Distributed Signal Processing via Chebyshev Polynomial Approximation

David I Shuman, Pierre Vandergheynst, Daniel Kressner, Pascal Frossard

***Abstract*—Unions of graph multiplier operators are an important class of linear operators for processing signals defined on graphs. We present a novel method to efficiently distribute the application of these operators. The proposed method features approximations of the graph multipliers by shifted Chebyshev polynomials, whose recurrence relations make them readily amenable to distributed computation. We demonstrate how the proposed method can be applied to distributed processing tasks such as smoothing, denoising, inverse filtering, and semi-supervised classification, and show that the communication requirements of the method scale gracefully with the size of the network.**

***Index Terms*—Chebyshev polynomial approximation, denoising, distributed lasso, distributed optimization, functions of matrices, learning, regularization, signal processing on graphs, spectral graph theory**

I. Introduction

In distributed signal processing tasks, the data to be processed is physically separated and cannot be transmitted to a central processing entity. This separation may be due to engineering limitations such as the limited communication range of wireless sensor network nodes, privacy concerns, or design considerations. Even when high-dimensional data can be processed centrally, it may be more efficient to process it with parallel computing. It is therefore important to develop distributed data processing algorithms that balance the trade-offs between performance, communication bandwidth, and computational complexity (speed).

A. The Communication Network and Signals on the Network

For concreteness, we focus throughout the paper on distributed processing examples in wireless sensor networks; however, the problems we consider could arise in a number of different settings. Due to the limited communication range of wireless sensor nodes, each sensor node in a large network is likely to communicate with only a small number of other nodes in the network. To model the communication patterns, we can write down a graph with each vertex corresponding to a sensor node and each edge corresponding to a pair of nodes that communicate. Moreover, because the communication graph is a function of the distances between nodes, it often captures spatial correlations between sensors' observations as well. That is, if two sensors are close enough to communicate, their observations are likely to be correlated. We can further specify these spatial correlations by adding weights to the edges of the graph, with higher weights associated to edges connecting sensors with closely correlated observations.

We model the communication network with an undirected, weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, w\}$, which consists of a set of vertices $\mathcal{V}$, a set of edges $\mathcal{E}$, and a weight function $w : \mathcal{E} \rightarrow \mathbb{R}^+$ that assigns a non-negative weight to each edge. We assume the number of nodes in the network, $N = |\mathcal{V}|$, is finite, and the graph is connected. The adjacency (or weight) matrix $\mathbf{W}$ for a weighted graph $\mathcal{G}$ is the $N \times N$ matrix with entries $W_{m,n}$, where

$$W_{m,n} = \begin{cases} w(e), & \text{if } e \in \mathcal{E} \text{ connects vertices } m \text{ and } n \\ 0, & \text{otherwise} \end{cases}.$$

Therefore, the weighted graph $\mathcal{G}$ can be equivalently represented as the triplet $\{\mathcal{V}, \mathcal{E}, \mathbf{W}\}$. The degree of each vertex is the sum of the weights of all the edges incident to it. We define the degree matrix $\mathbf{D}$ to be the diagonal matrix with the nth diagonal entry $D_{n,n}$ equal to the sum of the entries in the n^{th} row of $\mathbf{W}$.

A signal or function $f : \mathcal{V} \rightarrow \mathbb{R}$ defined on the vertices of the graph may be represented as a vector $\mathbf{f} \in \mathbb{R}^N$, where the n^{th} component of the vector $\mathbf{f}$ represents the function value at the n^{th} vertex in $\mathcal{V}$. Throughout, we use bold font to denote matrices and vectors, we denote the n^{th} component of a vector $\mathbf{f}$ by either $\mathbf{f}(n)$ or f_n, and we denote the n^{th} component of a matrix-vector product $\mathbf{Pf}$ by $(\mathbf{Pf})(n)$.

B. Distributed Signal Processing Tasks

We consider sensor networks whose nodes can only send messages to their local neighbors (i.e., they cannot communicate directly with a central entity). Much of the literature on distributed signal processing in such settings (see, e.g., [1]-[5] and references therein) focuses on coming to an agreement on simple features of the observed signal (e.g., consensus averaging, parameter estimation). We are more interested in processing the full function in a distributed manner, with each

David I Shuman is with the Department of Mathematics, Statistics, and Computer Science, Macalester College, St. Paul, MN 55105, USA (email: dshuman1@macalester.edu). Pierre Vandergheynst and Pascal Frossard are with the Signal Processing Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL), Institute of Electrical Engineering, CH-1015 Lausanne, Switzerland (email: {pierre.vandergheynst, pascal.frossard}@epfl.ch). Daniel Kressner is with the Institute of Mathematics, Ecole Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland (email: daniel.kressner@epfl.ch).

