbf{y}) = \mathbf{y}^\mathsf{T}\mathbf{L}\mathbf{y} = \|\boldsymbol{\Delta}^\top\mathbf{y}\|_2^2$, i.e., the squared ℓ_2-norm of the difference vector. Instead, the GTF works with regularized problems of the form

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{y}\|_2^2 + \gamma\|\boldsymbol{\Delta}^\top\mathbf{y}\|_1, \tag{48}$$

which penalizes the absolute difference of the signal variation in connected nodes. Problem (48) is an order $K=1$ GTF and estimates a signal $\mathbf{y}$ whose differences are nonzero only at a few edges. Higher-order GTFs substitute the indicence matrix $\boldsymbol{\Delta}^{(1)} := \boldsymbol{\Delta}$ in (48) with the higher-order versions $K \geq 1$

$$\boldsymbol{\Delta}^{(K+1),\top} = \begin{cases} \boldsymbol{\Delta}\boldsymbol{\Delta}^{(K),\top} = \mathbf{L}^{\frac{K+1}{2}}, & \text{for odd } K \\ \boldsymbol{\Delta}^\top\boldsymbol{\Delta}^{(K),\top} = \boldsymbol{\Delta}^\top\mathbf{L}^{\frac{K}{2}}, & \text{for even } K \end{cases}.$$

For an odd K, the GTF recovers a signal that has sparse diffused versions $\mathbf{L}\mathbf{y}, \mathbf{L}^2\mathbf{y}$, while for an even K, it recovers a signal that has sparse differences of the shifted versions $\boldsymbol{\Delta}^\top\mathbf{L}\mathbf{y}, \boldsymbol{\Delta}^\top\mathbf{L}^2\mathbf{y}$ etc. These sparsity constraints capture discontinuities in the graph signal and recover piecewise constant signals better than smooth filtering methods. One of the challenges of the GTF is that solving problem (48) requires running iterative algorithms. In addition, the ℓ_1-norm in (48) may often penalize towards zero when the signal components are large. To overcome the latter, the work in [105] proposes a GTF with a non-convex regularizer.

Total variation [22]: This is the straightforward extension of (46) that uses the regularizer $\mathsf{TV}_1(\mathbf{y})$; i.e., it solves

$$\underset{\mathbf{y}\in\mathbb{R}^N}{\operatorname{argmin}} \ \|\mathbf{x}-\mathbf{y}\|_2^2 + \gamma\|\mathbf{y}-\mathbf{S}\mathbf{y}\|_1. \tag{49}$$

This means that we are penalizing shifted variations that are substantial only at a few nodes. Similar to the GTF, this is also a convex problem that can be solved with iterative algorithms.

Wiener filtering. The above regularizers do not consider any statistical behavior of the true signal. When this signal exhibits a graph wide sense stationary behavior [3], [106], [107] or when it is generated by a Gaussian-Markov random field [108], we can incorporate such a prior to recover the optimal signal in a Wiener filtering sense.

First, consider a signal from a distribution $\mathbf{y} \sim \mathcal{D}(\mathbf{0}, \boldsymbol{\Sigma}_y)$ with covariance matrix $\boldsymbol{\Sigma}_y$ and let the noise be additive with zero

mean and covariance matrix $\mathbf{\Sigma}_n$. Given $\mathbf{y}$ and $\mathbf{n}$ are mutually independent, the Wiener filter comprises solving

$$\mathbf{H}^* = \underset{\mathbf{H}\in\mathbb{R}^{N\times N}}{\operatorname{argmin}} \mathbb{E}\|\mathbf{H}(\mathbf{y}+\mathbf{n})-\mathbf{y}\|_2^2 = \mathbf{\Sigma}_y(\mathbf{\Sigma}_y+\mathbf{\Sigma}_n)^{-1}, \quad (50)$$

and setting the solution to $\mathbf{y} = \mathbf{H}^*\mathbf{x}$. When the process $\mathbf{y}$ is defined over a graph and the covariance matrices have the same eigenvectors as the GSO – i.e., $\mathbf{\Sigma}_y = \mathbf{V}\mathsf{diag}(\sigma_y^2(\boldsymbol{\lambda}))\mathbf{V}^{\mathsf{H}}$, $\mathbf{\Sigma}_n = \mathbf{V}\mathsf{diag}(\sigma_n^2(\boldsymbol{\lambda}))\mathbf{V}^{\mathsf{H}}$ (independent noise) – the Wiener filter in (50) reduces to a graph Wiener filter $\mathbf{H}(\mathbf{S})$ [109]. This graph Wiener filter has the frequency response

$$\tilde{\mathsf{h}}(\lambda) = \frac{\sigma_d^2(\lambda)}{\sigma_d^2(\lambda)+\sigma_n^2(\lambda)} = \frac{1}{1+\frac{\sigma_n^2(\lambda)}{\sigma_d^2(\lambda)}}, \quad (51)$$

which is a rational filter, and the response at frequency λ is controlled by the inverse signal-to-noise (SNR) ratio $\mathrm{SNR}^{-1}(\lambda) := \sigma_n^2(\lambda)/\sigma_d^2(\lambda)$. Contrasting (51) with the other regularized filters, we see that the Wiener filter does not imply a constant regularization weight γ for each frequency λ, but rather a frequency-adaptive regularizer given by the inverse SNR. Similar to the rational graph filters discussed above, the output of the Wiener filter can be obtained with conjugate gradient methods; however, the matrices $\mathbf{\Sigma}_y, \mathbf{\Sigma}_n$ are typically dense. One way to tackle this is to approximate $\tilde{\mathsf{h}}(\lambda)$ with polynomial or rational filters and then implement it via iterative recursions [110], [111].

The main challenge of these filters is to identify a good regularizer or a combination thereof that represents the data. Often this may be a challenging task requiring domain expertise, hence advocating for the easier solution to use more general graph filters as input-output mappings.

E. Multi-GSO Filters

Unlike classical signal processing where the shift operation is a time delay, in the graph setting, different choices of the graph shift operator for the data are often possible, especially in abstract networks (Sec. II-A). Designing or learning both the filter coefficients and the GSO is challenging because of the powers of $\mathbf{S}$ appearing in the filter expression (e.g., (3)), and because of the high DoFs that can cause overfitting. A way to circumvent these challenges is to build a graph filter operating on multiple pre-specified GSOs [112]–[115]. Given Q GSOs $\{\mathbf{S}_q\}_{q=1}^Q$, we define a multi-GSO graph filter as

$$\mathsf{H}(\mathbf{x}) = \sum_{q=1}^{Q}\sum_{k=0}^{K} h_{qk}\mathbf{S}_q^k\mathbf{x} = \mathbf{H}(\{\mathbf{S}_q\}_{q=1}^Q)\mathbf{x}, \quad (52)$$

where $\{h_{qk}\}$ is the parameter applied to the kth signal shift with respect to the qth GSO. The multiple GSOs now act as inductive biases about the graph and / or data to aid modeling. In contrast to e.g., learning the GSOs, this approach reduces the filter parameters to $Q(K+1)$ and the computational cost to $\mathsf{O}(QK|\mathcal{E}|)$. In addition, because these filters are linear in the parameters, their data-driven design reduces to solving a least squares problem, similar to convolutional filtering.

VII. Graph Filter Banks and Wavelets

In many instances, a single graph filter suffices to smooth data, identify discontinuities, or classify a signal. However, the outputs of multiple filters (a *filter bank*) can also be combined to generate more nuanced representations of the data. The combined filter coefficients can serve as feature vectors in machine learning tasks or be leveraged in regularization problems when one has *a priori* modeling information that the graph signal of interest belongs to a class of signals whose filter bank coefficients exhibit specific structural patterns (e.g., they are sparse).

Throughout this section, unless stated otherwise, we assume the underlying graph is undirected and the graph shift operator is Hermitian (including real symmetric).

A. Undecimated Single-Level M-Channel Graph Filter Banks

A single-level graph filter bank without any downsampling (*undecimated*) applies M different filters to a signal $\mathbf{x}$ and concatenates the outputs into a single vector of length MN:

$$\boldsymbol{\alpha} := [\mathbf{H}_1(\mathbf{S})\mathbf{x};\ \mathbf{H}_2(\mathbf{S})\mathbf{x};\ \ldots;\ \mathbf{H}_M(\mathbf{S})\mathbf{x}].$$

An example shown in Fig. 5. When the filters, often called the *analysis filters*, are linear, the graph filter bank constitutes a linear transform from $\mathbb{X}^{\mathcal{V}}$ (the graph signal) to $\mathbb{X}_{MN}$ (the filtered signals). Equivalently, we can interpret each of the MN output coefficients as the inner product between the graph signal $\mathbf{x}$ and a *dictionary atom* of the form $\boldsymbol{\varphi}_{im} := \mathbf{H}_m(\mathbf{S})\boldsymbol{\delta}_i$, where $[\boldsymbol{\delta}_i]_j = 1$ if $j = i$ and 0 otherwise. Each atom $\boldsymbol{\varphi}_{im}$ can be viewed as a pattern defined through the filter $\tilde{\mathbf{h}}_m$ in the spectral domain and then centered at vertex i (cf. [30, Fig. 1]).

The most common method to reconstruct the signal from the output coefficients is through a synthesis filter bank. For an undecimated single-level M-channel graph filter bank, shown in Fig. 5, the reconstructed signal is given by

$$\mathbf{x}_{\mathbf{rec}} = \sum_{m=1}^{M} \mathbf{G}_m(\mathbf{S})\mathbf{H}_m(\mathbf{S})\mathbf{x}, \quad (53)$$

where $\{\mathbf{G}_m(\mathbf{S})\}$ are the *synthesis filters*.

Parseval frames. The dictionary atoms $\{\boldsymbol{\varphi}_{im}\}_{i=1,2,\ldots,N;m=1,2,\ldots,M}$ form a *tight Parseval frame* if $\sum_{m=1}^{M}\sum_{i=1}^{N}|\langle\mathbf{x},\boldsymbol{\varphi}_{im}\rangle|^2 = ||\mathbf{x}||_2^2$. A sufficient condition for these atoms to form a tight Parseval frame is that

$$\sum_{m=1}^{M}[\tilde{\mathsf{h}}_m(\lambda_i)]^2 = 1, \text{ for each } i = 1,2,\ldots,N; \quad (54)$$

that is, the chosen filters cover the entire spectrum evenly in the sense that the sums of their squared values are the same at every eigenvalue. Benefits of designing the filters to meet this condition include (i) $||\mathbf{x}||_2 = ||\boldsymbol{\alpha}||_2$; i.e., the filter bank preserves the energy of the signal, which also helps avoid numerical instabilities; and (ii) using the same filters for the synthesis filter bank as the analysis filter bank (i.e., $\mathbf{G}_m(\mathbf{S}) = \mathbf{H}_m(\mathbf{S})$ for all m) results in perfect reconstruction of the signal, because

$$\sum_{m=1}^{M}\mathbf{H}_m(\mathbf{S})\mathbf{H}_m(\mathbf{S})\mathbf{x} = \mathbf{V}\left[\sum_{m=1}^{M}[\mathsf{diag}(\tilde{\mathbf{h}}_m)]^2\right]\mathbf{V}^{\mathsf{H}}\mathbf{x} = \mathbf{x}.$$

Examples of such tight spectral graph filter frames include those constructed and investigated in [117]–[120].

As discussed in Sec. V, using polynomial filters circumvents the need to exactly compute the eigenvectors of $\mathbf{S}$ and also enables local processing. However, [121] shows that it is not possible to design a filter bank comprised of polynomial filters that satisfies $\sum_{m=1}[\tilde{\mathsf{h}}_m(\lambda)]^2 = 1$ for all $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ (the desired condition for a graph-independent tight frame guarantee since the idea is to not compute all of the eigenvalues). References [121]–[123] explore different methods to design

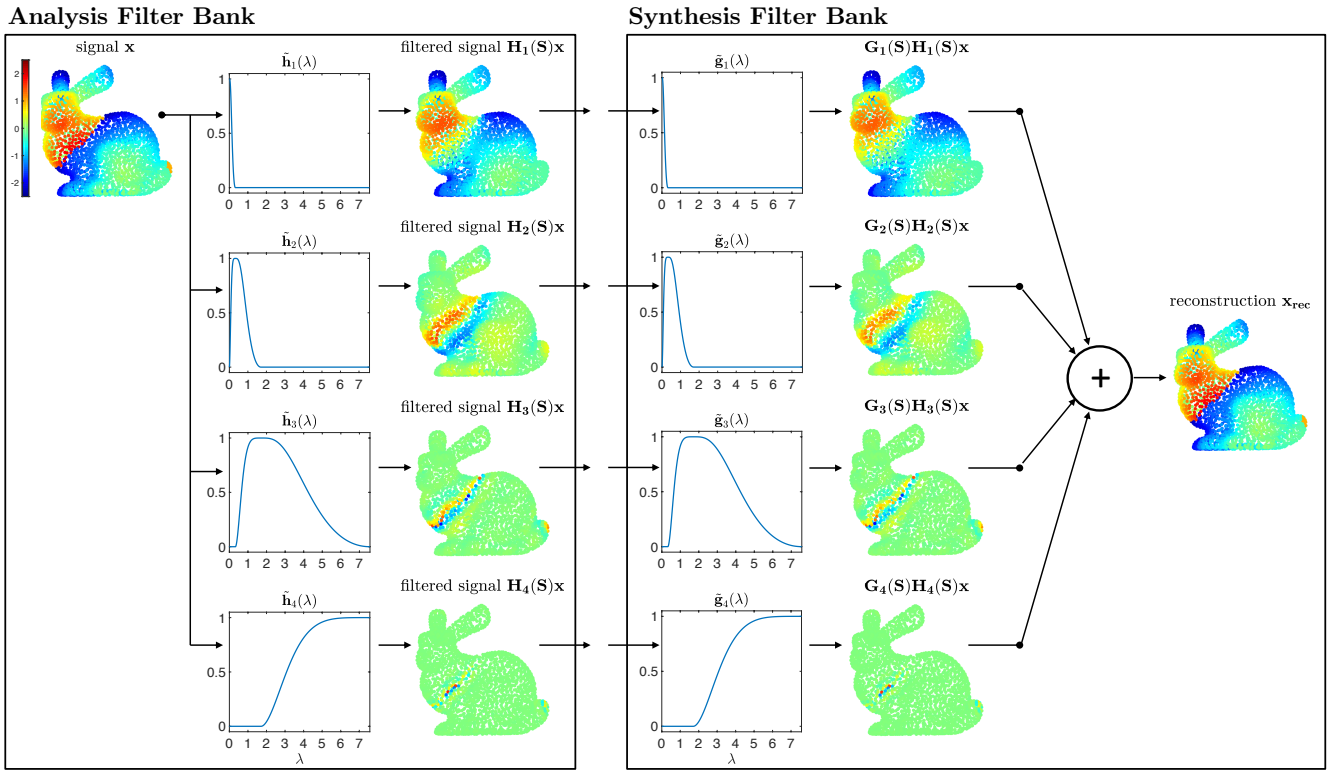


Fig. 5: An undecimated single-level four-channel graph filter bank. The signal is piecewise smooth with respect to the Stanford bunny graph [116]. As a result, the non-zero coefficients of the bandpass and highpass channels ($m = 2, 3, 4$) cluster around the two discontinuities at the midsection and tail of the bunny. The synthesis filters $\{\tilde{\mathsf{g}}_m\}$ used here are the same as the analysis filters $\{\tilde{\mathsf{h}}_m\}$, although they do not need to be in general. The filters $\{\tilde{\mathsf{h}}_m\}$ chosen for this example with the design method of [117] satisfy the tight Parseval frame condition (54), leading to perfect reconstruction.

polynomial filter banks that approximately satisfy the tight frame condition, while [124] allows the polynomial synthesis filters $\{\tilde{\mathsf{g}}_m\}$ to be different from the polynomial analysis filters $\{\tilde{\mathsf{h}}_m\}$, and outlines a method to design the filters to satisfy $\sum_{m=1}^{M} \tilde{\mathsf{h}}_m(\lambda)\tilde{\mathsf{g}}_m(\lambda) = 1$, guaranteeing perfect reconstruction.

Spectral graph wavelets. A seminal example of undecimated graph filter banks are *spectral graph wavelets*, introduced in [125] and later extended to tight frames [117]–[120]. Analogous to wavelet filter banks for discrete-time signals (see, e.g., [126]), choosing the filters to be dilated versions of each other with wider support in the highpass filters at the upper end of the spectrum yields atoms that are increasingly (as the filters become more dilated) localized in the vertex domain. As a result, the spectral graph wavelet filter bank coefficients are sparse for signals that are smooth or piecewise smooth with respect to the underlying graph [125], [127]. This phenomenon is illustrated in Fig. 5, where the coefficients in the bandpass and highpass filters (channels $m = 2, 3, 4$) are (i) close to 0 except around the discontinuities in the piecewise smooth signal, and (ii) increasingly sparse at higher scales (larger m). Spectral graph wavelets have been applied in community mining [128], mobility pattern analysis [129], semi-supervised learning [130], 3D action recognition from depth cameras [131], fMRI data analysis [132], and network topology analysis [133].

B. Downsampling and Critically-Sampled Graph Filter Banks

Without any downsampling, the M-channel graph filter bank is a redundant transform. In many applications, this is just fine and the fact that the output coefficients are sparse for specific classes of signals can be leveraged in regularization and machine learning problems. In some applications, it is desirable to subsample the output coefficients, keeping only those associated with the vertices in the set $\mathcal{V}_m$ at the mth channel, reducing the overall storage cost. When $\sum_{m=1}^{M} |\mathcal{V}_m| = N$, the filter bank is said to be *critically sampled* [134]. With a typical synthesis filter bank comprised of upsampling the output coefficients from each channel, filtering, and summing, the reconstructed signal is given by (cf. (53) for the effect of the downsampling and upsampling):

$$\mathbf{x}_{\mathbf{rec}} = \sum_{m=1}^{M} \mathbf{G}_m(\mathbf{S})\mathbf{M}_{\mathcal{V}_m}^{\top}\mathbf{M}_{\mathcal{V}_m}\mathbf{H}_m(\mathbf{S})\mathbf{x}, \tag{55}$$

where $\mathbf{M}_{\mathcal{V}_m}$ is a $|\mathcal{V}_m| \times N$ selection matrix with $[\mathbf{M}_{\mathcal{V}_m}]_{k,i} = 1$ if vertex i is the kth element of $\mathcal{V}_m$ and 0 otherwise, and $\mathbf{M}_{\mathcal{V}_m}^{\top}$ is the corresponding upsampling operator.

Perfect reconstruction. While [135] investigates how to select the sampling sets $\{\mathcal{V}_m\}_m$ to minimize the reconstruction error $||\mathbf{x}_{\mathbf{rec}} - \mathbf{x}||_2$ for a fixed choice of filters, a broader question is whether it is possible to jointly select the filters $\{\mathbf{H}_m(\mathbf{S})\}_m$ and $\{\mathbf{G}_m(\mathbf{S})\}_m$ and the sampling sets $\{\mathcal{V}_m\}_m$ to recover the original signal $\mathbf{x}$ exactly from the subsampled outputs $\{\mathbf{M}_{\mathcal{V}_m}\mathbf{H}_m(\mathbf{S})\mathbf{x}\}_m$.