Part of the work reported here was presented at the *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), June 2011, Barcelona, Spain.*

This work was supported in part by FET-Open grant number 255931 UNLocX and FNS grant number 200021-118230.

The authors would like to thank Jalal Fadili for his help deriving the bound in Proposition 6, and the anonymous reviewers for their constructive feedback.

MATLAB code for all numerical experiments in this paper is available at http://www.macalester.edu/~dshuman1/publications.html

node having its own objective. Some example tasks under this umbrella include:

- *Distributed denoising* – In a sensor network of N sensors, a noisy N-dimensional signal is observed, with each component of the signal corresponding to the observation at one sensor location. Using the prior knowledge that the denoised signal should be smooth or piecewise smooth with respect to the underlying weighted graph structure, the sensors' task is to denoise each of their components of the signal by iteratively passing messages to their local neighbors and performing computations.
- *Distributed semi-supervised learning / transductive classification* – A class label is associated with each sensor node; however, only a small number of nodes in the network have knowledge of their labels. The cooperative task is for each node to learn its label by iteratively passing messages to its local neighbors and performing computations.

While the methods we propose are applicable to signals on any weighted, undirected graph $\mathcal{G}$, we predominantly have in mind large, sparse graphs (i.e., the number of edges grows approximately linearly with the number of vertices), which are both common in practice and constitute the realm in which the proposed methods are most beneficial.

C. Related Work

The tasks mentioned in Section I-B as well as other similar tasks have been considered recently in centralized settings in the fields of machine learning and signal processing on graphs [6], [7]. For example, [8]-[11] consider general regularization frameworks on weighted graphs; [12]-[19] present graph-based semi-supervised learning methods; and [20]-[23] consider regularization and filtering on weighted graphs for image and mesh processing. Spectral regularization methods for ill-posed inverse problems (see, e.g., [24] and references therein) are also closely related.

Also in a centralized setting, [25] shows that a truncated Chebyshev polynomial expansion efficiently approximates the application of a spectral graph wavelet transform. The truncated Chebyshev polynomial expansion technique is originally introduced in [26] in the context of approximately computing the product of a matrix function and a vector. In Section II, we discuss the connection between the graph multiplier operators we define and more general matrix functions.

In the distributed setting, reference [27] considers denoising via wavelet processing and [28] presents a denoising algorithm that projects the measured signal onto a low-dimensional subspace spanned by smooth functions. References [29]-[32] consider different distributed regression problems. Reference [33] extends the approach proposed in this paper by examining robustness to quantization noise. Segarra et al. [34], [35] approximate general linear transformations by what we define in Section II-A as graph multiplier operators. Infinite impulse response (IIR) graph spectral filters, which have recently been introduced in [36], [37], comprise another approach to many distributed graph signal processing tasks. These filters, which we discuss in more detail in Section V-D, can be written as the ratio of two polynomial functions.

D. Main Contributions

In the the initial presentation of this work [38], we extend the Chebyshev polynomial approximation method to the general class of unions of graph Fourier multiplier operators, and show how the recurrence properties of the Chebyshev polynomials also enable distributed application of these operators. The communication requirements for distributed computation using this method scale gracefully with the number of sensors in the network (and, accordingly, the size of the signals).

Our main contributions in this paper are to i) generalize graph Fourier multiplier operators to graph multiplier operators (to be defined in detail in Section II); ii) show that the application of linear operators that are unions of graph multiplier operators is a key component of distributed signal processing tasks such as distributed smoothing, denoising, inverse filtering, and semi-supervised learning; iii) present a novel method to efficiently distribute the application of the graph multiplier operators to high-dimensional signals; iv) provide theoretical bounds on the approximation error incurred by the proposed method; and v) theoretically and numerically compare the proposed method to alternative distributed computation methods.

The remainder of the paper is as follows. In the next section, we provide some background from spectral graph theory and matrix function theory, and introduce graph multiplier operators. In Section III, we provide examples of distributed signal processing tasks that feature the application of graph multiplier operators. In Section IV, we introduce a method to efficiently approximate these operators in a distributed setting via shifted Chebyshev polynomials. We discuss alternative methods to perform these approximate distributed computations in Section V, and we theoretically and numerically compare these alternative methods. In Section VI, we show how these methods can also be used to perform distributed wavelet denoising with the lasso regularization problem. Section VII concludes the paper.

II. Matrix Functions and Graph Multiplier Operators

In this section, we leverage notation from the theory of matrix functions to introduce a class of operators that we call *graph multiplier operators*. We also relate these operators to multiplier operators from classical Fourier analysis.

A. Matrix Functions

Functions of matrices [39] appear throughout mathematics, science, and engineering. While functions of more general matrices can be defined via the Jordan canonical form (e.g., [39, Definition 1.2, p. 3]), we restrict our attention in this paper to the simpler case of functions of real symmetric positive semi-definite matrices. Such a matrix $\mathbf{P} \in \mathbb{R}^{N \times N}$ has a complete set of orthonormal eigenvectors $\{\mathbf{u}_\ell\}_{\ell=0,1,\ldots,N-1}$ and associated real, non-negative eigenvalues $\{\lambda_\ell\}_{\ell=0,1,\ldots,N-1}$ satisfying $\mathbf{P}\mathbf{u}_\ell = \lambda_\ell \mathbf{u}_\ell$. That is, $\mathbf{P}$ admits a spectral decomposition $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^*$, where $\mathbf{U}$ is the $N \times N$ matrix with the $(\ell+1)^{\text{th}}$ column equal to the eigenvector $\mathbf{u}_\ell$, and $\mathbf{\Lambda}$ is the $N \times N$ diagonal matrix with the $(\ell+1)^{\text{th}}$ diagonal element equal to λ_ℓ. Without loss of generality, we assume the eigenvalues to be ordered as

$$0 \leq \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{N-1} := \lambda_{\max}.$$

Given a function $g(\cdot)$ well-defined on the spectrum $\sigma(\mathbf{P}) := \{\lambda_0, \lambda_1, \ldots, \lambda_{\max}\}$, the corresponding *matrix function* $g(\mathbf{P})$ is defined (e.g., [39, p. 3]) as

$$g(\mathbf{P}) := \mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^* := \mathbf{U} \begin{bmatrix} g(\lambda_0) & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & g(\lambda_{N-1}) \end{bmatrix} \mathbf{U}^*. \tag{1}$$

The class of operators that can be written as matrix functions of $\mathbf{P}$ can be equivalently characterized as follows.

Proposition 1: For a fixed real symmetric positive semi-definite matrix $\mathbf{P}$, the following are equivalent:

(a) $\mathbf{\Psi} = g(\mathbf{P})$ for some $g : \sigma(\mathbf{P}) \to \mathbb{R}$.
(b) $\mathbf{\Psi}$ and $\mathbf{P}$ are simultaneously diagonalizable by a unitary matrix; i.e., there exists a unitary matrix $\mathbf{U}$ such that $\mathbf{U}^*\mathbf{\Psi}\mathbf{U}$ and $\mathbf{U}^*\mathbf{P}\mathbf{U}$ are both diagonal matrices.
(c) $\mathbf{\Psi}$ and $\mathbf{P}$ commute; i.e., $\mathbf{\Psi}\mathbf{P} = \mathbf{P}\mathbf{\Psi}$.

Proof of Proposition 1: (a) implies (b) because of the definition (1) of $g(\mathbf{P})$, and (b) implies (a) if we set $g(\lambda_\ell)$ to the $(\ell+1)^{\text{th}}$ diagonal element of $\mathbf{U}^*\mathbf{\Psi}\mathbf{U}$. The equivalence between (b) and (c) is shown in [40, Corollary 4.5.18]. ■

B. Graph Multiplier Operators

In the context of distributed signal processing tasks, we are particularly interested in functions of symmetric matrices whose sparsity pattern is consistent with the communication structure of the network.

Definition 1: $\mathbf{\Psi}$ is a *graph multiplier operator* with respect to the graph $\mathcal{G}$ if there exists a real symmetric positive semi-definite matrix $\mathbf{P}$ and a function $g : \sigma(\mathbf{P}) \to \mathbb{R}$ such that

(i) $\mathbf{\Psi} = g(\mathbf{P}) = \sum_{\ell=0}^{N-1} g(\lambda_\ell)\mathbf{u}_\ell\mathbf{u}_\ell^*$, and
(ii) $P_{i,j} = 0$ if $W_{i,j} = 0$ and $i \neq j$; i.e., $\mathbf{P}$ has the same sparsity pattern as the graph Laplacian $\mathcal{L}$ of the graph $\mathcal{G}$.