Indeed, when the underlying graph has special structural properties, it is possible to guarantee perfect reconstruction. For example, when the graph is bipartite and $\mathbf{S} = \mathbf{L}_{\mathrm{n}}$, the spectrum of normalized Laplacian eigenvalues (contained in $[0, 2]$) is symmetric around $\lambda = 1$ and the eigenvectors associated with eigenvalues λ and $2 - \lambda$ are closely related, leading to a *spectral folding* effect analogous to aliasing in one-dimensional signal

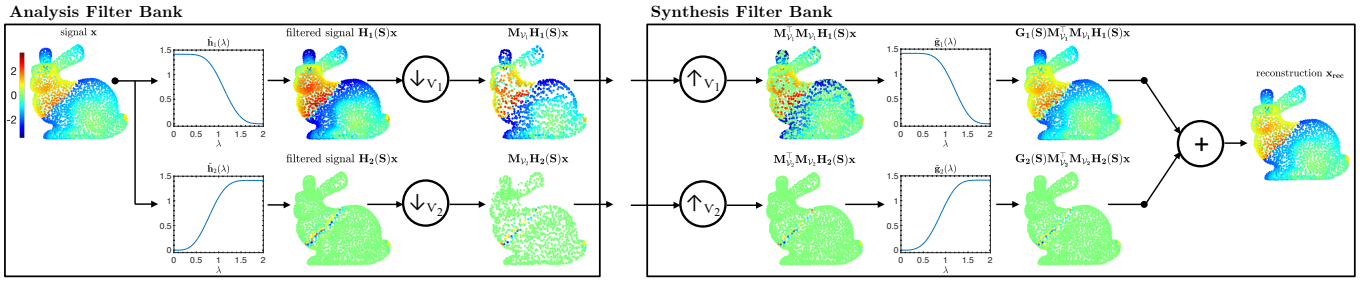


Fig. 6: A single-level critically-sampled two-channel generalized graph filter bank for the same signal shown in Fig. 5. This filter bank combines the perfect reconstruction biorthogonal filters of [136, Ex. 3] with the generalized filter bank approach of [137] for arbitrary graphs. The graph is partitioned into two approximately equal-sized complementary sets of vertices, $\mathcal{V}_1$ and $\mathcal{V}_2$. Here, $\bar{\mathbf{S}} = \mathbf{L}$, $\mathbf{Q} = \begin{bmatrix} [\mathbf{L}]_{\mathcal{V}_1,\mathcal{V}_1} & \mathbf{0} \\ \mathbf{0} & [\mathbf{L}]_{\mathcal{V}_2,\mathcal{V}_2} \end{bmatrix}$, $\mathbf{S} = \mathbf{Q}^{-1}\mathbf{L}$ (not Hermitian in general), and $\mathbf{H}_i(\mathbf{S}) = \mathbf{V}\mathsf{h}_i(\mathbf{\Lambda})\mathbf{V}^{-1} = \mathbf{V}\mathsf{h}_i(\mathbf{\Lambda})\mathbf{V}^H\mathbf{Q}$. Although they look similar in shape, the synthesis filters are not the same as the analysis filters.

processing. Specifically, with $M = 2$ and the downsampling sets selected according to the bipartition $\{\mathcal{V}_1, \mathcal{V}_2\}$, for each eigenvalue and $m = 1, 2$, we have

$$\mathbf{\Gamma}_\lambda\left(\mathbf{M}_{\mathcal{V}_m}^\top\mathbf{M}_{\mathcal{V}_m}\mathbf{x}\right) = \frac{1}{2}\left[\mathbf{\Gamma}_\lambda(\mathbf{x}) + \mathbf{J}_{\mathcal{V}_m}\mathbf{\Gamma}_{2-\lambda}(\mathbf{x})\right], \tag{56}$$

where $\mathbf{\Gamma}_\lambda$ performs an orthogonal projection of a vector onto the eigenspace associated with eigenvalue λ, and $\mathbf{J}_{\mathcal{V}_m} = 2\mathbf{M}_{\mathcal{V}_m}^\top\mathbf{M}_{\mathcal{V}_m} - \mathbf{I}_N$. A key takeaway from (56) is that the portion of the downsampled and upsampled signal in the eigenspace associated with λ only depends on the portions of the original signal in the eigenspaces associated with λ and $2-\lambda$. Ref. [138] shows that for this case, the following two conditions are necessary and sufficient for perfect reconstruction

$$\begin{aligned} \tilde{\mathsf{g}}_1(\lambda)\tilde{\mathsf{h}}_1(\lambda) + \tilde{\mathsf{g}}_2(\lambda)\tilde{\mathsf{h}}_2(\lambda) &= 2, \\ \tilde{\mathsf{g}}_1(\lambda)\tilde{\mathsf{h}}_1(2-\lambda) - \tilde{\mathsf{g}}_2(\lambda)\tilde{\mathsf{h}}_2(2-\lambda) &= 0. \end{aligned}$$

Leveraging these conditions, [136], [138]–[142] design two-channel critically-sampled perfect reconstruction graph filter banks.

Other special types of graph with structural properties that can be leveraged to generate critically-sampled perfect reconstruction graph filter banks include shift-invariant graphs that have a circulant graph Laplacian [143]–[145] and M-block cyclic graphs [146]–[148].

The generalized critically-sampled filter banks of [137] extend the spectral folding idea from bipartite graphs to arbitrary graphs. They do this by taking the filtering basis vectors $\{\bar{\mathbf{v}}_i\}_{i=1,2,\ldots,N}$ to be the solutions to the *generalized* eigenvalue problem

$$\bar{\mathbf{S}}\bar{\mathbf{v}}_i = \bar{\lambda}_i\mathbf{Q}\bar{\mathbf{v}}_i, \tag{57}$$

so that $\bar{\mathbf{v}}_i^{\mathsf{H}}\mathbf{Q}\bar{\mathbf{v}}_j = 0$ for $i \neq j$; i.e., the filtering basis is orthonormal with respect to the inner product $\langle\bar{\mathbf{v}}_i, \bar{\mathbf{v}}_j\rangle_{\mathbf{Q}} := \bar{\mathbf{v}}_j^{\mathsf{H}}\mathbf{Q}\bar{\mathbf{v}}_i$ instead of the standard dot product. In (57), $\bar{\mathbf{S}}$ is a Hermitian (including real symmetric) positive semi-definite matrix with off-diagonal sparsity pattern matching the adjacency matrix (e.g., a Laplacian). If, for any partition $\{\mathcal{V}_1, \mathcal{V}_2\}$ of the vertices, $\mathbf{Q}$ is selected to be equal to $\begin{bmatrix} [\bar{\mathbf{S}}]_{\mathcal{V}_1,\mathcal{V}_1} & \mathbf{0} \\ \mathbf{0} & [\bar{\mathbf{S}}]_{\mathcal{V}_2,\mathcal{V}_2} \end{bmatrix}$, then a spectral folding property analogous to the one for bipartite graphs holds (but this time for arbitrary graphs), leading to perfect reconstruction. Fig. 6 shows an example of such a critically-sampled two-channel generalized graph filter bank.

Orthogonality and biorthogonality. A critically-sampled filter bank is said *orthogonal* if $\sum_{m=1}^M \mathbf{H}_m(\mathbf{S})\mathbf{M}_{\mathcal{V}_m}^\top\mathbf{M}_{\mathcal{V}_m}\mathbf{H}_m(\mathbf{S}) = \mathbf{I}_N$, in which case selecting the synthesis filters to be the same as the analysis filters leads to perfect reconstruction (cf. (55)), and is said to be *biorthogonal* if $\sum_{m=1}^M \mathbf{G}_m(\mathbf{S})\mathbf{M}_{\mathcal{V}_m}^\top\mathbf{M}_{\mathcal{V}_m}\mathbf{H}_m(\mathbf{S}) = \mathbf{I}_N$, again guaranteeing perfect reconstruction. Refs. [136], [138], [139], [141], [149]–[151] examine orthogonal, near orthogonal, and biorthogonal filter designs for critically-sampled filter banks on bipartite graphs. The primary motivation for using biorthogonal filters with bipartite graphs is that it is impossible to choose polynomial filters that yield an orthogonal filter bank [139].

C. Alternative Structures for Arbitrary Graphs

A number of alternative structures for perfect signal reconstruction on arbitrary graphs have also been proposed:

1) graph extensions of lifting transforms [152]–[155], pyramid transforms [156], and oversampled filter banks [157], [158];
2) subgraph-based filter banks for graph signals [159] where the downsampling is performed by partitioning the graph into connected subsets of vertices and representing each subset by a single supernode;
3) filter banks where the synthesis portion (upsampling and filtering) is replaced with a different interpolation operator [160], [161];
4) filter banks where the downsampling is performed in the graph spectral domain instead of in the vertex domain [162]–[164];
5) filter banks that first replace the arbitrary underlying graph by a maximum spanning tree [165], [166];
6) multi-dimensional separable filter banks that first decompose an arbitrary graph into sums of bipartite graphs [138]–[140];
7) filter banks that first decompose an arbitrary graph into sums of circulant graphs [143]–[145];
8) filter banks that work with a similarity-transformed adjacency matrix [147].

D. Multi-Level Graph Filter Banks

In classical multi-level filter banks for time series data or images, multiple levels of filtering and downsampling are applied. For example, in the classical logarithmic wavelet filter bank, at each level, another filter bank is applied to the downsampled output of the lowpass channel from the previous level [126]. Numerous works have investigated extensions to multi-level filter banks, lifting transforms, and pyramids for graph signals (e.g., [137], [138], [143], [144], [156], [159], [162]). In classical time series analysis or image processing, the structure of the underlying domain enables regular sampling (e.g., every other

time sample) that preserves the notion of frequency entailed by filtering at each level of the multi-level filter bank. One main difference and significant challenge in the graph setting is that – unless the graph is highly symmetric – it is not obvious how to define a coarser graph at each subsequent level of the filter bank in a way that maintains a clear correspondence between the eigenvectors of the shift operator that are used for graph filtering at one level, and the eigenvectors of the shift operator on the coarsened graph that are used for filtering the downsampled signal on that coarsened graph (c.f., [167]–[169]).

E. Data-Adapted Transforms / Dictionary Learning

All of the design elements discussed so far – the graph(s), filters, and downsampling sets – can be adapted either to the specific graph signal being analyzed or to an additional set of representative training signals. For example, [170] presents a method to learn polynomial filters that yield sparse representations of the training signals and [171] presents a method to learn filters that yield a tight frame with each resulting filter subband capturing the same amount of energy on average across the training signals. This approach is particularly beneficial when the energy of a typical signal from the class of interest is concentrated on a small region of the spectrum, which the authors show is the case for brain fMRI data [171]. Meanwhile, [172] is just one of many examples of constructing the underlying graph from the signal, in this case for the purpose of image coding.

VIII. Graph Neural Networks

Graph neural networks (GNNs) are nonlinear layered architectures, in which each layer comprises a bank of graph filters (Sec. VII) and an activation function that is (typically) pointwise and nonlinear [4], [23], [173]. This nonlinear nature allow us to capture more complex relationships than the linear graph filters, and their compositional form allows for a sequential extraction of features, typically enhancing representational capabilities over simpler nonlinear graph filters.

The basic building block of GNNs is the *graph perceptron*, which is a straightforward extension of graph filters [23].

Graph perceptron. A graph perceptron is a nonlinear mapping comprising a linear graph filter $\mathsf{H}(\mathbf{x})$ nested into an activation function (a pointwise nonlinear function) $\sigma : \mathbb{R} \to \mathbb{R}$, i.e.,

$$\mathbf{y} = \sigma(\mathsf{H}(\mathbf{x})) \tag{58}$$

where $\sigma(\mathbf{x})$ signifies $[\sigma(\mathbf{x})]_i = \sigma([\mathbf{x}]_i)$.

Graph perceptrons can be built using any of the filters reported in Table I and the activation function can take different forms e.g., $\sigma(x) = \text{ReLU}(x) = \max\{0, x\}$ or hyperbolic tangent $\sigma(x) = \tanh(x)$. If we let $\mathsf{H}(\mathbf{x})$ be a convolutional filter of the form $\mathsf{H}(\mathbf{x}) = h_1\mathbf{S}\mathbf{x}$, the graph perceptron becomes the usual expression of GCNs given by $\mathbf{x}_1 = \sigma(h_1\mathbf{S}\mathbf{x}_0)$, where h_1 is the learnable coefficient. Extension to multi-featured graph signals comes in (62). Cascading graph perceptrons gives rise to a graph neural network (GNN) [23]. Formally, a GNN $\Phi : \mathbb{X}^{\mathcal{V}} \to \mathbb{X}^{\mathcal{V}}$ comprising L layers is given by

$$\Phi(\mathbf{x}) = \mathbf{x}_L \text{ where } \mathbf{x}_\ell = \sigma\big(\mathsf{H}_\ell(\mathbf{x}_{\ell-1})\big),\ \ell = 1, \ldots, L \tag{59}$$

with $\mathbf{x}_0 = \mathbf{x}$. That is, the input to the GNN is a graph signal that is processed by a graph perceptron to form the output of layer $\ell = 1$, i.e., $\mathbf{x}_1 = \sigma(\mathsf{H}_1(\mathbf{x}_0))$. Signal $\mathbf{x}_1$ is the input of the next layer and it is processed by another graph perceptron to output $\mathbf{x}_2 = \sigma(\mathsf{H}_2(\mathbf{x}_1))$. This procedure is repeated for all layers, and the GNN output is that of the last layer, $\mathbf{x}_L$; see Fig. 7.

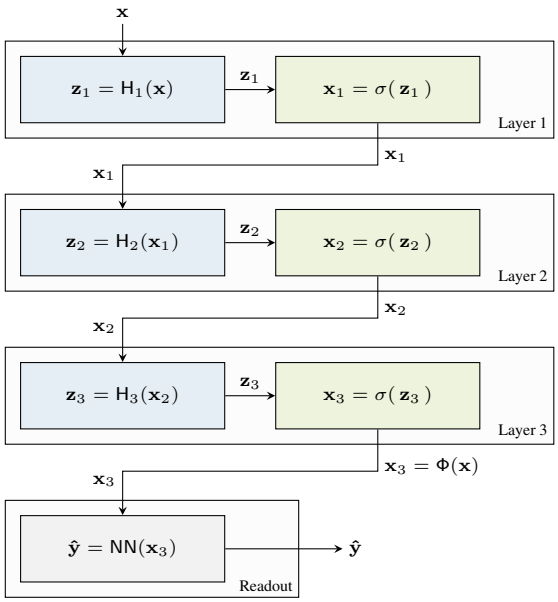


Fig. 7: Schematic for a GNN with 3 layers. The input $\mathbf{x}$ is processed by a graph filter H_1 and then an activation function σ. The output of this block – a graph perceptron – is fed into another graph perceptron corresponding to layer 2. The output of the GNN is the output of the third, cascaded graph perceptron. Each layer has a different filter whose coefficients are learned from data. If required for the problem, the output of the last layer of the GNN, in this case $\Phi(\mathbf{x}) = \mathbf{x}_3$ can be fed into a readout layer to finally compute the target value $\hat{\mathbf{y}}$. Depending on the nature of this readout layer, the distributed nature of the GNN may be violated. See paragraph on 'Readout Layer' for more details.

The graph filters incorporate the topology of the data structure, and these filters are dependent on the parameters at each layer. Grouping all filter parameters in the set $\mathcal{H}$, we estimate $\mathcal{H}$ in a data-driven fashion from a training set $\mathcal{T} = \{\mathbf{x}_i\}_{i=1}^{|\mathcal{T}|}$ by minimizing a task-dependent cost function $\mathsf{J} : \mathbb{R}^N \to \mathbb{R}$:

$$\min_{\mathcal{H}} \sum_{\mathbf{x}_i \in \mathcal{T}} \mathsf{J}\big(\Phi(\mathbf{x}_i)\big). \tag{60}$$

In general, it is assumed the samples in $\mathcal{T}$ are independent, identically distributed, and thus (60) becomes an empirical risk minimization problem [174]. For a supervised learning setting, we have output samples $\mathbf{y}_i$ for the training data $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{|\mathcal{T}|}$ and thus the objective function in (60) becomes $\mathsf{J}(\Phi(\mathbf{x}_i), \mathbf{y}_i)$. For semi-supervised learning – e.g., node classification, where we have $\mathcal{T} = \{\mathbf{x}, \bar{\mathbf{y}}\}$ with inputs $\mathbf{x}$ typically available for all nodes and output $\bar{\mathbf{y}}$ available only at a subset of nodes – the i.i.d. assumption on the samples does not hold. The objective is not necessarily to find the filter parameters $\mathcal{H}$ that minimize $\mathsf{J}(\cdot)$, but rather to take gradient descent steps that would improve the generalization performance on unseen data; see [2, Ch. 8] for more details on training neural networks. GNNs have taken over as a very powerful and promising tool in machine learning, with notable applications in recommender systems [175], drug discovery [176], biology [177], [178], and time of arrival prediction [179].

Choosing the form of filters H_ℓ determines the overall GNN characteristics [27]. In the following, we discuss the convolutional filters (Sec. VIII-A) and the non-convolutional filters (Sec. VIII-B). We close with a brief overview of other uses of graph filters in GNN-style architectures (Sec. VIII-C).

Multiple features. The representation power of the GNN in (59) can be increased by utilizing a bank of filters (see Sec. VII)

instead of a single filter [27]. To see this, consider that the input graph signal $\mathbf{x} = \mathbf{x}_0$ gets processed by F_1 different graph filters $\{\mathsf{H}_1^f\}_{f=1}^{F_1}$, creating a set of F_1 output graph signals $\mathcal{X}_1^{F_1} = \{\mathbf{x}_1^1, \ldots, \mathbf{x}_1^{F_1}\}$ after applying the activation function to each of them, i.e. $\mathbf{x}_1^f = \sigma(\mathsf{H}_1^f(\mathbf{x}))$. At the next layer, we have a set of F_1 input graph signals, instead of just a single one. If we want to use a bank of filters again, the most general linear operation would be to use a distinct bank for each of the input graph signals. That is, if we want F_2 output graph signals, we need F_2 filters $\{\mathsf{H}_2^{1f}, \ldots, \mathsf{H}_2^{F_2 f}\} = \{\mathsf{H}_2^{gf}\}_{g=1}^{F_2}$ for each input graph signal $f \in \{1, \ldots, F_1\}$. Doing so creates a set of $F_2 F_1$ graph signals, each one obtained as $\mathbf{x}_2^{gf} = \sigma(\mathsf{H}_2^{gf}(\mathbf{x}_1^f))$. To prevent the number of signals from growing exponentially, a summary is created by adding up the signals resulting from each of the gth filters, i.e. $\mathbf{x}_2^g = \sum_{f=1}^{F_1} \mathbf{x}_1^f$. In this way, we can think of the layer as taking F_1 input signals, and giving F_2 output signals. Repeating this for every layer, we can think of taking $F_{\ell-1}$ input signals, and giving F_ℓ output signals, which leads to a generic description of GNNs as follows

$$\Phi(\mathcal{X}^F) = \mathcal{X}_L^{F_L} \text{ where } \mathbf{x}_\ell^g = \sigma\Big(\sum_{f=1}^{F_{\ell-1}} \mathsf{H}_\ell^{gf}(\mathbf{x}_{\ell-1}^f)\Big), \tag{61}$$

for $g = 1, \ldots, F_\ell$ at every layer $\ell = 1, \ldots, L$, and where $\mathcal{X}_\ell^{F_\ell} = \{\mathbf{x}_\ell^1, \ldots, \mathbf{x}_\ell^{F_\ell}\}$ is the set of F_ℓ features at layer ℓ. The resulting filter bank can be interpreted as an undecimated, analysis filter bank (Sec. VII), where the filter coefficients are learned from data instead of being designed. Understanding each layer as a learnable filter bank may allow us to impose certain characteristics, such as Parseval tight frames (cf. (54)), during design or training. The values of $\{F_\ell\}$ and L are hyperparameters, and are often used as a proxy for representational capability (see [2, Ch. 5] for the relationship between capacity, width, and generalization).