In order for the distributed computational methods we introduce in Sections IV and V to be applicable to a wider range of applications, we can generalize slightly from graph multiplier operators to *unions of graph multiplier operators*. A union of graph multiplier operators is a linear operator $\Phi : \mathbb{R}^N \to \mathbb{R}^{\eta N}$ ($\eta \in \{1, 2, \ldots\}$) that can be written as

$$\mathbf{\Phi} = \begin{bmatrix} g_1(\mathbf{P}) \\ g_2(\mathbf{P}) \\ \vdots \\ g_\eta(\mathbf{P}) \end{bmatrix} = \left.\overbrace{\begin{bmatrix} \mathbf{\Psi}_1 \\ \mathbf{\Psi}_2 \\ \vdots \\ \mathbf{\Psi}_\eta \end{bmatrix}}^{N}\right\} \eta N \quad . \tag{2}$$

The application of the operator $\mathbf{\Phi}$ to a function $\mathbf{f}$ can equivalently be written as

$$(\mathbf{\Phi f})\left((j-1)N + n\right) = \sum_{\ell=0}^{N-1} g_j(\lambda_\ell)\langle \mathbf{f}, \mathbf{u}_\ell \rangle \mathbf{u}_\ell(n), \tag{3}$$
$$\text{for } j \in \{1, 2, \ldots, \eta\},\ n \in \{1, 2, \ldots, N\}.$$

C. Graph Fourier Multiplier Operators

When the matrix $\mathbf{P}$ in Definition 1 is the graph Laplacian $\mathcal{L}$, we call $\mathbf{\Psi}$ a *graph Fourier multiplier operator*. The non-normalized graph Laplacian is the real symmetric matrix $\mathcal{L} := \mathbf{D} - \mathbf{W}$, the difference between the degree matrix and the weighted adjacency matrix (see, e.g., [41], [42], for introductions to *spectral graph theory*). Because this situation arises frequently, we briefly motivate this terminology and relate it to the analogous operators from the classical signal processing literature.

For a function f defined on the real line, a *Fourier multiplier operator* or *filter* Ψ reshapes the function's frequencies through multiplication in the Fourier domain:

$$\widehat{(\Psi f)}(\omega) = g(\omega)\hat{f}(\omega), \text{ for every frequency } \omega.$$

Taking an inverse Fourier transform yields

$$\begin{aligned}(\Psi f)(x) &= \mathcal{F}^{-1}\Big(g(\omega)\mathcal{F}(f)(\omega)\Big)(x) \\ &= \frac{1}{2\pi}\int_{\mathbb{R}} g(\omega)\hat{f}(\omega)e^{i\omega x}\, d\omega.\end{aligned} \tag{4}$$

Denoting the eigenvectors of $\mathcal{L}$ by $\{\boldsymbol{\chi}_\ell\}_{\ell=0,1,\ldots,N-1}$, we can extend this straightforwardly to functions defined on the vertices of a graph by replacing the Fourier transform and its inverse in (4) with the graph Fourier transform $\hat{\mathbf{f}}(\ell) := \langle \mathbf{f}, \boldsymbol{\chi}_\ell \rangle = \sum_{n=1}^{N} \mathbf{f}(n)\boldsymbol{\chi}_\ell^*(n)$, and its inverse $\mathbf{f}(n) = \sum_{\ell=0}^{N-1} \hat{\mathbf{f}}(\ell)\boldsymbol{\chi}_\ell(n)$. Namely, a graph Fourier multiplier operator is a linear operator $\Psi : \mathbb{R}^N \to \mathbb{R}^N$ that can be written as

$$\begin{aligned}(\mathbf{\Psi f})(n) &= \mathcal{F}^{-1}\Big(g(\lambda_\ell)\mathcal{F}(f)(\ell)\Big)(n) \\ &= \sum_{\ell=0}^{N-1} g(\lambda_\ell)\hat{\mathbf{f}}(\ell)\boldsymbol{\chi}_\ell(n).\end{aligned} \tag{5}$$