Readout layer. The GNN output $\Phi(\mathcal{X}^F) = \mathcal{X}_L^{F_L}$ (cf. (61)) at each vertex is a vector of dimension F_L. Thus, the dimensions of the GNN output may not match the dimensions of the target output $\mathbf{y}$. A readout layer is therefore used to match the dimensions and decode the GNN encoded embeddings into the final output, see [2, Ch. 9]. Depending on whether we use the GNN for centralized or distributed processing, the readout layer has different forms. In a centralized processing, all node features are usually concatenated into a vector of size NF_L $\mathbf{x}_{\text{GNN}} = [(\mathbf{x}_L^1)^\mathsf{T}, \ldots, (\mathbf{x}_L^{F_L})^\mathsf{T}]^\mathsf{T}$ and then mapped to the output dimension as per e.g., the linear transform $\mathbf{u} = \boldsymbol{\Theta}\mathbf{x}_{\text{GNN}}$, where the matrix of parameters $\boldsymbol{\Theta}$ matches the output dimensions. Instead, in distributed processing,[2] the readout layer must also be local. One conventional case is to consider a readout layer operating on a single node. That is, let vector $\boldsymbol{\chi}_i = [[\mathbf{x}_L^1]_i, \ldots, [\mathbf{x}_L^{F_L}]_i]^\mathsf{T} \in \mathbb{R}^{F_L}$ be the vector of GNN output features at node i and suppose the target output is a real scalar. Then, the readout layer at node i is of the form $u_i = \boldsymbol{\theta}^\top \boldsymbol{\chi}_i$, where vector $\boldsymbol{\theta} \in \mathbb{R}^{F_L}$ is common for all nodes. The readout layers can also be nonlinear multi-layer perceptron layers. In either case, they are trainable parameters and are used to minimize cost (60) or alternative objective functions.

Pooling. Pooling is included in regular CNNs to construct regional summaries of information. This mainly serves two objectives: (i) control the computational cost by trading spatial information with feature information (i.e., reducing the size of the images while increasing the number of features); (ii) aggregate global information in the deeper CNN layers. Pooling approaches have also been developed for GNNs and can be interleaved with the graph perceptron layers [180]. These also follow two different lines: (i) use some sort of multiscale hierarchical clustering algorithm [181], creating ever smaller graph supports at each layer; or (ii) use graph sampling methods [182] that leave the graph topology unaltered. The former is typically of more interest in abstract networks where the graph supports can be manipulated. The latter is typically of more interest for physical networks and distributed processing where we want to use the original topology to process signals in deeper layers. However, in many applications such as those involving physical graphs (robotic, sensor, or power grid networks), nodes have computational power and thus the cost of computing the GNN output is naturally distributed among these nodes. In these cases, pooling is less crucial and may not be needed.

[2]GNNs are distributed architectures if the graph filters are distributable.

A. Graph Convolutional Neural Networks

The most popular GNN architectures are those that use a GCF at each layer; i.e., substitute H_ℓ in (59) with (3). This leads to graph convolutional neural networks (GCNN) [181]–[183]. The GCNN can be compactly written as [26]

$$\Phi(\mathbf{X}) = \mathbf{X}_L \text{ where } \mathbf{X}_\ell = \sigma\Big(\sum_{k=1}^{K} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k}\Big), \tag{62}$$

where $\mathbf{X}_\ell \in \mathbb{R}^{N \times F_\ell}$ collects the F_ℓ graph signal features $\mathbf{x}_\ell^g$ obtained at the output of layer ℓ, and $\mathbf{H}_{\ell k} \in \mathbb{R}^{F_\ell \times F_{\ell-1}}$ contains the kth filter parameters of the $F_\ell F_{\ell-1}$ filters involved in (61); i.e. $[\mathbf{H}_{\ell k}]_{gf} = h_{\ell k}^{gf}$ for $f = 1, \ldots, F_{\ell-1}$ and $g = 1, \ldots, F_\ell$. The multiplications $\mathbf{S}^k$ on the left of $\mathbf{X}_{\ell-1}$ shift the different signals locally over the graph up to k hops away, whereas the multiplications on the right carry out a linear combination of values contained in the same node via the filter bank coefficients, and as such, can be arbitrary (which is the case when $\mathbf{H}_{\ell k}$ is learned from data).

This structure, coupled with the pointwise nature of the activation function, makes the GCNN a local architecture that respects Properties 3-6 and Property 8 of the graph convolutional filter [56]. GCNNs are also Lipschitz continuous to changes in the underlying graph support (cf. (14)), albeit with a slightly modified constant. They can, however, process information located in large GSO eigenvalues in a stable manner, a feat that cannot be achieved by linear graph convolutions (see [56]). This makes GCNNs better suited for problems in which information located at large GSO eigenvalues is important.

Implementations. While it is perfectly feasible to implement GCNNs via (62), different (sub-)implementations, often derived from a different stating point, have become popular. These include:

1) GCNNs with orthogonal polynomials such as Chebyshev [181], [184], Berstein [185], and Jacobi [186].
2) The GCN of [187] uses in (62) $\mathbf{S} = \mathbf{D}^{-1/2}(\mathbf{I} + \mathbf{A})\mathbf{D}^{-1/2}$, $K = 1$ and, more crucially, $\mathbf{H}_{\ell 0} = \mathbf{0}$ for all ℓ. This forces all the learned filters to be low-pass filters leading to the *oversmoothing* problem so thoroughly discussed in the GCNN literature [188], [189].
3) A Simplifying Graph Convolutional (SGC) Network [190] further exacerbates this problem by setting $\mathbf{H}_{\ell k} = \mathbf{0}$ for all $k < K$ for some order K.

4) A Graph Isomorphism Network (GIN) [191] can be obtained from (62) by setting $\mathbf{S}$ to be the binary adjacency matrix, $K = 1$, and $\mathbf{H}_{\ell 0} = (1 + \varepsilon_\ell)\mathbf{H}_{\ell 1}$ for some pre-defined ε_ℓ. GINs further propose to use $K = 0$ for some intermediate layers to mimic a node-based multi-layer perceptron (MLP).
5) GraphSAGE [192] with a linear 'aggregate' function is obtained from (62) by setting $K = 1$ and then normalizing feature-wise the resulting graph signal. While GraphSAGE does not suffer from oversmoothing, forcing $K = 1$ precludes sharp transitions.
6) A Jumping Knowledge Network (JKNet) [193] with a summation aggregation can be seen as computing the GCNN operation (62) where residual connections are used to account for multi-hop neighbors.

The design of GNN architectures is evolving at a fast pace and thus many newer architectures become readily available each month. While we have only mentioned the most popular ones, we would like to encourage readers to identify whether these new architectures are convolutional in nature, and thus fit the description in (62), as the ones above, or they are essentially non-convolutional and may be equivalently described by leveraging other filter structures as we discuss in the next section.

B. Non-convolutional GNNs

In principle, we can build a different GNN architecture by exchanging the filters in (61) with any of the types discussed in Sec. VI. These GNNs will exploit different aspects of the data structure, closely following the properties that the chosen graph filters themselves exhibit.

Rational graph filters (Sec. VI-A) lead to GNNs that are convolutional in practice, but capable of achieving much sharper frequency transitions with fewer learnable parameters. Cayleynets [78] and ARMANets [27] are two such examples.

Node-varying graph filters (32) lead to non-convolutional GNNs [94] as a means of learning frequency content creation (refer to [56] for a thorough discussion on the effects of having new frequencies at the otuput). Edge-varying graph filters (34) also lead to non-convolutional GNNs [27], with graph attention transformers (GATs, [194]) and natural graph convolutions [195] being two of the most popular exponents. The edge varying filter has been used here to propose a broad family of GNN solutions as a way to show benefits and limitations of the different architectures and how they trade parameter sharing with permutation equivariance. In fact, neither the node varying nor the edge varying GNNs are permutation equivariant architectures. This is particularly useful in applications where nodes are distinguishable. For example, in power grids, some nodes represent generators while others represent consumers, and thus it may be useful that they learn different behaviors.

C. Other Uses of Filtering in GNNs

Graph filters also play other key roles in GNNs. For instance, graph wavelets (Sec. VII) can be used in lieu of filters in (61) to avoid training procedures (cf. (60)) and to gain interpretability. The resulting architectures are known as graph scattering transforms [196], [197] and have been used successfully in biological applications [198] and 3D point clouds [199], where data is scarce or the training process is computationally intensive.

Nonlinear graph filters, such as max or median filters (40), can be used as local activation functions (instead of pointwise ones). They preserve permutation equivariance (Property 3), while achieving a higher expressive power. Using these filters also implies that the activation functions are learnable [98], [99].

IX. Applications in Signal Processing

Graph filters have found widespread use in several signal processing application areas. These include the standard problems of graph signal reconstruction from partial and noisy observations (Sec. IX-A), anomaly detection over networks (Sec. IX-B), and network topology inference (Sec. IX-C). Graph filters are also key components of many recently developed graph-based image processing methods (Sec. IX-D). Lastly, due to their local implementation, graph filters have been used for distributed signal processing tasks (Sec. IX-E).

A. Signal Reconstruction

This task consists of reconstructing graph signals from one or more noisy (and possibly partial) observations. Filtering by regularization [Sec. VI-D] has been extensively used for this task, and different regularizers have been developed to match signal priors in different settings. A second strategy is to fit the observed signals with a graph filter and use this filter to reconstruct the missing values. This strategy is first discussed in [21] and subsequently extended to the various graph filters discussed in Sec. VI. These techniques are applied in sensor networks [32] and speech enhancement [200], among others. A third strategy to reconstruct graph signals is to represent them as sparse linear combinations of atoms of an overcomplete graph-based dictionary [170]; that is, write a signal as $\mathbf{x} = \mathbf{D}_\mathcal{G}\mathbf{s}$, where $\mathbf{D}_\mathcal{G} \in \mathbb{R}^{N \times NS}$ is the graph-based dictionary and $\mathbf{s} \in \mathbb{R}^{NS}$ is a sparse vector. Graph filters are used to define the atoms of the dictionary, as $\mathbf{D}_\mathcal{G} = [\mathbf{H}_1(\mathbf{S}), \ldots, \mathbf{H}_S(\mathbf{S})]$, where each $\mathbf{H}_s(\mathbf{S})$ is a graph convolutional filter [170]. The advantage over graph-agnostic dictionaries is that the filter order dictates both the atoms' vertex locality and the number of trainable parameters. Differently, [201]–[203] learn dictionaries where the columns in $\mathbf{D}_\mathcal{G}$ behave as smooth graph signals. Using the Tikhonov regularizer, this boils down to solving versions of

$$\min_{\mathbf{D}_\mathcal{G}, \mathbf{X}} \|\mathbf{Y} - \mathbf{D}_\mathcal{G}\mathbf{X}\|_F^2 + \gamma_1 \mathsf{trace}(\mathbf{D}_\mathcal{G}^\top \mathbf{L} \mathbf{D}_\mathcal{G}) + \gamma_2 \mathsf{trace}(\mathbf{X}^\top \mathbf{L}_c \mathbf{X}) \quad \text{s.t. } \|\mathbf{x}_i\|_0 \leq T \;\forall\; i, \tag{63}$$

where $\mathbf{L}$ and $\mathbf{L}_c$ are Laplacians of two graphs capturing the manifold structure of the dictionary atoms $\mathbf{D}_\mathcal{G}$ and of the data $\mathbf{X}$, respectively. The constraint (63) imposes a maximum sparsity T on each column $\mathbf{x}_i$ of $\mathbf{X}$ and $\gamma_1, \gamma_2 > 0$ control the respective trade-offs. Further, [204] augments problem (63) with graph wavelets to learn multi-scale atoms that facilitate scalability to large graphs. Finally, [205] considers quantization effects on the learned atoms when such dictionaries are used for distributed signal processing tasks.

B. Anomaly Detection

Many graph signals – including, e.g., voltage measurements in power grids [34] or brain imaging recordings in healthy patients [206] – exhibit bandlimited (cf. (8)) and/or low-pass behavior [5], [31]. When an anomaly occurs, these signals contain unexpected components in their high-pass spectrum. We can leverage graph filters to localize such components associated, e.g., with corrupted signals [34] or non-healthy patients [206].

The idea is to design a graph filter $\mathbf{H}(\mathbf{S})$ and form a hypothesis test on the filtered signal $\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x}$ of the form

$$\begin{aligned} \mathcal{H}_0: \quad & \mathsf{f}(\mathbf{y}) \leq \gamma \\ \mathcal{H}_1: \quad & \mathsf{f}(\mathbf{y}) > \gamma \end{aligned}, \tag{64}$$

where $\mathsf{f}(\mathbf{y})$ is a transformation of the filtered output (e.g., $\mathsf{f}(\mathbf{y}) = \|\mathbf{y}\|_2$) and γ is a threshold; that is, the filtered signal shows different characteristics under the null hypothesis $\mathcal{H}_0$ and the alternative hypothesis $\mathcal{H}_1$.

The work in [22] considers such a setting to detect anomalous sensors. The input signal is filtered with a high-pass convolutional filter and the signal is classified as anomalous if one or more GFT coefficients exceed a threshold. References [207]–[209] consider nonlinear filters (Sec. VI-C) to reconstruct the data under normal behavior, and the low-pass signal components are used to detect and localize anomalous sensors. This idea is extended in [210] to identify a cluster of abnormal nodes. The work in [211] proposes an unsupervised setting for the scenario when we do not have knowledge of how normal and/or anomalous graph signals behave. Under the assumption that normal data are more frequent than abnormal ones, the authors use two complementary ideal step graph filters – one low-pass and one high-pass – with the same cut-off frequency, and optimize the cut-off frequency to minimize the cluster size.

In the context of brain imaging, [206] uses similar principles to detect early stage Alzheimer's disease. Two band-pass filters are built (one per type of patient) to localize signal components not belonging to that class. Subsequently, an energy-based Neyman-Pearson detector is derived from the filtered outputs. Reference [212] generalizes this to the setting where information about the alternative hypothesis $\mathcal{H}_1$ is unavailable, leading to a Neyman-Pearson detector only with respect to hypothesis $\mathcal{H}_0$.

C. Network Topology Inference

Often the graph $\mathcal{G}$ is unavailable and, accordingly, network topology inference from a set of (graph signal) measurements is a prominent yet challenging problem. Early foundational contributions can be traced back to the statistical literature of graphical model selection [213], [214]. Recently, the fresh signal representation perspectives offered by GSP through graph filters have sparked renewed interest [41]. At its core, network topology inference assumes some relation between the set of observed graph signals $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m]$ and the GSO $\mathbf{S}$ to be recovered. This relation can be modeled as each $\mathbf{x}_i$ being the output of a graph (convolutional) filter defined on the unknown $\mathbf{S}$. Intuitively, this means that the observations $\mathbf{x}_i$ were generated through (linear) local interactions in the unknown graph, so that the topologocal information of $\mathbf{S}$ is contained in $\mathbf{X}$.

Different assumptions on the filter type generating $\mathbf{X}$ lead to different formulations of the topology inference problem. We can state a generic version of this inverse problem as

$$\min_{\mathbf{S} \in \mathcal{S}} \ \mathsf{f}(\mathbf{X}, \mathbf{H}(\mathbf{S})) + \mathsf{r}(\mathbf{S}), \tag{65}$$

where the fitting loss $\mathsf{f}(\cdot)$ quantifies how well $\mathbf{X}$ can be modeled as the output of a filter $\mathbf{H}(\mathbf{S})$, the regularizer $\mathsf{r}(\cdot)$ promotes desirable properties on $\mathbf{S}$ such as sparsity, and the feasibility set $\mathcal{S}$ encodes the type of GSO that we are looking for (e.g., Laplacian or adjacency matrix).

Smoothness. A first type of (convolutional) graph filter considered in the literature is the low-pass graph filter. This is a reasonable modeling assumption in averaging dynamics such as opinion formation. This model leads to signals $\mathbf{x}_i$ being slow varying, which when $\mathbf{S}$ is the graph Laplacian implies smoothness of $\mathbf{x}_i$ on the unknown graph (cf. (5)). Two early proponents of this model are [215] and [216]. Although their regularizers $\mathsf{r}(\mathbf{S})$ are different, in both cases the fitting term is $\mathsf{f}(\mathbf{X}, \mathbf{S}) = \mathrm{Tr}(\mathbf{X}^\top \mathbf{L}\mathbf{X})$ so that it penalizes graphs not leading to a smooth representation of the observed signals.

Stationarity. Another approach is to not assume any specific form (low-pass, band-pass, high-pass) for the graph filter generating $\mathbf{X}$ [217]. Under certain statistical assumptions on the inputs to the filter, this setting leads to the signals $\mathbf{x}_i$ being graph stationary on the unknown $\mathbf{S}$ [3], [106], [107]; (cf. Sec. VI-D, Wiener filter). In short, this implies that the covariance matrix $\mathbf{\Sigma}_x$ of the observed signals shares the eigenvectors with $\mathbf{S}$, or, equivalently and more practically, that $\mathbf{\Sigma}_x$ and $\mathbf{S}$ commute. Ref. [217] uses a two-step procedure to first estimate the eigenvectors of $\mathbf{S}$ and then restate (65), where only the eigenvalues of $\mathbf{S}$ are unknown. In contrast, the commutativity property can be imposed through a fitting term of the form $\mathsf{f}(\mathbf{X}, \mathbf{S}) = \|\hat{\mathbf{\Sigma}}_x \mathbf{S} - \mathbf{S}\hat{\mathbf{\Sigma}}_x\|_{\mathrm{F}}$, where $\hat{\mathbf{\Sigma}}_x$ is an estimate of the covariance matrix [218]. Other assumptions on the filter include modeling the signal as the superposition of several heat-diffusion filters [219], [220], or as a consensus-like process [221], [222].

D. Image Processing

Graph-based image processing complements conventional image processing approaches with new insights and techniques for tasks such as image reconstruction and filtering [223], [224]. Images have a natural grid structure that can be represented as a graph, where each node is a pixel, an edge connects two pixels, and the graph signal is the pixel intensity. The edge weights can be set via the Gaussian kernels as

$$w_{ij} = \exp\left(- \frac{\|\mathbf{f}_i - \mathbf{f}_j\|_2^2}{\sigma_l^2} \right) \exp\left(- \frac{(x_i - x_j)^2}{\sigma_x^2} \right), \tag{66}$$

where $\mathbf{f}_i$ is the location (feature) of pixel i, x_i its intensity, and σ_l, σ_x are two parameters. Such edge weighs are a combination of the geometric distance (pixels' locations) and photometric distance (signal intensities x_i), where a larger distance implies a smaller weight. Graph-based image processing works mainly with undirected graphs and low-pass filtering since connected pixel nodes have a stronger edge weight if they are close (either geometrically or photometrically). Graph filters are used for image reconstruction (e.g., denoising, debluring) and edge-preserving filtering (i.e., preserve edges appearing in an image, not graph edges).

Image reconstruction. This task consists of reconstructing an image signal $\mathbf{x}$ from a degraded version, which can be noisy, blurred, or have missing pixels. These are all ill-posed inverse problems and regularization is typically used. In the GSP language, this is a graph signal reconstruction task and regularized filtering [Sec. VI-D] is often used to impose low-pass behavior. The Tikhonov regularized filter (42) is leveraged for image denoising in [225]. Reference [226] explores the connection with manifold regularization and provides an explanation why low-pass filtering is particularly useful for denoising depth images. Reference [227] uses a form of the total variation regularizer (46) to denoise the image. Reference [228] approaches the problem from a Wiener filtering perspective (50)-(51). Since regularized filters are particular forms of rational graph filtering, [229] proposes a non-parametric rational filter to denoise the

image. Finally, [16] considers a smoothing graph filter of the form $\mathbf{H}(\mathbf{S}) = e^{-\gamma \mathbf{L}} = \sum_{k=0}^{\infty} \frac{\gamma^k}{k!}(-\mathbf{L})^k$ (a.k.a. the heat kernel) to perform low-pass graph filtering. This can be seen as a convolutional filter of order $K \to \infty$ with frequency response $\tilde{\mathsf{h}}(\lambda) = e^{-\gamma \lambda}$, which for $\gamma > 0$ acts as a low-pass filter.

Edge-preserving filtering. Some conventional image filters that preserve image edges can also be interpreted from a GSP perspctive. Ref. [55], [230] study the bilateral image filter and show that it is an order $K = 1$ low-pass GCF (compare to (66)) that smooths the image. To boost smoothing, [231] develops the trilateral filter, which has a rational graph frequency response, explaining its improved performance. GCFs are also used for guided image filtering in [232], [233].

Simple forms of GCFs are also used for smoothing and edge enhancement with low computational cost. A convolutional filter of small order (e.g., two) is used to smooth the image, and a successive filter of the form $\mathbf{H}(\mathbf{L}) = \mathbf{I} + h_1 \mathbf{L}$ is used to sharpen the edges [234], [235]. Ref. [236] uses GCFs to efficiently implement the sparse low-pass discrete cosine transform in the vertex domain. Lastly, median graph filters (40) are used in [237] to detect ships in image data.

E. Distributed Signal Processing

Because of their local implementation (Property 5), graph filters are readily distributable, where the graph captures both the signal structure and the distributed communication pattern. For example, we may want to denoise sensor measurements (the graph signal) over a network where each node can exchange information only locally. The research on *distributed graph filtering* has evolved in three main directions: (i) using graph filters to approximate a desired operation and apply it in a distributed fashion; (ii) analyzing the filtering performance when facing distributed communication challenges such as interference, asynchronous implementation, and quantization; and, (iii) estimating the filter parameters distributively.

Distributed tasks. Using graph filters for distributed processing implies first matching a desired operator [Sec. V-A] and then deploying the filter. Here, we first discuss distributed average consensus and then other general operators.

1) Consensus: Distributed average consensus is a cornerstone method underpinning myriad distributed estimation and detection tasks [238]. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing connectivities $\mathcal{E}$ between different agents $\mathcal{V}$, we want the agents to estimate the average value of their signals $\mathbf{x}$ by only exchanging information with their local neighbors. Let $\bar{x} := 1/N \sum_{n=1}^{N} x_n$ be the true average, $\mathbf{y} := \bar{x}\mathbf{1}$ the vector of averages, and $\mathbf{B} := \frac{1}{N}\mathbf{1}\mathbf{1}^{\top}$ the consensus operator. Then $\mathbf{y} = \mathbf{B}\mathbf{x}$. Graph convolutional filters can be used to reach *exact* and *finite-time* consensus as long as we design appropriately their coefficients, as stated by the following proposition.

Proposition 4 (Finite-time consensus [239])**.** *Average consensus can be computed exactly in finite-time by a graph convolutional filter of appropriate order if the Laplacian eigenvalue $\lambda_1 = 0$ is of multiplicity one, i.e., the graph is connected.*

Since the constant vector $\mathbf{v}_1 = 1/\sqrt{N}\mathbf{1}$ is an eigenvector of the Laplacian (i.e., $\mathbf{L}\mathbf{1} = 0$), we can find the filter coefficients to achieve the frequency response

$$\tilde{\mathsf{h}}(\lambda_n) = \begin{cases} 1 & \text{for } \lambda_n = 0 \ (n = 1) \\ 0 & \text{for } \lambda_n > 0 \ (n = 2, \ldots, N) \end{cases}. \tag{67}$$

References [239], [240] provide closed-form solutions for $\{h_k\}$, whereas [241] discusses theoretical limits on the minimum filter order for which consensus can be achieved.

Exact finite-time consensus can be challenging due to numerical issues related to computing close-by eigenvalues. Approximate consensus is analyzed in [239] for the convolutional filter (cf. (3)), in [8] for the node varying filter (32), and in [10] for the edge varying filter (34). The common observation is that filters with higher orders approximate better the consensus operator; however, instabilities during the design phase arise and more advanced design strategies are needed [93]. When the underlying graph comes from a random distribution, reaching consensus via graph filtering can be improved by accounting for the distribution of the eigenvalues [58].

Recent literature also discusses the links between consensus via graph filtering and control theory. Specifically, [242] discusses both finite-time and asymptotic consensus, and derives conditions when they can be achieved even over uncertain graphs. References [243], [244] focus on graph filters of order two to accelerate consensus. By linking the eigenvalues of the respective graph Laplacian with the graph properties, [243] provides optimal filter design for finite-time consensus and characterizes the convergence rate. On the other hand, [244] shows that for some graphs it is impossible to accelerate consensus. Finite-time consensus over directed graphs is discussed in [245], [246], where, as for the undirected case, the multiplicity of the eigenvalues influences the number of steps. Reference [246] discusses asymptotic consensus for unknown directed graphs, and [247] considers finite-time consensus over random graphs. Lastly, [248] focuses on group consensus via graph filtering, i.e., that nodes within a group achieve average consensus, but different groups can have different values.

2) General operator: In Sec. V-A, we discussed filter design strategies to match any general operator $\mathbf{B}$ via graph convolutional filters. Exploiting the filter locality, it is then possible to implement $\mathbf{B}$ (or an approximation) distributively over the graph. Distributed operator matching via graph filters is investigated for the convolutional filter in [9], [240], [249], rational filter in [57], [250], node varying filter in [8], edge varying in [10], [93], and for other modifications of these filters in [250]. The common theme is to approximate $\mathbf{B}$ with a low filter order, so as to limit the communication costs. Reference [251] details this challenge and designs the minimum order convolutional filter to either match or approximate the operator.

Distributed challenges. In distributed graph filtering, we must also account for the communication challenges, including:

1) Interference: In distributed processing, the communication edge weights $\hat{\mathbf{S}}$ may differ from the nominal ones $\mathbf{S}$ used to design the filter. Property 8 characterizes the impact of small differences in the GSO on the output of graph convolutional filters, which are Lipschitz. However, it focuses on small relative perturbations, while we often encounter larger perturbations such as link losses. The effect of link losses on graph filters is discussed in [252], [253]. The following result generalizes Property 8 to this stochastic setting.

Proposition 5 ([253])**.** *If the edges in the graph realization $\hat{\mathbf{S}} \subseteq \mathbf{S}$ are preserved independently with a probability p and the filter is Lipschitz (Property 8) with constant C, the expected squared deviation of the filter output is bounded as*

$$\mathbb{E}\big[\|\mathbf{H}(\hat{\mathbf{S}})\mathbf{x} - \mathbf{H}(\mathbf{S})\mathbf{x}\|_2^2\big] \leq \alpha N C^2 (1-p)\|\mathbf{x}\|_2^2 + \mathsf{O}((1-p)^2), \tag{68}$$

where α is either 2 or the maximal node degree, depending of the choice of shift operator.

Similar results are developed in [254, Proposition 1] for convolutional filters and in [252, Thm. 3] for distributed rational filters. Ref. [255] considers the setting where the link preservation probabilities are different for each edge, and characterizes the statistical output of both convolutional and node varying filters [255, Proposition 1]. The authors then consider such a statistical deviation to design robust filters and propose a cross-layer protocol to run graph filters over an asymmetric wireless sensor network. Robust data-driven learning of graph filters in stochastic settings is also investigated for GNNs: [254] shows that by learning the parameters on a perturbed graph, we can achieve robust transference; and [256] proposes a constraint-learning framework where the parameters are optimized in the expectation while bounding the output variance.

2) Asynchronous implementation: Another challenge in distributed filtering is that nodes cannot always communicate in a synchronous manner. Asynchronous communication enables scalability [257], as it avoids the need for global synchronization; however, in general, it compromises the guarantee that the filter output converges. The work in [258] provides sufficient conditions for an asynchronous implementation to converge to the designed output in a mean-squared error sense. Similar results are derived for filter banks in [259] and for the edge varying filter in [260].

3) Signal quantization: In a distributed setting, we may also need to account for the low communication capacity between sensors. In these cases, the exchanged signal shifts $\mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x}$ need to be quantized prior to transmission. The quantized signal can be written as $\tilde{\mathbf{x}}^{(k)} = \mathbf{x}^{(k)} + \mathbf{n}_{\text{q}}^{(k)}$, where $\mathbf{n}_{\text{q}}^{(k)}$ is the quantization error. In turn, the quantized filter output becomes

$$\mathbf{y}_{\text{q}} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x} + \sum_{k=1}^{K} h_k \sum_{\kappa=0}^{k-1} \mathbf{S}^{k-\kappa}\mathbf{n}_{\text{q}} := \mathbf{H}(\mathbf{S})\mathbf{x} + \boldsymbol{\varepsilon}_{\text{q}}, \quad (69)$$

where is $\boldsymbol{\varepsilon}_{\text{q}}$ the accumulated quantization error. This quantization error distorts the filter output and needs to be accounted for during the filter design phase. If $\tilde{\beta}(\lambda)$ is the desired frequency response and $\text{MSE}_Q(\mathbf{h})$ is the mean squared quantization error, the robust filter design problem looks like

$$\begin{aligned} \underset{\mathbf{h}}{\text{minimize}} \quad & \int_{\lambda} \left| \sum_{k=0}^{K} h_k \lambda^k - \tilde{\beta}(\lambda) \right|^2 d\lambda, \\ \text{subject to} \quad & \text{MSE}_Q(\mathbf{h}) \leq \gamma \end{aligned} \quad (70)$$

where γ controls the distortion and needs to be set in accordance with the quantization step size [261], [262]. Ref. [263] further discusses optimal quantization schemes and links them with the graph topology, while [262] discusses robust quantization in the presence of link losses. Ref. [205] further discusses the impact of quantization errors when learning localized dictionaries (cf. (63)) via graph filters, while [264] discusses a joint design of signal sampling and recovery under quantization.

Filter estimation. The above works consider filters that are designed centrally and implemented distributively. A recent stream of works consider the task of estimating the filter coefficients distributively when data is available. Formally, consider a series of input-output pairs $\{\mathbf{x}^{(t)}, \mathbf{y}^{(t)}\}$, where for each t, model (21) holds. We can reformulate the latter as

$$\mathbf{y}^{(t)} = \mathbf{Z}^{(t)}\mathbf{h} + \boldsymbol{\nu}^{(t)}, \quad (71)$$

where $\mathbf{Z}^{(t)} = [\mathbf{x}^{(t)}, \mathbf{S}\mathbf{x}^{(t)}, \ldots, \mathbf{S}^K\mathbf{x}^{(t)}]$. Under standard statistical assumptions, we can find the filter parameters $\mathbf{h}$ that minimize $\mathbb{E}\|\mathbf{y}^{(t)} - \mathbf{Z}^{(t)}\mathbf{h}\|_2^2$ by using classical centralized linear regression techniques [91]. More interestingly, we can decompose this objective among the N nodes as

$$\mathbf{h}^* = \arg\min_{\mathbf{h}} \sum_{i=1}^{N} \mathbb{E}|y_i^{(t)} - \mathbf{z}_i^{(t)\top}\mathbf{h}|^2, \quad (72)$$

where $\mathbf{z}_i^{(t)\top}$ is the i-th row of $\mathbf{Z}^{(t)}$. The reformulation in (72) leads to a decentralized solution. In particular, diffusion strategies are attractive since they are scalable, robust, and enable continuous learning and adaptation. A distributed adapt-then-combine diffusion least mean squares (LMS) algorithm takes the following form at every node i

$$\boldsymbol{\psi}_i^{(t+1)} = \mathbf{h}_i^{(t)} + \mu_i \mathbf{z}_i^{(t)} \left(y_i^{(t)} - \mathbf{z}_i^{(t)\top}\mathbf{h}_i^{(t)} \right), \quad (73a)$$

$$\mathbf{h}_i^{(t+1)} = c_{ii}\boldsymbol{\psi}_i^{(t+1)} + \sum_{j \in \mathcal{N}_i} c_{ji}\boldsymbol{\psi}_j^{(t+1)}, \quad (73b)$$

where $\mu_i > 0$ is a local step size and $\{c_{ji}\}$ are non-negative combination parameters satisfying $c_{ji} = 0$ if $j \notin \mathcal{N}_i$, and $\sum_{j=1}^{N} c_{ji} = 1$ [91]. In the adaptation step (73a), each node i updates its local parameter estimate $\mathbf{h}_i^{(t)}$ to an auxiliary intermediate vector $\boldsymbol{\psi}_i^{(t+1)}$. In the combination step (73b), node i aggregates its own intermediate vector $\boldsymbol{\psi}_i^{(t+1)}$ and those from its neighbors to update its estimate $\mathbf{h}_i^{(t+1)}$. We run T iterations of this diffusion algorithm to estimate the filter parameters at each node. Assuming that vectors $\mathbf{z}_i^{(t)}$ are drawn from a zero-mean random process that is white over the temporal dimension t, the following result holds.

Proposition 6 ([91])**.** *For any initial condition, the iterative algorithm in* (73) *converges asymptotically in the mean towards the optimal vector $\mathbf{h}^*$ (i.e., the expected value of the error goes to zero) if the step sizes μ_i are small enough.*

Several extensions of this basic formulation have been proposed. First, a state space formulation is discussed in [265], which allows also to find the minimum filter order. Second, model (21) assumes instantaneous diffusion, where node i processes $\mathbf{y}_i^{(t)}$ at each time instant t by collecting samples of $\mathbf{x}^{(t)}$ that are up to K hops away. Since this limits the practical implementation, [91] also considers a modified model where the successive shifts in the filter are applied to different time samples of the input. Third, the convergence rate of LMS is notoriously slow. To alleviate this problem, (i) [91] presents a modified adaptation step (73a) based on Newton's method where Hessian information is considered but at an increased (per iteration) computational cost; and (ii) [92] considers recursive least squares adaptive estimators. Lastly, [266] extends these techniques to nonlinear filters, and [267] discusses the distributed parameter estimation of GNNs.

X. Applications in Machine Learning

In machine learning, graph filters act as a parameterized mapping between input-output data pairs and use the graph structure as an inductive bias. Particular properties of interest include the limited number of parameters, permutation equivariance, and the linear computation cost. Hence, graph filters have been useful in the standard tasks of semi-supervised learning on graphs

[Sec. X-A] and unsupervised learning, especially in clustering-like algorithms [Sec. X-B]. Graph filters have also been used for graph-based matrix completion [Sec. X-C] and Gaussian processes [Sec. X-D]. Lastly, we review some applications in computer graphics and computer vision [Sec. X-E].

A. Semi-Supervised Learning

Semi-supervised learning on graphs classifies unlabelled nodes given labels on some other nodes. Graph filters can be used to weigh and propagate the label information of multi-hop neighbors to the unknown nodes. Mathematically, consider the label matrix $\mathbf{X} \in \mathbb{R}^{N \times C}$ such that row n represents the label of node n among the C classes, i.e., entry $[\mathbf{X}]_{nc} = 1$ if node n belongs to class $c \in [C]$ and zero otherwise. We consider that only $M \ll N$ nodes are labeled, treat each column of $\mathbf{X}$ as a graph signal, and infer the labels as

$$\mathbf{Y} = \mathbf{H}(\mathbf{S})\mathbf{X}, \tag{74}$$

where the unlabeled node m is assigned to class c for which entry $[\mathbf{Y}]_{mc}$ is highest. The filter parameters are estimated as

$$\underset{\mathcal{H}}{\text{minimize}} \ \big\|\mathbf{M}\big(\mathbf{H}(\mathbf{S})\mathbf{X} - \mathbf{X}\big)\big\|_{\text{F}} + \gamma \mathsf{r}(\mathcal{H}, \mathbf{Y}), \tag{75}$$

where $\mathbf{M} = \mathsf{diag}(\mathbf{m})$ and $\mathbf{m} \in \{0,1\}^N$ is a masking matrix to compute the error only on the labeled nodes. Instead, $\mathsf{r}(\mathcal{H}, \mathbf{Y})$ is a regularizer on the filter parameters $\mathcal{H}$ (e.g., norm two) or on the output $\mathbf{Y}$ (e.g., smooth label variation, cf. (5)). Refs. [21], [268] consider the convolutional filter (3) for binary and multi-class classification of blog networks and indirect bridge monitoring, respectively. Ref. [109] considers the Wiener graph filter (51) and shows improvement upon conventional label propagation algorithms. To further improve the expressivity of the mapping, [269] uses a bank of filters with multiple GSOs (cf. (52)), where each GSO represents a different similarity graph built from node features. Finally, [270] considers a bank of convolutional filters, where each filter is fitted to a particular class. Then, the unlabelled nodes are assigned to the class with the highest filter output. These works, however, solve the classification problem via regression-like cost functions (e.g., Frobeius norm $\|\cdot\|_{\text{F}}$ in (78)) which may lead to a suboptimal performance despite the efficient and convex implementation properties. GNNs are also a valid alternative, given their state-of-the-art performance in this task [187], [271]

B. Unsupervised Learning

The canonical task in unsupervised learning on graphs consists of grouping nodes in the absence of labels into different clusters such that nodes are tightly connected within clusters and loosely connected between them. In this context, graph filters have been used to tackle the scalability issues of different variants of spectral clustering, a conventional unsupervised learning technique. Graph filters have also been used as a signal model to detect clusters in networks when no topology information is available.

Spectral clustering. This is a family of algorithms that compute spectral embeddings of data points based on the eigenvectors of a graph Laplacian matrix; Alg. 1 shows the steps of a spectral clustering algorithm [39]. The spectral embeddings (step 4) are built from the k eigenvectors associated with the lower variation on the graph (cf. (6)); hence, behaving as an ideal low-pass graph filter. This step and the k-means clustering in step 7 are the computational bottlenecks of spectral clustering and limit its scalability. Thus, approximate solutions are often preferred to trade accuracy with scalability [272]. The scalability of spectral clustering is enhanced via graph filtering in [273]. The ideal graph filter is approximated via convolutional (cf. (19)) or rational [88] Chebyshev fitting; k-means is run only on a sampled number of nodes; and the cluster labels on the remaining nodes are obtained by solving a smooth regularized problem (cf. (42)). The filtering operations adopted in compressive spectral clustering are also implemented via the power method in [274] and via an asynchronous implementation in [275].

Algorithm 1 Spectral clustering blueprint.

1: **Input:** A set of N $d-$dimensional data points $\mathbf{f}_1, \ldots, \mathbf{f}_N$ and the number of clusters k;
2: Build an undirected similarity sparse graph $\mathcal{G}$ (cf. (1)) with each node a data point (e.g., a K nearest neighbor graph of N nodes);
3: Set $\mathbf{S} = \mathbf{L} \in \mathbb{R}^{N \times N}$ to be a graph Laplacian of $\mathcal{G}$;
4: Take the k eigenvectors $\mathbf{U}_k \in \mathbb{R}^{N \times k}$ of $\mathbf{L}$ associated with the k lowest eigenvalues (smoothest, cf. (6));
5: Normalize $\mathbf{U}_k$ row-wise (unit norm) to have $\tilde{\mathbf{U}}_k \in \mathbb{R}^{N \times k}$;
6: Treat each node n as a data point in $\mathbb{R}^k$ and define its feature vector $\tilde{\mathbf{f}}_n \in \mathbb{R}^k$ as the nth row of $\tilde{\mathbf{U}}_k$

$$\tilde{\mathbf{f}}_n = \tilde{\mathbf{U}}_k^{\top} \boldsymbol{\delta}_n, \tag{76}$$

where $\boldsymbol{\delta}_n \in \mathbb{R}^N$ is a Dirac vector with $[\boldsymbol{\delta}_n]_m = 1$ if $m = n$ and zero otherwise;
7: Obtain the k clusters via $k-$means with the Euclidean distance $d_{nm} = \|\tilde{\mathbf{f}}_n - \tilde{\mathbf{f}}_m\|_2$.

Blind community detection. As discussed in Section IX-C, graph filters can serve as generative models for nodal observations, inspiring a range of network inference methods. Inferring the entire graph structure is often only the first step of a longer pipeline where the ultimate goal is to obtain interpretable information from graph signals. To this end, a feature that is often sought in network science is the community structure that offers a coarse description of graphs. For this task, applying conventional methods necessitates a two-step procedure comprising graph learning and community detection. An alternative line of work, called blind community detection, recovers the communities directly from the observed signals bypassing the intermediate network inference step [276]. More precisely, under the assumption that the observed signals $\mathbf{x}$ are obtained by passing white noise through a low-pass filter, it follows that the leading eigenvectors of the covariance $\mathbf{\Sigma}_x$ coincide with the k lowest eigenvalues of $\mathbf{L}$ (see step 4 in Algorithm 1); see [276] for theoretical guarantees. Once this information is attained, the same steps as spectral clustering can be followed to reveal the community structure. The benefit of this direct approach stems from the fact that fewer observations are needed to recover the coarse community features compared to the detailed graph structure. Blind recovery of network features has been extended to community detection in dynamic graphs [277], node centrality estimation [278], [279], and topology change-point detection [280]–[282].

C. Matrix Completion and Collaborative Filtering

Matrix completion comprises filling the missing entries of a partially observed matrix. While its staple application is in recommender systems [283] it is also used in bioinformatics

[284], signal processing [285], and chemistry [286], to name a few. Graphs have been used to capture the structural side information among the rows and columns of this matrix and the entries are treated as signals over these graphs. Then, graph filters have been used to interpolate the missing values in a form akin to the signal reconstruction task seen in Sec. IX-A.

Specifically, consider matrix $\mathbf{R} \in \mathbb{R}^{R\times C}$ capturing interactions between RC entities, e.g., R users interacting with C items in a recommender system. We observe only a portion of $\mathbf{R}$, which we represent with the masked version $\mathbf{M} \odot \mathbf{R}$ where $\mathbf{M} \in \{0,1\}^{R\times C}$ is the masking matrix with $[\mathbf{M}]_{ij} = 1$ if $[\mathbf{R}]_{ij}$ is observed and zero otherwise. Then, the canonical matrix completion problem consists of solving

$$\underset{\mathbf{X}}{\text{minimize}} \ \|\mathbf{M} \odot (\mathbf{X} - \mathbf{R})\|_F^2 + \gamma \mathsf{r}(\mathbf{X}) \tag{77}$$

which looks for a matrix $\mathbf{X} \in \mathbb{R}^{R\times C}$ that is close to $\mathbf{R}$ on the observed entries while at the same time having a particular structure, e.g., low rank via the nuclear norm $\mathsf{r}(\mathbf{X}) := \|\mathbf{X}\|_*$. Such solutions suffer when an entire row/column of $\mathbf{R}$ is not observed or when the low-rank structure in $\mathbf{R}$ does not hold. In these cases, one can exploit side information, including, e.g., user features (age, gender, geolocation) for R or social interaction within them; or item features (category, co-purchase) for C. Such side information can be used to build two graphs $\mathbf{S}_R \in \mathbb{R}^{R\times R}$ and $\mathbf{S}_C \in \mathbb{R}^{C\times C}$ and treat each row $\mathbf{r}^r$ and column $\mathbf{r}_c$ of $\mathbf{R}$ as signals on these graphs, respectively. If the side information is unavailable, the graph can be built based on a similarity distance by using the available values in $\mathbf{R}$ [287].

Regularized filtering. Under the assumption that connected entities have similar preferences (e.g., similar users tend to like similar products, or co-purchased products tend to be liked similarly), regularized filtering (Sec. VI-D) is used to smooth the available values into the adjacent nodes by imposing $\mathsf{r}(\mathbf{X}) = \text{Tr}(\mathbf{X}^\top \mathbf{S}_R \mathbf{X}) + \text{Tr}(\mathbf{X}\mathbf{S}_C\mathbf{X}^\top)$ as a regularizer in (77), with $\mathbf{S}_R$ and $\mathbf{S}_C$ being some Laplacian form [288]–[291]. Such regularizers impose a low-pass filtering behavior on the two graphs.

Collaborative filtering. The above graph-based regularizer may be suboptimal because the low-pass filtering leads to interpolated values that are similar in strongly connected nodes. While this issue could be tackled by choosing a different graph regularizer, going down this path often leads to a trial and error process of choosing regularizer kernels. Another approach is to learn the parameters of a graph convolutional filter, in order to gather multi-hop neighbor information. The filter parameters are designed as

$$\underset{\mathcal{H}}{\text{minimize}} \ \sum_{(r,c)\in\mathcal{T}} \left| \left[\mathbf{H}(\mathbf{S}_R)\mathbf{x}^r\right]_c - [\mathbf{R}]_{rc} \right|^2 + \gamma \mathsf{r}(\mathcal{H}), \tag{78}$$

which fits to the available interactions while regularizing them (e.g., norm two). Reference [43] shows that a learned graph convolutional filter in this setting behaves as bandstop, in which the low-pass component smoothes the available values while the high-pass component improves diversity. Furthermore, the vanilla nearest neighbor collaborative filter is the particular case of an order one graph convolutional filter. Ref. [292] uses a filter bank of convolutional filters to balance recommendation accuracy with diversity, while [293] follows a graph convolutional approach over an item-item graph with the shift operator being a random walk Laplacian. Differently, [294], [295] treat matrix $\mathbf{R}$ as the interactions of a bipartite graph and build a convolutional filter on this graph. Reference [296] discusses further the details of these techniques with regularizer filtering for recommender systems. Extensions to GNNs could be found in [287].

D. Supervised Learning with Gaussian Processes

Consider the common scenario where we are given input-output data pairs $\{\mathbf{x}_n, \mathbf{y}_n\}$, with each input $\mathbf{x}_n \in \mathbb{R}^K$ and each output $\mathbf{y}_n \in \mathbb{R}^N$. We wish to learn a model of the form

$$\mathbf{y}_n = \mathsf{f}(\mathbf{x}_n) + \boldsymbol{\varepsilon}_n, \tag{79}$$

where $\{\boldsymbol{\varepsilon}_n\}$ are white Gaussian noise vectors and $\mathsf{f} : \mathbb{R}^K \to \mathbb{R}^N$ is an unknown, multi-output function. In Gaussian process (GP) regression [297], [298], $\mathsf{f}(\mathbf{x}_n)$ is modeled to be distributed as a GP

$$\mathsf{f}(\mathbf{x}_n) \sim \mathcal{GP}(\mathsf{m}(\mathbf{x}_n), \mathsf{K}(\mathbf{x}_n, \mathbf{x}_m)), \tag{80}$$

which is a distribution over functions characterized by a mean function $\mathsf{m}(\mathbf{x}) = \mathbb{E}[\mathsf{f}(\mathbf{x})]$ (i.e., the weighted average of the evaluations at $\mathbf{x}$ of all functions in the distribution) and a covariance kernel function

$$\mathsf{K}(\mathbf{x}_n, \mathbf{x}_m) = \mathbb{E}\left[\big(\mathsf{f}(\mathbf{x}_n) - \mathsf{m}(\mathbf{x}_n)\big)\big(\mathsf{f}(\mathbf{x}_m) - \mathsf{m}(\mathbf{x}_m)\big)\right]$$

that models the dependence between function values at two inputs $\mathbf{x}_n$ and $\mathbf{x}_m$.

When the outputs $\mathbf{y}_n \in \mathbb{R}^N$ are graph signals, an alternative model to (79) is

$$\mathbf{y}_n = \mathbf{H}(\mathbf{S})\mathsf{f}(\mathbf{x}_n) + \boldsymbol{\varepsilon}_n, \tag{81}$$

where again $\mathsf{f}(\mathbf{x}_n)$ is assumed to be a Gaussian process with covariance kernel $\mathsf{K}(\mathbf{x}_n, \mathbf{x}_m)$. Consequently, the covariance matrix between the respective outputs is $\text{Cov}(\mathbf{y}_n, \mathbf{y}_m) = \mathsf{K}(\mathbf{x}_n, \mathbf{x}_m)\mathbf{H}(\mathbf{S})\mathbf{H}(\mathbf{S})^\top$ [299], [300].

The advantage of incorporating the graph filters into the regression model (81) is that we can now impose particular signal behavior properties. We mention three examples. First, [299] considers $\mathbf{H}(\mathbf{S})$ to be a low-pass rational filter of order one (43), so that the model (81) outputs graph signals that are smooth with respect to the underlying graph. Second, [301] generalizes the Matérn kernel to the graph setting, yielding a graph filter of the form $\mathbf{H}(\mathbf{S}) = \left(\frac{2\alpha}{\beta^2}\mathbf{I}_N + \mathbf{S}\right)^{\frac{\alpha}{2}}$, where $\alpha, \beta > 0$ and $\mathbf{S}$ is some form of the Laplacian. Third, to further enhance the kernel flexibility, [300] considers a graph convolutional filter, where the parameters are estimated from the data to ensure a valid kernel. Because of the multi-hop locality of graph filters, such a parametric approach weighs accordingly the information of multi-hop neighbors and has shown a better performance compared with regularized-filtering kernels.

E. Computer Graphics and Computer Vision

In computer graphics and in computer vision – including subdomains such as virtual reality, geographic information systems, and autonomous driving – two types of sensing data have become increasingly prevalent [302]–[305]. First, using light detection and ranging (LiDAR) sensing, the external surfaces of objects are often represented with 3D point clouds and their physical coordinates (and possibly color information). Second, using depth cameras such as Microsoft Kinect, depth maps can be associated with the pixels of 2D images. For both types of data, graph filters have proved useful in common tasks such

as object classification [306], object tracking [307]–[309], motion estimation and forecasting [310], [311], facial recognition [312], visual localization [313], segmentation [314], [315], pose estimation [316], [317], pose transfer [318], compression [310], [319], registration [319], surface smoothing [11], [12], [319], [320], edge detection [321], inpainting [322], deblurring [323], and color denoising [324]. We highlight a few examples.

Surface smoothing. In one of the earliest examples of using the eigenvectors of a discrete Laplacian to perform graph filtering (1995), Taubin [11], [12] smooths polyhedral surfaces (also called *surface fairing*) by (i) creating a graph by connecting each pair of vertices that share a face in the polyhedral surface, and (ii) updating the vertex locations by applying a lowpass polynomial filter of the random walk Laplacian $\mathbf{L}_{\text{rw}}$ to each vector of coordinates; e.g., $\mathbf{x}_{\text{updated}} = \mathbf{H}(\mathbf{L}_{\text{rw}})\mathbf{x}$, $\mathbf{y}_{\text{updated}} = \mathbf{H}(\mathbf{L}_{\text{rw}})\mathbf{y}$, and $\mathbf{z}_{\text{updated}} = \mathbf{H}(\mathbf{L}_{\text{rw}})\mathbf{z}$.

Point cloud compression. To compress a single 3D point cloud in a manner that enhances application-dependent features such as edges, key points, or flatness, [319] suggests to resample the point cloud with a resampling distribution that is proportional to the norms of filtered attributes. That is, the probability of resampling vertex i is proportional to $||\boldsymbol{\delta}_i^\top \mathbf{H}(\mathbf{S})\mathbf{X}||_2$, where $\mathbf{X}$ is an $N \times K$ matrix of attributes, with the ith row corresponding to the selected attributes (e.g., 3D coordinates, RGB colors, textures) of the ith vertex. For example, when $\mathbf{X}$ is just the $N \times 3$ matrix of the coordinates, $\mathbf{S} = \mathbf{L}_{\text{rw}}$, and $\mathbf{H}(\mathbf{S})$ is a highpass graph filter, this resampling strategy leads to choosing relatively more points along the contours (e.g., corner points, edges, end points) of the 3D point cloud. The result can be beneficial for contour-based registration to align point clouds.

Given a sequence of 3D point clouds, [310] (i) uses graph wavelet coefficients as feature vectors to compute point-to-point correspondences between a sparse set of points from point clouds at each successive time; (ii) uses those sparse point-to-point correspondences to estimate motion over time; (iii) interpolates the motion to get a complete point-to-point correspondence mapping over the sequence of point clouds; and, (iv) leverages that motion map to compress and efficiently code the entire sequence of point clouds.

Object tracking. This problem consists of identifying an object in a sequence of images, and following its movement through time. This can typically be done by graph matching of the object through the sequence of images. An alternative approach [307], is to consider the object of interest as a grid graph and designing a graph filter tailored to identifying the object (see Sec. V). In particular, [307] learns a graph convolutional filter via least-squares (see Sec. VII). Subsequently, [308] considers popular solutions in the space of spatio-temporal Siamese networks, and suggests to replace these networks by graph convolutional networks (see Sec. VIII).

A more challenging problem is that of multi-object tracking, where many different objects have to be tracked simultaneously. The typical approach consists of first learning discriminative features for each object, and then tracking the temporal evolution of those features. In [309], a feature extraction mechanism based on GNNs is proposed. The main idea is to exploit the relationship between the objects to learn features that are more discriminative, and thus, easier to distinguish during tracking. This can be achieved by learning graph filters (either by themselves, or included within a GNN) to highlight high-frequency features – that is, the ones that are more different across the elements of the graph (see Sec. VII).

XI. Where to Start

Despite the extensive analysis of the different filtering forms, we purposely did not address in detail questions about *when* to use a particular filter type or *when* to choose filter banks or GNNs. While Secs. VI, VII, VIII and Table I discuss the advantages and limitations for each method, we believe there is no single recipe about what solution to use when, and this depends largely on the task at hand. That said, our general recommendation is to start simple, checking if the task can be accomplished with a single graph filter before moving to a filter bank, and seeing if the filter bank provides sufficient representations before proceeding to graph neural networks. Within the class of single filters, we recommend to start with the convolutional form (specifically a polynomial filter), and consider the more involved node or edge varying filters when a non-spectral operator is provided or a low (distributed) computation cost is a priority. Within the class of filter banks, the least complicated starting place is probably a single-level tight frame filter bank like the one shown in Fig. 5, as it avoids many extra choices about which vertices to downsample or how to reconnect the downsampled vertices in a coarser graph. We recommend moving to nonlinear filters, filter banks, or GNNs when interested in learning a nonlinear mapping from data. However, the expressive power of the latter (filter order, number of features, and layers) does not have to be too large, as widely suggested by the literature in computer science.

For hands-on practitioners, entry points from a GSP perspective are the toolboxes [325], [326], whereas from a machine learning perspective (especially GNNs), we suggest PyTorch Geometric [327] and the toolbox available at https://github.com/alelab-upenn/graph-neural-networks, which contains several of the filtering solutions discussed in this overview.

XII. A Look Ahead

We have identified the following main promising directions regarding fundamental research on graph filters.

1) The computation cost of graph filters is at best linear in the number of edges in the graph. While this may allow scalability to tens of thousands of nodes, it becomes a challenge for web-scale graphs containing billions of nodes and edges. In these cases, sparsifying techniques on the filter implementation are needed but at the same time the implications of these solutions into the output become more challenging to address.
2) Some recent works have shown that for particular classes of graphs we can exploit the graph frequency density distribution to improve the filter design. However, it is still unaddressed how to use properties of particular graph families to aid learning and to understand better how the statistical topological properties affect the filter frequency response.
3) We focused on the role of graph filters over static and idealistic graphs. However, real networks are dynamic, noisy, and the respective signals are also time varying. Therefore, one of the biggest challenges is to extend graph filters to this dynamic setting in a principled manner by accounting for the variability in the graph signals and in the number of nodes and edges [328]–[330].
4) In several nonlinear tasks (e.g., classification) graph filters are often designed via suboptimal losses to prioritize convex

TABLE I: Summary of the different graph filtering forms and their properties (P). "?" means that property has not been proven to hold or not.

Filter / Properties	P1	P2	P3	P4	P5	P6	P7	P8	Discussion & Recommendation
Convolutional (3)	✓	✓	✓	✓	✓	✓	✓	✓	Extends naturally from the conventional convolutional filters in DSP and respects also the convolution theorem in (11). May require high-orders K and suffers numerical instabilities for large powers $\mathbf{S}^k$. Recommendation is to use them as the baseline solution but often with a normalized GSO.
Rational (27)	✓	✓	✓	✓	✓	✓	✓	✓	Requires lower orders to approximate a given frequency response. Design is more challenging and requires solving a non-convex constrained problem (cf. (28)). Obtaining the output implies approximating an inverse problem via iterative methods (cf. (26)). Recommendation is to use them when the design could be centralized and the implementation distributed to reduce the communication cost of higher-oder convolutional filters.
Node var. (32)	✓	✗	✗	✗	✓	✓	✓	✓	Can approximate a broader family of operators than convolutional/rational while maintaining local implementation. It is not permutation equivariant thus cannot be transferred across graphs. Hence, recommendation is to consider them for approximating a desired operator over a fixed graph.
Edge var. (34)	✓	✗	✗	✗	✓	✓	?	?	Increases further the DoFs w.r.t. the node varying filter while maintaining the local implementation. The design problem to fit it into a defined operator is a least squares problem but with higher dimensions compared with the node varying and convolutional filter. As the node varying filter, it cannot be transferred across graphs and the high DoFs need to be reduced when used in a data-driven fashion (regularize design problem or share parameters). Recommendation is to consider for approximating complex tasks on a fixed graph or when a large amount of data is available.
Volterra (39)	✗	✗	?	✓	✓	✓	?	?	It is more flexible than the convolutional filter but shares parameters among nodes and enjoys a local implementation. Spectral design is more challenging. It is more appropriate for data fitting compared with the node and edge varying filters because of the low number of parameters and permutation equivariance. It can be a good alternative to the convolutional filtering when the spectral interpretation is not needed and to the node domain filters when parameters are estimated from data. Can still run into overfitting and numerical instabilities, thus, orthogonal polynomials are recommended.
Median (40)	✗	✗	✓	✓	✓	✓	?	?	Allows tackling outliers in graph signals propagation via a median operation of locally shifted inputs. Enjoys a local implementation and parameter sharing but the design is feasible only in a data driven fashion. Its application domain is more restricted than the convolutional filter but for denoising in anomalous signals it can be a viable tool.
Tikhonov (43)	✓	✓	✓	✓	✓	✓	✓	✓	Particular form of rational filtering of order one for undirected graphs. Typically used to smooth the observed signals.
Sobolev (45)	✓	✓	✓	✓	✓	✓	✓	✓	Particular form of rational filtering which can achieve arbitrary order for undirected graphs. It generalizes the Tikhonov filter to smooth observed signals.
Quadratic shift variation (47)	✓	✓	✓	✓	✗	✓	?	?	Inverse smooth filtering on directed graphs. Differently from the undirected counterpart it penalizes sharp shifted signal transitions and obtaining a local implementation with iterative solvers is challenging.
Trend filtering (48)	✗	✗	?	✓	?	?	?	?	Performs sparse filtering on undirected graphs by penalizing sharp transitions that happen only at a few nodes. It is more appropriate to use where the signal has similar values in group of nodes but arbitrary values in different groups. Proving what properties this type of filter satisfies is more challenging because it lacks a closed-form solution.
Total variation (49)	✗	✗	?	✓	?	?	?	?	Performs sparse filtering on directed graphs by penalizing sharp shifts at a few nodes. It complements the smoothness total variation counterpart (47). As for the graph trend filter, it lacks a closed-form solution and can be solved only with iterative methods.
Wiener filtring (51)	✓	✓	✓	✗	✗	✗	✗	?	Performs optimal statistical filtering for stationary graph signals. In the vanilla form, it is a rational filter that does not respect the graph sparsity, hence, many of the properties do not hold. But if we approximate its frequency response either with polynomial or rational filters, we could implement an approximation where all the properties hold.
Multi-GSO (52)	✓	✗	✓	✓	✓	✓	?	?	Performs convolutional filtering over multiple GSOs to represent input-output relations. It inherits several properties of the convolutional form but has a higher descriptive power. Yet, differently from node domain and nonlinear filters, it has less chances to overfit the data. The challenge remains to build multiple GSOs that can aid the problem at hand.

and mathematically tractable solutions. Further improvement can be achieved by using non-convex losses and iterative algorithms to find the filter parameters.

5) Federated learning tackles the problem of training a model when the data and/or the parameters are located on several different machines [331]. The central tenet of federated learning involves exchanging messages among these machines in order to train models, while satisfying security, privacy, and communication constraints. This exchange of messages can be interpreted as the implementation of one or more graph filters, and thus, graph filtering has the potential to be a useful framework for analyzing and synthesizing federated learning methods.
6) Regarding applications, graph filters and respective extensions have potential in power and water networks [34], [332], Internet of Things [32], and finance [333].

Finally, we remark that graphs represent only pairwise relationships between data points but complex networks and data may often be better represented by higher-order network structures [334] such as multi-relational graphs [335], cell or simplicial complexes [336]–[339], and hypergraphs [340]–[342]. Developing and analysing filters in these settings is an interesting avenue with large potential in both signal processing and machine learning.

REFERENCES

[1] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River, NJ: Pearson, 2010.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Comput. Mach. Learning. Cambridge, MA: The MIT Press, 2016.

[3] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, "Stationary graph processes and spectral estimation," *IEEE Trans. Signal Process.*, vol. 65, no. 22, pp. 5911–5926, 2017.

[4] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv:2104.13478v2*, 2022. [Online]. Available: http://arxiv.org/abs/2104.13478

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

[6] X. Dong, D. Thanou, L. Toni, M. Bronstein, and P. Frossard, "Graph signal processing for machine learning: A review and new perspectives," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 117–127, 2020.

[7] F. R. K. Chung, *Spectral Graph Theory*, ser. Regional Conf. Ser. Math. Providence, RI: Amer. Math. Soc., 1997, no. 92.

[8] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear networks operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, 2017.

[9] D. I Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via Chebyshev polynomial approximation," *IEEE Trans. Signal Inform. Process. Networks*, vol. 4, no. 4, pp. 736–751, 2018.

[10] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Trans. Signal Process.*, vol. 67, no. 9, pp. 2320–2333, 2019.

[11] G. Taubin, "A signal processing approach to fair surface design," in *ACM Conf. Comput. Graph. Interactive Techn.*, 1995, pp. 351–358.

[12] G. Taubin, T. Zhang, and G. Golub, "Optimal surface smoothing as filter design," in *Eur. Conf. Comput. Vision*, 1996, pp. 283–292.

[13] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Conf. Comput. Learning Theory*, 2003, pp. 144–158.

[14] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *Int. Conf. Mach. Learning*, 2004.

[15] S. Bougleux, A. Elmoataz, and M. Melkemi, "Discrete regularization on weighted graphs for image and mesh filtering," in *Int. Conf. Scale Space Variational Methods Comput. Vision*, vol. 4485, 2007, pp. 128–139.

[16] F. Zhang and E. R. Hancock, "Graph spectral image smoothing using the heat kernel," *Pattern Recognition*, vol. 41, no. 11, pp. 3328–3342, 2008.

[17] M. Crovella and E. Kolaczyk, "Graph wavelets for spatial traffic analysis," in *Joint Conf. IEEE Comput. Commun. Soc.*, 2003, pp. 1848–1857.

[18] R. R. Coifman and M. Maggioni, "Diffusion wavelets," *Appl. Comput. Harmonic Anal.*, vol. 21, no. 1, pp. 53–94, 2006.

[19] 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 Process. Mag.*, vol. 30, no. 3, pp. 83–98, 2013.

[20] M. Püschel and J. M. F. Moura, "Algebraic signal processing: Foundation and 1-D time," *IEEE Trans. Signal Process.*, vol. 56, no. 8, pp. 3572–3585, 2008.

[21] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, 2013.

[22] A. Sandyhaila and J. M. F. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3042–3054, 2014.

[23] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 128–138, 2020.

[24] Y. Ma, X. Liu, T. Zhao, Y. Liu, J. Tang, and N. Shah, "A unified view on graph neural networks as graph signal denoising," in *ACM Int. Conf. Inf. Knowl. Manag.*, 2021, pp. 1202–1211.

[25] M. Zhu, X. Wang, C. Shi, H. Ji, and P. Cui, "Interpreting and unifying graph neural networks with an optimization framework," in *ACM Web Conf.*, 2021, pp. 1215–1226.

[26] L. Ruiz, F. Gama, and A. Ribeiro, "Graph neural networks: Architectures, stability and transferability," *Proc. IEEE*, vol. 109, no. 5, pp. 660–682, 2021.

[27] E. Isufi, F. Gama, and A. Ribeiro, "EdgeNets: Edge varying graph neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 3862–3877, 2021.

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

[29] N. Tremblay, P. Gonçalves, and P. Borgnat, "Design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing: Principles and Applications*. Academic Press, 2018, ch. 11, pp. 299–324.

[30] D. I Shuman, "Localized spectral graph filter frames: A unifying framework, survey of design considerations, and numerical comparison," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 43–63, 2020.

[31] R. Ramakrishna, H.-T. Wai, and A. Scaglione, "A user guide to low-pass graph signal processing and its applications: Tools and applications," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 74–85, 2020.

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

[33] F. Gama, Q. Li, E. Tolstaya, A. Prorok, and A. Ribeiro, "Synthesizing decentralized controllers with graph neural networks and imitation learning," *IEEE Trans. Signal Process.*, vol. 70, pp. 1932–1946, 2022.

[34] R. Ramakrishna and A. Scaglione, "Grid-graph signal processing (grid-GSP): A graph signal processing framework for the power grid," *IEEE Trans. Signal Process.*, vol. 69, pp. 2725–2739, 2021.

[35] M. Eisen and A. Ribeiro, "Optimal wireless resource allocation with random edge graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 2977–2991, 2020.

[36] A. Chowdhury, G. Verma, C. Rao, A. Swami, and S. Segarra, "Unfolding WMMSE using graph neural networks for efficient power allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 6004–6017, 2021.

[37] A. Candelieri, D. Conti, and F. Archetti, "A graph based analysis of leak localization in urban water networks," *Procedia Eng.*, vol. 70, pp. 228–237, 2014.

[38] R. K. Jain, J. M. Moura, and C. E. Kontokosta, "Big data + big cities: Graph signals of urban air pollution," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 130–136, 2014.

[39] U. Von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.

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

[41] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 16–43, 2019.

[42] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 44–63, 2019.

[43] W. Huang, A. G. Marques, and A. R. Ribeiro, "Rating prediction via graph signal processing," *IEEE Trans. Signal Process.*, vol. 66, no. 19, pp. 5066–5081, 2018.

[44] O. Sporns, *Networks of the Brain*. MIT Press, 2016.

[45] M. O. Jackson, *Social and Economic Networks*. Princeton University Press, 2010.

[46] W. Huang, L. Goldsberry, N. F. Wymbs, S. T. Grafton, D. S. Bassett, and A. Ribeiro, "Graph frequency analysis of brain signals," *IEEE J. Sel. Topics Signal Process.*, vol. 10, no. 7, pp. 1189–1203, 2016.

[47] L. Xing and L. Sela, "Graph neural networks for state estimation in water distribution systems: Application of supervised and semisupervised learning," *J. Water Resour. Plan. Manag.*, vol. 148, no. 5, p. 04022018, 2022.

[48] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst*, vol. 32, no. 1, pp. 4–24, 2020.

[49] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Phys. A: Stat. Mech. Appl.*, vol. 390, no. 6, pp. 1150–1170, 2011.

[50] S. Mallat, "Group invariant scattering," *Commun. Pure, Appl. Math.*, vol. 65, no. 10, pp. 1331–1398, 2012.

[51] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and Systems*, 2nd ed., ser. Prentice Hall Signal Process. Prentice Hall, 1997.

[52] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Pearson, 2010.

[53] S. Sardellitti, S. Barbarossa, and P. Di Lorenzo, "On the graph Fourier transform for directed graphs," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 6, pp. 796–811, 2017.

[54] R. Shafipour, A. Khodabakhsh, G. Mateos, and E. Nikolova, "A directed graph Fourier transform with spread frequency components," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 946–960, 2018.

[55] B. Girault, A. Ortega, and S. S. Narayanan, "Irregularity-aware graph fourier transforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5746–5761, 2018.

[56] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 5680–5695, 2020.

[57] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 274–288, 2016.

[58] S. Kruzick and J. M. F. Moura, "Optimal filter design for signal processing on random graphs: Accelerated consensus," *IEEE Trans. Signal Process.*, vol. 66, no. 5, pp. 1258–1272, 2018.

[59] V. L. Druskin and L. A. Knizhnerman, "Two polynomial methods of calculating functions of symmetric matrices," *USSR Comput. Math. Math. Phys.*, vol. 29, no. 6, pp. 112–121, 1989.

[60] C.-C. Tseng and S.-L. Lee, "Minimax design of graph filter using Chebyshev polynomial approximation," *IEEE Trans. Circuits Syst. II: Express Br.*, vol. 68, no. 5, pp. 1630–1634, 2021.

[61] D. Pakiyarajah and C. U. Edussooriya, "Minimax design of computationally-efficient FIR graph filters using semidefinite programming," *IEEE Trans. Circuits Syst. II: Express Br.*, 2022.

[62] L. N. Trefethen, *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.

[63] M. Coutino, S. P. Chepuri, T. Maehara, and G. Leus, "Fast spectral approximation of structured graphs with applications to graph filtering," *Algorithms*, vol. 13, no. 9, p. 214, 2020.

[64] T. Fan, D. I Shuman, S. Ubaru, and Y. Saad, "Spectrum-adapted polynomial approximation for matrix functions with applications in graph signal processing," *Algorithms*, vol. 13, no. 11, p. 295, 2020.

[65] S. Kruzick and J. M. F. Moura, "Graph signal processing: Filter design and spectral statistics," in *IEEE Int. Workshop Comput. Advances Multi-Sensor Adaptive Process.*, 2017, pp. 1–5.

[66] J. Liu, E. Isufi, and G. Leus, "Filter design for autoregressive moving average graph filters," *IEEE Trans. Signal Inform. Process. Networks*, vol. 5, no. 1, pp. 47–60, 2018.

[67] A. Sakiyama, T. Namiki, and Y. Tanaka, "Design of polynomial approximated filters for signals on directed graphs," in *IEEE Global Conf. Signal and Inform. Process.*, 2017, pp. 633–637.

[68] S. Segarra, G. Mateos, A. G. Marques, and A. Ribeiro, "Blind identification of graph filters," *IEEE Trans. Signal Process.*, vol. 65, no. 5, pp. 1146–1159, 2017.

[69] D. Ramírez, A. G. Marques, and S. Segarra, "Graph-signal reconstruction and blind deconvolution for structured inputs," *Signal Process.*, vol. 188, p. 108180, 2021.

[70] A. Natali, M. Coutino, and G. Leus, "Topology-aware joint graph filter and edge weight identification for network processes," in *IEEE Int. Workshop Mach. Learning Signal Process.*, 2020, pp. 1–6.

[71] S. Rey, V. M. Tenorio, and A. G. Marques, "Robust graph filter identification and graph denoising from signal observations," *arXiv preprint arXiv:2210.08488*, 2022.

[72] S. Rey and A. G. Marques, "Robust graph-filter identification with graph denoising regularization," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2021, pp. 5300–5304.

[73] A. Ahmed, B. Recht, and J. Romberg, "Blind deconvolution using convex programming," *IEEE Trans. Inf. Theory*, vol. 60, no. 3, pp. 1711–1732, 2014.

[74] K. Iwata, K. Yamada, and Y. Tanaka, "Graph blind deconvolution with sparseness constraint," *arXiv preprint arXiv:2010.14002*, 2020.

[75] Y. Zhu, F. J. I. Garcia, A. G. Marques, and S. Segarra, "Estimating network processes via blind identification of multiple graph filters," *IEEE Trans. Signal Process.*, vol. 68, pp. 3049–3063, 2020.

[76] F. J. Iglesias, S. Segarra, and A. G. Marques, "Blind demixing of diffused graph signals," *arXiv preprint arXiv:2012.13301*, 2020.

[77] X. Shi, H. Feng, M. Zhai, T. Yang, and B. Hu, "Infinite impulse response graph filters in wireless sensor networks," *IEEE Signal Process. Lett.*, vol. 22, no. 8, pp. 1113–1117, 2015.

[78] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, 2018.

[79] J. Jiang and D. B. Tay, "Decentralised signal processing on graphs via matrix inverse approximation," *Signal Process.*, vol. 165, pp. 292–302, 2019.

[80] C. Cheng, N. Emirov, and Q. Sun, "Preconditioned gradient descent algorithm for inverse filtering on spatially distributed networks," *IEEE Signal Process. Lett.*, vol. 27, pp. 1834–1838, 2020.

[81] A. Loukas, A. Simonetto, and G. Leus, "Distributed autoregressive moving average graph filters," *IEEE Signal Process. Lett.*, vol. 22, no. 11, pp. 1931–1935, 2015.

[82] E. Isufi, A. Loukas, and G. Leus, "Autoregressive moving average graph filters: A stable distributed implementation," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2017, pp. 4119–4123.

[83] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 2009.

[84] T. Aittomäki and G. Leus, "Graph filter design using sum-of-squares representation," in *Eur. Signal Process. Conf.*, 2019, pp. 1–5.

[85] A. Jiang, B. Ni, J. Wan, and H. K. Kwan, "Stable ARMA graph filter design via partial second-order factorization," in *IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.

[86] D. Pakiyarajah and C. U. Edussooriya, "WLS design of ARMA graph filters using iterative second-order cone programming," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2022, pp. 5937–5941.

[87] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *ACM Conf. Comput. Graph. Interactive Techn.*, 1999, pp. 317–324.

[88] O. Rimleanscaia and E. Isufi, "Rational Chebyshev graph filters," in *Asilomar Conf. Signals, Systems and Comput.*, 2020, pp. 736–740.

[89] C.-C. Tseng, "Rational graph filter design using spectral transformation and IIR digital filter," in *IEEE Region 10 Conf.*, 2020, pp. 247–250.

[90] C. Cheng, J. Jiang, N. Emirov, and Q. Sun, "Iterative Chebyshev polynomial algorithm for signal denoising on graphs," in *Int. Conf. Samp. Theory and Appl.*, 2019, pp. 1–5.

[91] F. Hua, R. Nassif, C. Richard, H. Wang, and A. H. Sayed, "Online distributed learning over graphs with multitask graph-filter models," *IEEE Trans. Signal Inform. Process. Networks*, vol. 6, pp. 63–77, 2020.

[92] A. Alinaghi, S. Weiss, V. Stankovic, and I. Proudler, "Graph filter design for distributed network processing: a comparison between adaptive algorithms," in *2021 Sensor Signal Processing for Defence Conference (SSPD)*, 2021, pp. 1–5.

[93] M. A. Coutino Minguez and G. Leus, "A cascaded structure for generalized graph filters," *IEEE Trans. Signal Process.*, 2021.

[94] F. Gama, B. G. Anderson, and S. Sojoudi, "Node-variant graph filters in graph neural networks," in *IEEE Data Sci. Learning Workshop*, 2022, pp. 1–6.

[95] Z. Xiao, H. Fang, and X. Wang, "Distributed nonlinear polynomial graph filter and its output graph spectrum: Filter analysis and design," *IEEE Trans. Signal Process.*, vol. 69, pp. 1–15, 2021.

[96] S. Segarra, A. G. Marques, G. R. Arce, and A. Ribeiro, "Center-weighted median graph filters," in *IEEE Global Conf. Signal and Inform. Process.*, 2016, pp. 336–340.

[97] S. Segarra, A. G. Marques, G. R. Arce, and A. Ribeiro, "Design of weighted median graph filters," in *IEEE Int. Workshop Comp. Adv. Multi-Sensor Adaptive Process.*, 2017, pp. 1–5.

[98] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Invariance-preserving localized activation functions for graph neural networks," *IEEE Trans. Signal Process.*, vol. 68, pp. 127–141, 2019.

[99] B. Iancu, L. Ruiz, A. Ribeiro, and E. Isufi, "Graph-adaptive activation functions for graph neural networks," in *IEEE Int. Workshop Mach. Learn. Signal Process.*, 2020, pp. 1–6.

[100] J. H. Giraldo, A. Mahmood, B. Garcia-Garcia, D. Thanou, and T. Bouwmans, "Reconstruction of time-varying graph signals via Sobolev smoothness," *IEEE Trans. Signal Inform. Process. Networks*, vol. 8, pp. 201–214, 2022.

[101] P.-Y. Chen and S. Liu, "Bias-variance tradeoff of graph Laplacian regularizer," *IEEE Signal Process. Lett.*, vol. 24, no. 8, pp. 1118–1122, 2017.

[102] D. Romero, M. Ma, and G. B. Giannakis, "Kernel-based reconstruction of graph signals," *IEEE Trans. Signal Process.*, vol. 65, no. 3, pp. 764–778, 2016.

[103] M. Yang, M. Coutino, G. Leus, and E. Isufi, "Node-adaptive regularization for graph signal reconstruction," *IEEE Open J. Signal Process.*, vol. 2, pp. 85–98, 2021.

[104] Y.-X. Wang, J. Sharpnack, A. Smola, and R. Tibshirani, "Trend filtering on graphs," in *Int. Conf. Artificial Intell., Statist.*, 2015, pp. 1042–1050.

[105] R. Varma, H. Lee, J. Kovačević, and Y. Chi, "Vector-valued graph trend filtering with non-convex penalties," *IEEE Trans. Signal Inform. Process. Networks*, vol. 6, pp. 48–62, 2019.

[106] B. Girault, "Stationary graph signals using an isometric graph translation," in *Eur. Signal Process. Conf.*, 2015, pp. 1516–1520.

[107] N. Perraudin and P. Vandergheynst, "Stationary signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3462–3477, 2017.

[108] C. Zhang, D. Florêncio, and P. A. Chou, "Graph signal processing-a probabilistic framework," *Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2015-31*, 2015.

[109] B. Girault, P. Gonçalves, E. Fleury, and A. S. Mor, "Semi-supervised learning for graph to signal mapping: A graph signal Wiener filter interpretation," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2014, pp. 1115–1119.

[110] E. Isufi, P. Di Lorenzo, P. Banelli, and G. Leus, "Distributed Wiener-based reconstruction of graph signals," in *IEEE Stat. Signal Process. Workshop*, 2018, pp. 21–25.

[111] C. Zheng, C. Cheng, and Q. Sun, "Wiener filters on graphs and distributed polynomial approximation algorithms," *arXiv preprint arXiv:2205.04019*, 2022.

[112] H. Sevi, G. Rilling, and P. Borgnat, "Harmonic analysis on directed graphs and applications: From Fourier analysis to wavelets," *Appl. Comput. Harmon. Anal.*, vol. 62, pp. 390–440, 2023.

[113] F. Hua, C. Richard, J. Chen, H. Wang, P. Borgnat, and P. Gonçalves, "Learning combination of graph filters for graph signal modeling," *IEEE Signal Process. Lett.*, vol. 26, no. 12, pp. 1912–1916, 2019.

[114] J. Fan, C. Tepedelenlioglu, and A. Spanias, "Global optimization of graph filters with multiple shift matrices," in *Asilomar Conf. Signals, Systems and Comput.*, 2019, pp. 2082–2086.

[115] N. Emirov, C. Cheng, J. Jiang, and Q. Sun, "Polynomial graph filters of multiple shifts and distributed implementation of inverse filtering," *Sampl. Theory Signal Process. Data Anal.*, vol. 20, no. 1, pp. 1–39, 2022.

[116] Stanford University Computer Graphics Laboratory, "The Stanford 3D Scanning Repository," http://graphics.stanford.edu/data/3Dscanrep/.

[117] D. I Shuman, C. Wiesmeyr, N. Holighaus, and P. Vandergheynst, "Spectrum-adapted tight graph wavelet and vertex-frequency frames," *IEEE Trans. Signal Process.*, vol. 63, no. 16, pp. 4223–4235, 2015.

[118] N. Leonardi and D. Van De Ville, "Tight wavelet frames on multislice graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 13, pp. 3357–3367, 2013.

[119] B. Dong, "Sparse representation on graphs by tight wavelet frames and applications," *Appl. Comput. Harmon. Anal.*, vol. 42, no. 3, pp. 452–479, 2017.

[120] F. Göbel, G. Blanchard, and U. von Luxburg, "Construction of tight frames on graphs and application to denoising," in *Handbook of Big Data Analytics*. Springer, 2018, pp. 503–522.

[121] D. B. Tay, Y. Tanaka, and A. Sakiyama, "Almost tight spectral graph wavelets with polynomial filters," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 6, pp. 812–824, 2017.

[122] A. Sakiyama, K. Watanabe, and Y. Tanaka, "Spectral graph wavelets and filter banks with low approximation error," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 2, no. 3, pp. 230–245, 2016.

[123] T. Fan, D. I Shuman, S. Ubaru, and Y. Saad, "Spectrum-adapted polynomial approximation for matrix functions with applications in graph signal processing," *Algorithms*, vol. 13, no. 11, 295, pp. 1–22, 2020.

[124] J. Jiang, C. Cheng, and Q. Sun, "Nonsubsampled graph filter banks: Theory and distributed algorithms," *IEEE Trans. Signal Process.*, vol. 67, no. 15, pp. 3938–3953, 2019.

[125] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmonic Anal.*, vol. 30, no. 2, pp. 129–150, 2011.

[126] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. SIAM, 1996.

[127] B. Ricaud, D. I Shuman, and P. Vandergheynst, "On the sparsity of wavelet coefficients for signals on graphs," in *SPIE Wavelets and Sparsity*, 2013.

[128] N. Tremblay and P. Borgnat, "Graph wavelets for multiscale community mining," *IEEE Trans. Signal Process.*, vol. 62, pp. 5227–5239, 2014.

[129] X. Dong, A. Ortega, P. Frossard, and P. Vandergheynst, "Inference of mobility patterns via spectral graph wavelets," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2013, pp. 3118–3122.

[130] D. I Shuman, M. J. Faraji, and P. Vandergheynst, "Semi-supervised learning with spectral graph wavelets," in *Int. Conf. Samp. Theory and Appl.*, 2011.

[131] T. Kerola, N. Inoue, and K. Shinoda, "Spectral graph skeletons for 3D action recognition," in *Asian Conf. Comp. Vision*, 2014, pp. 417–432.

[132] H. Behjat, N. Leonardi, L. Sörnmo, and D. Van De Ville, "Anatomically-adapted graph wavelets for improved group-level fMRI activation mapping," *NeuroImage*, vol. 123, pp. 185–199, 2015.

[133] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *ACM Int. Conf. Knowl. Discov. Data Min.*, 2018, pp. 1320–1329.

[134] S. K. Narang and A. Ortega, "Local two-channel critically sampled filter-banks on graphs," in *IEEE Int. Conf. Image Process.*, 2010, pp. 333–336.

[135] A. Anis and A. Ortega, "Critical sampling for wavelet filterbanks on arbitrary graphs," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2017, pp. 3889–3893.

[136] D. B. Tay and J. Zhang, "Techniques for constructing biorthogonal bipartite graph filter banks," *IEEE Trans. Signal Process.*, vol. 63, no. 21, pp. 5772–5783, 2015.

[137] E. Pavez, B. Girault, A. Ortega, and P. A. Chou, "Two channel filter banks on arbitrary graphs with positive semi definite variation operators," *IEEE Trans. Signal Process.*, vol. 71, pp. 917–932, 2023.

[138] S. K. Narang and A. Ortega, "Perfect reconstruction two-channel wavelet filter-banks for graph structured data," *IEEE. Trans. Signal Process.*, vol. 60, no. 6, pp. 2786–2799, 2012.

[139] S. K. Narang and A. Ortega, "Compact support biorthogonal wavelet filterbanks for arbitrary undirected graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4673–4685, 2013.

[140] J. Zeng, G. Cheung, and A. Ortega, "Bipartite approximation for graph wavelet signal decomposition," *IEEE Trans. Signal Process.*, vol. 65, no. 20, pp. 5466–5480, 2017.

[141] D. B. Tay and A. Ortega, "Bipartite graph filter banks: Polyphase analysis and generalization," *IEEE Trans. Signal Process.*, vol. 65, no. 18, pp. 4833–4846, 2017.

[142] D. B. Tay, Y. Tanaka, and A. Sakiyama, "Critically sampled graph filter banks with polynomial filters from regular domain filter banks," *Signal Process.*, vol. 131, pp. 66–72, 2017.

[143] V. N. Ekambaram, G. C. Fanti, B. Ayazifar, and K. Ramchandran, "Circulant structures and graph signal processing," in *IEEE Int. Conf. Image Process.*, 2013.

[144] V. N. Ekambaram, G. Fanti, B. Ayazifar, and K. Ramchandran, "Critically-sampled perfect-reconstruction spline-wavelet filterbanks for graph signals," in *IEEE Glob. Conf. Signal and Inform. Process.*, 2013, pp. 475–478.

[145] M. S. Kotzagiannidis and P. L. Dragotti, "Splines and wavelets on circulant graphs," *Appl. Comput. Harmon. Anal.*, vol. 47, no. 2, pp. 481–515, 2019.

[146] O. Teke and P. P. Vaidyanathan, "Graph filter banks with M-channels, maximal decimation, and perfect reconstruction," in *IEEE Int. Conf. Acc., Speech, and Signal Process.*, 2016, pp. 4089–4093.

[147] O. Teke and P. P. Vaidyanathan, "Extending classical multirate signal processing theory to graphs – Part II: M-channel filter banks," *IEEE Trans. Signal Process.*, vol. 65, no. 2, pp. 423–437, 2017.

[148] D. B. Tay and A. Ortega, "M-channel graph filter banks: Polyphase analysis and structures," *IEEE Signal Process. Lett.*, vol. 26, no. 5, pp. 730–734, 2019.

[149] D. B. Tay and Z. Lin, "Design of near orthogonal graph filter banks," *IEEE Signal Process. Lett.*, vol. 22, no. 6, pp. 701–704, 2014.

[150] X. Zhang, "Design of orthogonal graph wavelet filter banks," in *Conf. IEEE Ind. Electron. Soc.*, 2016, pp. 889–894.

[151] D. B. Tay, Y. Tanaka, and A. Sakiyama, "Near orthogonal oversampled graph filter banks," *IEEE Signal Process. Lett.*, vol. 23, no. 2, pp. 277–281, 2016.

[152] M. Jansen, G. P. Nason, and B. W. Silverman, "Multiscale methods for data on graphs and irregular multidimensional situations," *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 71, no. 1, pp. 97–125, 2009.

[153] S. K. Narang and A. Ortega, "Lifting based wavelet transforms on graphs," in *APSIPA ASC*, 2009, pp. 441–444.

[154] D. B. Tay, A. Ortega, and A. Anis, "Cascade and lifting structures in the spectral domain for bipartite graph filter banks," in *APSIPA ASC*, 2018, pp. 1141–1147.

[155] J. Jiang, D. B. Tay, Q. Sun, and S. Ouyang, "Design of nonsubsampled graph filter banks via lifting schemes," *IEEE Signal Process. Lett.*, vol. 27, pp. 441–445, 2020.

[156] D. I Shuman, M. Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *IEEE Trans. Signal Process.*, vol. 64, no. 8, pp. 2119–2134, 2016.

[157] Y. Tanaka and A. Sakiyama, "M-channel oversampled graph filter banks," *IEEE Trans. Signal Process.*, vol. 62, no. 14, pp. 3578–3590, 2014.

[158] A. Sakiyama and Y. Tanaka, "Oversampled graph Laplacian matrix for graph filter banks," *IEEE Trans. Signal Process.*, vol. 62, no. 24, pp. 6425–6437, 2014.

[159] N. Tremblay and P. Borgnat, "Subgraph-based filterbanks for graph signals," *IEEE Trans. Signal Process.*, vol. 64, no. 15, pp. 3827–3840, 2016.

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

[161] S. Li, Y. Jin, and D. I Shuman, "Scalable M-channel critically sampled filter banks for graph signals," *IEEE Trans. Signal Process.*, vol. 67, no. 15, pp. 3954–3969, 2019.

[162] A. Sakiyama, K. Watanabe, Y. Tanaka, and A. Ortega, "Two-channel critically sampled graph filter banks with spectral domain sampling," *IEEE Trans. Signal Process.*, vol. 67, no. 6, pp. 1447–1460, 2019.

[163] A. Sakiyama, K. Watanabe, and Y. Tanaka, "M-channel critically sampled spectral graph filter banks with symmetric structure," *IEEE Signal Process. Lett.*, vol. 26, no. 5, pp. 665–669, 2019.

[164] A. Miraki, H. Saeedi-Sourck, N. Marchetti, and A. Farhang, "Spectral domain spline graph filter bank," *IEEE Signal Process. Lett.*, vol. 28, pp. 469–473, 2021.

[165] H. Q. Nguyen and M. N. Do, "Downsampling of signals on graphs via maximum spanning trees," *IEEE Trans. Signal Process.*, vol. 63, no. 1, pp. 182–191, 2014.

[166] X. Zheng, Y. Y. Tang, and J. Zhou, "A framework of adaptive multiscale wavelet decomposition for signals on undirected graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 7, pp. 1696–1711, 2019.

[167] A. Loukas and P. Vandergheynst, "Spectrally approximating large graphs with smaller graphs," in *Int. Conf. Mach. Learn.*, 2018, pp. 3237–3246.

[168] A. Loukas, "Graph reduction with spectral and cut guarantees," *J. Mach. Learn. Res.*, vol. 20, no. 116, pp. 1–42, 2019.

[169] Y. Jin, A. Loukas, and J. JaJa, "Graph coarsening with preserved spectral properties," in *Int. Conf. Artificial Intell., Statist.*, 2020, pp. 4452–4462.

[170] D. Thanou, D. I Shuman, and P. Frossard, "Learning parametric dictionaries for signals on graphs," *IEEE Trans. Signal Process.*, vol. 62, no. 15, pp. 3849–3862, 2014.

[171] H. Behjat, U. Richter, D. Van De Ville, and L. Sörnmo, "Signal-adapted tight frames on graphs," *IEEE Trans. Signal Process.*, vol. 64, no. 22, pp. 6017–6029, 2016.

[172] S. K. Narang, Y.-H. Chao, and A. Ortega, "Critically sampled graph-based wavelet transforms for image coding," in *APSIPA ASC*, 2013, pp. 1–4.

[173] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.

[174] V. N. Vapnik, "Principles of risk minimization for learning theory," in *Conf. Neural Inform. Process. Syst.*, 1991, pp. 831–838.

[175] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *ACM Int. Conf. Knowl. Discov. Data Min.*, 2018, pp. 974–983.

[176] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.

[177] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

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

[179] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire *et al.*, "ETA prediction with graph neural networks in Google maps," in *ACM Int. Conf. Inf. Knowl. Manag.*, 2021, pp. 3767–3776.

[180] M. Cheung, J. Shi, O. Wright, L. Y. Jiang, X. Liu, and J. M. F. Moura, "Graph signal processing and deep learning: Convolution, pooling, and topology," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 139–149, 2020.

[181] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Conf. Neural Inform. Process. Syst.*, 2016, pp. 3844–3858.

[182] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 4, pp. 1034–1049, 2018.

[183] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Int. Conf. Learning Representations*, 2014, pp. 1–14.

[184] M. He, Z. Wei, and J.-R. Wen, "Convolutional neural networks on graphs with Chebyshev approximation, revisited," *Adv. Neural Inform. Process. Syst.*, vol. 35, pp. 7264–7276, 2022.

[185] M. He, Z. Wei, H. Xu *et al.*, "Bernnet: Learning arbitrary graph spectral filters via bernstein approximation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 239–14 251, 2021.

[186] X. Wang and M. Zhang, "How powerful are spectral graph neural networks," in *International Conference on Machine Learning*. PMLR, 2022, pp. 23 341–23 362.

[187] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Int. Conf. Learning Representations*, 2017, pp. 1–14.

[188] H. Nt and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.

[189] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *AAAI Conf. Artif. Intell.*, vol. 34, no. 4, 2020, pp. 3438–3445.

[190] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Int. Conf. Mach. Learning*, vol. 97, 2019, pp. 6861–6871.

[191] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Int. Conf. Learning Representations*, 2019, pp. 1–17.

[192] W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Conf. Neural Inform. Process. Syst.*, 2017, pp. 1–11.

[193] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Int. Conf. Mach. Learning*, 2018, pp. 5453–5462.

[194] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Int. Conf. Learning Representations*, 2018, pp. 1–12.

[195] D. Pim, T. S. Cohen, and M. Welling, "Natural graph convolutions," 2021, US Patent App. 17/239,580.

[196] F. Gama, J. Bruna, and A. Ribeiro, "Stability of graph scattering transforms," in *Conf. Neural Inform. Process. Syst.*, 2019, pp. 8038–8048.

[197] D. Zou and G. Lerman, "Graph convolutional neural networks via scattering," *Appl. Comput. Harmonic Anal.*, vol. 49, no. 3, pp. 1046–1074, 2020.

[198] F. Gao, G. Wolf, and M. Hirn, "Geometric scattering for graph data analysis," in *Int. Conf. Mach. Learning*, vol. 97, 2019, pp. 2122–2131.

[199] V. N. Ioannidis, S. Chen, and G. B. Giannakis, "Efficient and stable graph scattering transforms via pruning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 3, pp. 1232–1246, 2020.

[200] T. Wang, H. Guo, X. Yan, and Z. Yang, "Speech signal processing on graphs: The graph frequency analysis and an improved graph Wiener filtering method," *Speech Commun.*, vol. 127, pp. 82–91, 2021.

[201] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai, "Graph regularized sparse coding for image representation," *IEEE Trans. Image Process.*, vol. 20, no. 5, pp. 1327–1336, 2010.

[202] K. N. Ramamurthy, J. J. Thiagarajan, P. Sattigeri, and A. Spanias, "Learning dictionaries with graph embedding constraints," in *Asilomar Conf. Signals, Systems and Comput.*, 2012, pp. 1974–1978.

[203] Y. Yankelevsky and M. Elad, "Dual graph regularized dictionary learning," *IEEE Trans. Signal Inform. Process. Networks*, vol. 2, no. 4, pp. 611–624, 2016.

[204] Y. Yankelevsky and M. Elad, "Finding gems: Multi-scale dictionaries for high-dimensional graph signals," *IEEE Trans. Signal Process.*, vol. 67, no. 7, pp. 1889–1901, 2019.

[205] D. Thanou and P. Frossard, "Learning of robust spectral graph dictionaries for distributed processing," *EURASIP J. Adv. Signal Process*, vol. 2018, no. 1, pp. 1–17, 2018.

[206] C. Hu, J. Sepulcre, K. A. Johnson, G. E. Fakhri, Y. M. Lu, and Q. Li, "Matched signal detection on graphs: Theory and application to brain imaging data classification," *NeuroImage*, vol. 125, pp. 587–600, 2016.

[207] Z. Xiao, H. Fang, and X. Wang, "Anomalous IoT sensor data detection: An efficient approach enabled by nonlinear frequency-domain graph analysis," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3812–3821, 2020.

[208] Z. Xiao, H. Fang, and X. Wang, "Nonlinear polynomial graph filter for anomalous IoT sensor detection and localization," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4839–4848, 2020.

[209] P. Ferrer-Cid, J. M. Barcelo-Ordinas, and J. Garcia-Vidal, "Volterra graph-based outlier detection for air pollution sensor networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2759–2771, 2022.

[210] R. Francisquini, A. C. Lorena, and M. C. Nascimento, "Community-based anomaly detection using spectral graph filtering," *Appl. Soft Comput.*, vol. 118, p. 108489, 2022.

[211] H. E. Egilmez and A. Ortega, "Spectral anomaly detection using graph-based filtering for wireless sensor networks," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2014, pp. 1085–1089.

[212] E. Isufi, A. S. Mahabir, and G. Leus, "Blind graph topology change detection," *IEEE Signal Process. Lett.*, vol. 25, no. 5, pp. 655–659, 2018.

[213] A. P. Dempster, "Covariance selection," *Biometrics*, vol. 28, no. 1, pp. 157–175, 1972.

[214] N. Meinshausen and P. Buhlmann, "High-dimensional graphs and variable selection with the Lasso," *Ann. Stat.*, vol. 34, pp. 1436–1462, 2006.

[215] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning Laplacian matrix in smooth graph signal representations," *IEEE Trans. Signal Process.*, vol. 64, no. 23, pp. 6160–6173, 2016.

[216] V. Kalofolias, "How to learn a graph from smooth signals," in *Int. Conf. Artificial Intell., Statist.* J Mach. Learn. Res., 2016, pp. 920–929.

[217] S. Segarra, A. Marques, G. Mateos, and A. Ribeiro, "Network topology inference from spectral templates," *IEEE Trans. Signal Inform. Process. Networks*, vol. 3, no. 3, pp. 467–483, 2017.

[218] R. Shafipour, S. Segarra, A. G. Marques, and G. Mateos, "Directed network topology inference via graph filter identification," in *IEEE Data Sci. Workshop*, 2018, pp. 210–214.

[219] D. Thanou, X. Dong, D. Kressner, and P. Frossard, "Learning heat diffusion graphs," *IEEE Trans. Signal Inform. Process. Networks*, vol. 3, no. 3, pp. 484–499, 2017.

[220] M. Coutino, E. Isufi, T. Maehara, and G. Leus, "State-space network topology identification from partial observations," *IEEE Trans. Signal Inform. Process. Networks*, vol. 6, pp. 211–225, 2020.

[221] Y. Zhu, M. T. Schaub, A. Jadbabaie, and S. Segarra, "Network inference from consensus dynamics with unknown parameters," *IEEE Trans. Signal Inform. Process. Networks*, vol. 6, pp. 300–315, 2020.

[222] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from filtered signals: Graph system and diffusion kernel identification," *IEEE Trans. Signal Inform. Process. Networks*, vol. 5, no. 2, pp. 360–374, 2019.

[223] O. Lézoray and L. Grady, *Image Processing and Analysis with Graphs*. CRC Press Boca Raton, 2012.

[224] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng, "Graph spectral image processing," *Proc. IEEE*, vol. 106, no. 5, pp. 907–930, 2018.

[225] X. Liu, D. Zhai, D. Zhao, G. Zhai, and W. Gao, "Progressive image denoising through hybrid graph Laplacian regularization: A unified framework," *IEEE Trans. Image Process.*, vol. 23, no. 4, pp. 1491–1503, 2014.

[226] J. Pang and G. Cheung, "Graph Laplacian regularization for image denoising: Analysis in the continuous domain," *IEEE Trans. Image Process.*, vol. 26, no. 4, pp. 1770–1785, 2017.

[227] A. Elmoataz, O. Lezoray, and S. Bougleux, "Nonlocal discrete regularization on weighted graphs: A framework for image and manifold processing," *IEEE Trans. Image Process.*, vol. 17, no. 7, pp. 1047–1060, 2008.

[228] A. C. Yağan and M. T. Özgen, "A spectral graph Wiener filter in graph Fourier domain for improved image denoising," in *IEEE Global Conf. Signal and Inform. Process.*, 2016, pp. 450–454.

[229] D. Tian, H. Mansour, A. Knyazev, and A. Vetro, "Chebyshev and conjugate gradient filters for graph image denoising," in *IEEE Int. Conf. Multimed. Expo Workshops*, 2014, pp. 1–6.

[230] A. Gadde, S. K. Narang, and A. Ortega, "Bilateral filter: Graph spectral interpretation and extensions," in *IEEE Int. Conf. Image Process.*, 2013, pp. 1222–1226.

[231] M. Onuki, S. Ono, M. Yamagishi, and Y. Tanaka, "Graph signal denoising via trilateral filter on graph spectral domain," *IEEE Trans. Signal Inform. Process. Networks*, vol. 2, no. 2, pp. 137–148, 2016.

[232] A. Knyazev and A. Malyshev, "Accelerated graph-based spectral polynomial filters," in *IEEE Int. Workshop Mach. Learn. Signal Process.*, 2015, pp. 1–6.

[233] Q. Huang, R. Li, Z. Jiang, W. Feng, S. Lin, H. Feng, and B. Hu, "Fast color-guided depth denoising for RGB-D images by graph filtering," in *Asilomar Conf. Signals, Systems and Comput.*, 2019, pp. 1811–1815.

[234] H. Sadreazami, A. Asif, and A. Mohammadi, "Data-driven image stylization using graph-based filtering," in *Can. Conf. Electr. Comput. Eng*, 2017, pp. 1–4.

[235] H. Sadreazami, A. Asif, and A. Mohammadi, "Data-adaptive color image denoising and enhancement using graph-based filtering," in *IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.

[236] K.-S. Lu, A. Ortega, D. Mukherjee, and Y. Chen, "DCT and DST filtering with sparse graph operators," *IEEE Trans. Signal Process.*, 2022.

[237] P. Salembier, S. Liesegang, and C. López-Martínez, "Ship detection in SAR images based on maxtree representation and graph signal processing," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 5, pp. 2709–2724, 2018.

[238] R. Olfati-Saber and J. S. Shamma, "Consensus filters for sensor networks and distributed sensor fusion," in *IEEE Conf. Decision Control*, 2005, pp. 6698–6703.

[239] A. Sandryhaila, S. Kar, and J. M. Moura, "Finite-time distributed consensus through graph filters," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2014, pp. 1080–1084.

[240] S. Safavi and U. A. Khan, "Revisiting finite-time distributed algorithms via successive nulling of eigenvalues," *IEEE Signal Process. Lett.*, vol. 22, no. 1, pp. 54–57, 2014.

[241] M. Coutino, E. Isufi, T. Maehara, and G. Leus, "On the limits of finite-time distributed consensus through successive local linear operations," in *Asilomar Conf. Signals, Systems and Comput.*, 2018, pp. 993–997.

[242] J.-W. Yi, L. Chai, and J. Zhang, "Average consensus by graph filtering: New approach, explicit convergence rate, and optimal design," *IEEE Trans. Automat. Control.*, vol. 65, no. 1, pp. 191–206, 2019.

[243] S. Apers and A. Sarlette, "Accelerating consensus by spectral clustering and polynomial filters," *IEEE Trans. Control Netw. Syst.*, vol. 4, no. 3, pp. 544–554, 2016.

[244] Q. Ran, J.-w. Yi, and L. Chai, "Fast consensus of multi-agent systems by second-order graph filters," in *Chinese Conf. Control*, 2021, pp. 5222–5227.

[245] T. Charalambous and C. N. Hadjicostis, "Laplacian-based matrix design for finite-time average consensus in digraphs," in *IEEE Conf. Decision Control*, 2018, pp. 3654–3659.

[246] K. Li, J.-W. Yi, and L. Chai, "Fast consensus of multi-agent systems on digraphs by graph filtering," in *Chinese Conf. Control Decision*, 2021, pp. 5279–5284.

[247] S. Kruzick and J. F. Moura, "Optimal filter design for consensus on random directed graphs," in *IEEE Stat. Signal Process. Workshop*, 2018, pp. 16–20.

[248] Q. Ran, J.-W. Yi, L. Chai, Y.-W. Wang, and X. Chen, "Group consensus of multi-agent systems by graph filtering," in *IEEE Int. Conf. Control Autom.*, 2020, pp. 1290–1295.

[249] S. Chen, A. Sandryhaila, J. M. Moura, and J. Kovacevic, "Signal denoising on graphs via graph filtering," in *IEEE Global Conf. Signal and Inform. Process.*, 2014, pp. 872–876.

[250] N. Emirov, C. Cheng, J. Jiang, and Q. Sun, "Polynomial graph filter of multiple shifts and distributed implementation of inverse filtering," *arXiv preprint arXiv:2003.11152*, 2020.

[251] D. Romero, S. Mollaebrahim, B. Beferull-Lozano, and C. Asensio-Marco, "Fast graph filters for decentralized subspace projection," *IEEE Trans. Signal Process.*, vol. 69, pp. 150–164, 2020.

[252] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Filtering random graph processes over random time-varying graphs," *IEEE Trans. Signal Process.*, vol. 65, no. 16, pp. 4406–4421, 2017.

[253] Z. Gao, E. Isufi, and A. Ribeiro, "Stability of graph convolutional neural networks to stochastic perturbations," *Signal Process.*, p. 108216, 2021.

[254] Z. Gao, E. Isufi, and A. Ribeiro, "Stochastic graph neural networks," *IEEE Trans. Signal Process.*, vol. 69, pp. 4428–4443, 2021.

[255] L. B. Saad and B. Beferull-Lozano, "Accurate graph filtering in wireless sensor networks," *IEEE Internet Things J.*, vol. 7, no. 12, pp. 11 431–11 445, 2020.

[256] Z. Gao and E. Isufi, "Learning stochastic graph neural networks with constrained variance," *IEEE Trans. Signal Process.*, vol. 71, pp. 358–371, 2023.

[257] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *J. ACM*, vol. 25, no. 2, pp. 226–244, 1978.

[258] O. Teke and P. P. Vaidyanathan, "IIR filtering on graphs with random node-asynchronous updates," *IEEE Trans. Signal Process.*, vol. 68, pp. 3945–3960, 2020.

[259] O. Teke and P. Vaidyanathan, "Node-asynchronous implementation of filter banks on graphs," in *Asilomar Conf. Signals, Systems and Comput.*, 2020, pp. 460–464.

[260] M. Coutino and G. Leus, "Asynchronous distributed edge-variant graph filters," in *IEEE Data Sci. Workshop*, 2019, pp. 115–119.

[261] L. F. Chamon and A. Ribeiro, "Finite-precision effects on graph filters," in *IEEE Global Conf. Signal and Inform. Process.*, 2017, pp. 603–607.

[262] L. B. Saad, B. Beferull-Lozano, and E. Isufi, "Quantization analysis and robust design for distributed graph filters," *IEEE Trans. Signal Process.*, 2021.

[263] I. C. M. Nobre and P. Frossard, "Optimized quantization in distributed graph signal processing," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2019, pp. 5376–5380.

[264] P. Li, N. Shlezinger, H. Zhang, B. Wang, and Y. C. Eldar, "Task-based graph signal compression," *arXiv preprint arXiv:2110.12387*, 2021.

[265] K. Ding, J. Wu, and L. Xie, "Minimum-degree distributed graph filter design," *IEEE Trans. Signal Process.*, vol. 69, pp. 1083–1096, 2021.

[266] V. R. Elias, V. C. Gogineni, W. A. Martins, and S. Werner, "Adaptive graph filters in reproducing kernel hilbert spaces: Design and performance analysis," *IEEE Trans. Signal Inform. Process. Networks*, vol. 7, pp. 62–74, 2020.

[267] S. Scardapane, I. Spinelli, and P. Di Lorenzo, "Distributed training of graph convolutional networks," *IEEE Trans. Signal Inform. Process. Networks*, vol. 7, pp. 87–100, 2020.

[268] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett, and J. Kovačević, "Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2879–2893, 2014.

[269] J. Fan, C. Tepedelenlioglu, and A. Spanias, "Graph-based classification with multiple shift matrices," *IEEE Trans. Signal Inform. Process. Networks*, 2022.

[270] D. Berberidis, A. N. Nikolakopoulos, and G. B. Giannakis, "Adaptive diffusions for scalable learning over graphs," *IEEE Trans. Signal Process.*, vol. 67, no. 5, pp. 1307–1321, 2018.

[271] Z. Song, X. Yang, Z. Xu, and I. King, "Graph-based semi-supervised learning: A comprehensive review," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[272] N. Tremblay and A. Loukas, "Approximating spectral clustering via sampling: A review," *Sampling Techniques for Supervised or Unsupervised Tasks*, pp. 129–183, 2020.

[273] N. Tremblay, G. Puy, R. Gribonval, and P. Vandergheynst, "Compressive spectral clustering," in *Int. Conf. Mach. Learning*, 2016, pp. 1002–1011.

[274] C. Boutsidis, P. Kambadur, and A. Gittens, "Spectral clustering via the power method-provably," in *Int. Conf. Mach. Learning*, 2015, pp. 40–48.

[275] O. Teke and P. P. Vaidyanathan, "Node-asynchronous spectral clustering on directed graphs," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2020, pp. 5325–5329.

[276] H.-T. Wai, S. Segarra, A. E. Ozdaglar, A. Scaglione, and A. Jadbabaie, "Blind community detection from low-rank excitations of a graph filter," *IEEE Trans. Signal Process.*, vol. 68, pp. 436–451, 2020.

[277] T. M. Roddenberry, M. T. Schaub, H.-T. Wai, and S. Segarra, "Exact blind community detection from signals on multiple graphs," *IEEE Trans. Signal Process.*, vol. 68, pp. 5016–5030, 2020.

[278] T. M. Roddenberry and S. Segarra, "Blind inference of eigenvector centrality rankings," *IEEE Trans. Signal Process.*, vol. 69, pp. 3935–3946, 2021.

[279] Y. He and H.-T. Wai, "Detecting central nodes from low-rank excited graph signals via structured factor analysis," *IEEE Trans. Signal Process.*, vol. 70, pp. 2416–2430, 2022.

[280] C. Kaushik, T. M. Roddenberry, and S. Segarra, "Network topology change-point detection from graph signals with prior spectral signatures," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2021, pp. 5395–5399.

[281] S. Shaked and T. Routtenberg, "Identification of edge disconnections in networks based on graph filter outputs," *IEEE Trans. Signal Inform. Process. Networks*, vol. 7, pp. 578–594, 2021.

[282] B. Marenco, P. Bermolen, M. Fiori, F. Larroca, and G. Mateos, "Online change point detection for weighted and directed random dot product graphs," *IEEE Trans. Signal Inform. Process. Networks*, vol. 8, pp. 144–159, 2022.

[283] Z. Chen and S. Wang, "A review on matrix completion for recommender systems," *Knowl. Inf. Syst.*, pp. 1–34, 2022.

[284] N. Natarajan and I. S. Dhillon, "Inductive matrix completion for predicting gene–disease associations," *Bioinformatics*, vol. 30, no. 12, pp. i60–i68, 2014.

[285] Z. Weng and X. Wang, "Low-rank matrix completion for array signal processing," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2012, pp. 2697–2700.

[286] S.-g. Lee and H.-g. Seol, "A survey on the matrix completion problem," *Trends in Mathematics*, vol. 4, no. 1, pp. 38–43, 2001.

[287] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He, and Y. Li, "A survey of graph neural networks for recommender systems: Challenges, methods, and directions," *ACM Trans. Rec. Syst.*, vol. 1, no. 1, pp. 1–51, 2023.

[288] Q. Gu, J. Zhou, and C. Ding, "Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs," in *SIAM Int. Conf. Data Min.*, 2010, pp. 199–210.

[289] L. Du, X. Li, and Y.-D. Shen, "User graph regularized pairwise matrix factorization for item recommendation," in *Int. Conf. Adv. Data Min. Appl.*, 2011, pp. 372–385.

[290] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst, "Matrix completion on graphs," *arXiv preprint arXiv:1408.1717*, 2014.

[291] N. Rao, H.-F. Yu, P. K. Ravikumar, and I. S. Dhillon, "Collaborative filtering with graph information: Consistency and scalable methods," *Adv. Neural Inform. Process. Syst.*, vol. 28, 2015.

[292] E. Isufi, M. Pocchiari, and A. Hanjalic, "Accuracy-diversity trade-off in recommender systems via graph convolutions," *Inf. Process. Manag.*, vol. 58, no. 2, p. 102459, 2021.

[293] A. N. Nikolakopoulos, D. Berberidis, G. Karypis, and G. B. Giannakis, "Personalized diffusions for top-n recommendation," in *ACM Conf. Rec. Syst.*, 2019, pp. 260–268.

[294] L. Chen, L. Wu, R. Hong, K. Zhang, and M. Wang, "Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach," in *AAAI Conf. Artif. Intell.*, vol. 34, no. 01, 2020, pp. 27–34.

[295] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "LightGCN: Simplifying and powering graph convolution network for recommendation," in *ACM/SIGIR Conf. Res. Dev. Inf. Retr.*, 2020, pp. 639–648.

[296] Y. Shen, Y. Wu, Y. Zhang, C. Shan, J. Zhang, B. K. Letaief, and D. Li, "How powerful is graph convolution for recommendation?" in *ACM Int. Conf. Inf. Knowl. Manag.*, 2021, pp. 1619–1629.

[297] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions," *J. Math. Psychol.*, vol. 85, pp. 1–16, 2018.

[298] J. Wang, "An intuitive tutorial to Gaussian processes regression," *arXiv preprint arXiv:2009.10862*, 2020.

[299] A. Venkitaraman, S. Chatterjee, and P. Handel, "Gaussian processes over graphs," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2020, pp. 5640–5644.

[300] Y.-C. Zhi, Y. C. Ng, and X. Dong, "Gaussian processes on graphs via spectral kernel learning," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 9, pp. 304–314, 2023.

[301] V. Borovitskiy, I. Azangulov, A. Terenin, P. Mostowsky, M. Deisenroth, and N. Durrande, "Matérn Gaussian processes on graphs," in *Int. Conf. Artificial Intell., Statist.*, 2021, pp. 2593–2601.

[302] F. Lozes, A. Elmoataz, and O. Lézoray, "PDE-based graph signal processing for 3-D color point clouds: Opportunities for cultural heritage," *IEEE Signal Process. Mag.*, vol. 32, no. 4, pp. 103–111, 2015.

[303] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez, and C. Wellington, "3D point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception," *IEEE Signal Process. Mag.*, vol. 38, no. 1, pp. 68–86, 2020.

[304] W. Hu, J. Pang, X. Liu, D. Tian, C.-W. Lin, and A. Vetro, "Graph signal processing for geometric data and beyond: Theory and applications," *IEEE Trans. Multimedia*, 2021.

[305] L. Jiao, J. Chen, F. Liu, S. Yang, C. You, X. Liu, L. Li, and B. Hou, "Graph representation learning meets computer vision: A survey," *IEEE Trans. Artificial Intell.*, 2022.

[306] R. Khasanova and P. Frossard, "Graph-based classification of omnidirectional images," in *IEEE Int. Conf. Comput. Vision*, 2017, pp. 869–878.

[307] Z. Cui, Y. Cai, W. Zheng, C. Xu, and J. Yang, "Spectral filter tracking," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2479–2489, 2018.

[308] J. Gao, T. Zhang, and C. Xu, "Graph convolutional tracking," in *Conf. Comput. Vision and Pattern Recognition*, 2019, pp. 4649–4659.

[309] X. Weng, Y. Wang, Y. Man, and K. M. Kitani, "GNN3DMOT: Graph neural network for 3D multi-object tracking with 2D-3D multi-feature learning," in *Conf. Comput. Vision and Pattern Recognition*, 2020, pp. 6499–6508.

[310] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3D point cloud sequences," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1765–1778, 2016.

[311] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *Eur. Conf. Comput. Vision*, 2020, pp. 541–556.

[312] Z. Wang, L. Zheng, Y. Li, and S. Wang, "Linkage based face clustering via graph convolution network," in *Conf. Comput. Vision and Pattern Recognition*, 2019, pp. 1117–1125.

[313] C. Lassance, Y. Latif, R. Garg, V. Gripon, and I. Reid, "Improved visual localization via graph filtering," *J. Imaging*, vol. 7, no. 2, p. 20, 2021.

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

[315] J. H. Giraldo, S. Javed, M. Sultana, S. K. Jung, and T. Bouwmans, "The emerging field of graph signal processing for moving object segmentation," in *Int. Workshop Frontiers Comput. Vision*, 2021, pp. 31–45.

[316] Y. Cai, L. Ge, J. Liu, J. Cai, T.-J. Cham, J. Yuan, and N. M. Thalmann, "Exploiting spatial-temporal relationships for 3D pose estimation via graph convolutional networks," in *IEEE Conf. Comput. Vision*, 2019, pp. 2272–2281.

[317] H. Choi, G. Moon, and K. M. Lee, "Pose2Mesh: Graph convolutional network for 3D human pose and mesh recovery from a 2D human pose," in *Eur. Conf. Comput. Vision*, 2020, pp. 769–787.

[318] J. Liu, Y. Zhao, S. Chen, and Y. Zhang, "A 3D mesh-based lifting-and-projection network for human pose transfer," *IEEE Trans. Multimedia*, 2021.

[319] S. Chen, D. Tian, C. Feng, A. Vetro, and J. Kovačević, "Fast resampling of three-dimensional point clouds via graphs," *IEEE Trans. Signal Process.*, vol. 66, no. 3, pp. 666–681, 2017.

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

[321] E. Bayram, "Spectral graph based approach for analysis of 3D LIDAR point clouds," Master's thesis, Middle East Technical University, 2017.

[322] P. Akyazi and P. Frossard, "Graph-based inpainting of disocclusion holes for zooming in 3D scenes," in *Eur. Signal Process. Conf.*, 2018, pp. 867–871.

[323] K. Yamamoto, M. Onuki, and Y. Tanaka, "Deblurring of point cloud attributes in graph spectral domain," in *Int. Conf. Image Process.*, 2016, pp. 1559–1563.

[324] C. Dinesh, G. Cheung, and I. V. Bajić, "3D point cloud color denoising using convex graph-signal smoothness priors," in *IEEE Int. Workshop Multimedia Signal Process.*, 2019, pp. 1–6.

[325] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, "GSPBOX: A toolbox for signal processing on graphs," *arXiv preprint arXiv:1408.5781*, 2014.

[326] B. Girault, S. S. Narayanan, A. Ortega, P. Gonçalves, and E. Fleury, "Grasp: A matlab toolbox for graph signal processing," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 6574–6575.

[327] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[328] E. Isufi, G. Leus, and P. Banelli, "2-dimensional finite impulse response graph-temporal filters," in *IEEE Global Conf. Signal and Inform. Process.*, 2016, pp. 405–409.

[329] F. Grassi, A. Loukas, N. Perraudin, and B. Ricaud, "A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs," *IEEE Trans. Signal Process.*, vol. 66, no. 3, pp. 817–829, 2017.

[330] B. Das and E. Isufi, "Graph filtering over expanding graphs," in *IEEE Data Sci. Learning Workshop*, 2022, pp. 1–8.

[331] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, 2020.

[332] X. Zhou, S. Liu, W. Xu, K. Xin, Y. Wu, and F. Meng, "Bridging hydraulics and graph signal processing: A new perspective to estimate water distribution network pressures," *Water Res.*, vol. 217, p. 118416, 2022.

[333] J. V. d. M. Cardoso, J. Ying, and D. P. Palomar, "Algorithms for learning graphs in financial markets," *arXiv preprint arXiv:2012.15410*, 2020.

[334] C. Bick, E. Gross, H. A. Harrington, and M. T. Schaub, "What are higher-order networks?" *arXiv preprint arXiv:2104.11329*, 2021.

[335] J. S. Stanley, E. C. Chi, and G. Mishne, "Multiway graph signal processing on tensors: Integrative analysis of irregular geometries," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 160–173, 2020.

[336] S. Barbarossa and S. Sardellitti, "Topological signal processing: Making sense of data building on multiway relations," *IEEE Signal Process. Mag.*, vol. 37, no. 6, pp. 174–183, 2020.

[337] M. T. Schaub, Y. Zhu, J.-B. Seby, T. M. Roddenberry, and S. Segarra, "Signal processing on higher-order networks: Livin'on the edge... and beyond," *Signal Process.*, vol. 187, p. 108149, 2021.

[338] M. Yang, E. Isufi, M. T. Schaub, and G. Leus, "Simplicial convolutional filters," *arXiv preprint arXiv:2201.11720*, 2022.

[339] T. M. Roddenberry, N. Glaze, and S. Segarra, "Principled simplicial neural networks for trajectory prediction," in *Int. Conf. Mach. Learning*, M. Meila and T. Zhang, Eds., vol. 139, 2021, pp. 9020–9029.

[340] S. Barbarossa and M. Tsitsvero, "An introduction to hypergraph signal processing," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2016, pp. 6425–6429.

[341] S. Zhang, Z. Ding, and S. Cui, "Introducing hypergraph signal processing: Theoretical foundation and practical applications," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 639–660, 2019.

[342] G. Leus, M. Yang, M. Coutino, and E. Isufi, "Topological Volterra filters," in *IEEE Int. Conf. Acoust., Speech and Signal Process.*, 2021, pp. 5385–5399.