We refer to $g(\cdot)$ as the *multiplier* or *graph spectral filter*.[1] Equivalently, borrowing the above notation from the theory of matrix functions [39], we can write

$$\mathbf{\Psi} = g(\mathcal{L}) = \sum_{\ell=0}^{N-1} g(\lambda_\ell)\boldsymbol{\chi}_\ell\boldsymbol{\chi}_\ell^* = \boldsymbol{\chi}g(\mathbf{\Lambda})\boldsymbol{\chi}^*.$$

A high-level intuition behind graph spectral filtering (5) is as follows. The eigenvectors corresponding to the lowest eigenvalues of the graph Laplacian are the "smoothest" in the sense that $|\boldsymbol{\chi}_\ell(m) - \boldsymbol{\chi}_\ell(n)|$ is small for neighboring vertices m and n. The inverse graph Fourier transform provides a representation of a signal $\mathbf{f}$ as a superposition of the orthonormal set of eigenvectors of the graph Laplacian. The effect of the graph Fourier multiplier operator Ψ is to modify the contribution of each eigenvector. For example, applying a multiplier $g(\cdot)$ that is 1 for all λ_ℓ below some threshold, and 0 for all λ_ℓ above the threshold is equivalent to projecting the signal onto the eigenvectors of the graph Laplacian associated with the lowest eigenvalues. This is analogous to ideal lowpass filtering in the continuous domain. Section III contains further intuition about

[1]Unlike [6], we omit the hat symbol ($\hat{\cdot}$) on the multiplier $g(\cdot)$, in order to maintain consistency with the notation most commonly used for matrix functions.

and examples of graph Fourier multiplier operators. For more properties of the graph Laplacian eigenvectors, see [6] and [43], and references therein.

III. Illustrative Distributed Signal Processing Applications

In this section, we show that a number of distributed signal processing tasks can be solved as applications of graph multiplier operators or unions of graph multiplier operators.

A. Denoising with Distributed Tikhonov Regularization

First, we consider the distributed denoising task discussed in Section I. We start with a noisy signal $\mathbf{y} \in \mathbb{R}^N$ that is defined on a graph of N sensors and has been corrupted by uncorrelated additive Gaussian noise. Through an iterative process of local communication and computation, each sensor should end up with a denoised estimate of its component, f_n^0, of the true underlying signal, $\mathbf{f}^0$.

To solve this problem, we enforce *a priori* information that the target signal is smooth with respect to the underlying graph topology. To enforce the global smoothness prior, we consider the class of regularization terms $\mathbf{f}^{\mathrm{T}}\mathcal{L}^r\mathbf{f}$ for $r \geq 1$. The resulting distributed regularization problem has the form

$$\operatorname*{argmin}_{\mathbf{f}} \frac{\tau}{2}\|\mathbf{f}-\mathbf{y}\|_2^2 + \mathbf{f}^{\mathrm{T}}\mathcal{L}^r\mathbf{f}. \tag{6}$$

Intuitively, the regularization term $\mathbf{f}^{\mathrm{T}}\mathcal{L}^r\mathbf{f}$ is small when the signal $\mathbf{f}$ has similar values at neighboring vertices with large weights (i.e., it is smooth). For example, when $r = 1$,

$$\mathbf{f}^{\mathrm{T}}\mathcal{L}\mathbf{f} = \frac{1}{2}\sum_{n\in\mathcal{V}}\sum_{m\sim n} W_{m,n}\big(f_m - f_n\big)^2.$$

The proof of the following proposition is included in the Appendix of [44].

Proposition 2: The solution to (6) is given by $\mathbf{R}\mathbf{y}$, where $\mathbf{R}$ is a graph Fourier multiplier operator of the form (5), with multiplier $g(\lambda_\ell) = \frac{\tau}{\tau+2\lambda_\ell^r}$.[2]

So, one way to do distributed denoising is to approximately compute $\mathbf{R}\mathbf{y}$ in a distributed manner. We discuss methods to do this in Sections IV and V, and numerical examples are included in Section IV-D and Section V-E.

B. Distributed Smoothing

An application closely related to distributed denoising is distributed smoothing. Here, the graph Fourier multiplier is the *heat kernel* $g(\lambda_\ell) = e^{-t\lambda_\ell}$. In other words, a signal $\mathbf{y} \in \mathbb{R}^N$ is smoothed by computing $\mathbf{H}_t\mathbf{y}$, where $(\mathbf{H}_t\mathbf{y})(n) := \sum_{\ell=0}^{N-1} e^{-t\lambda_\ell}\hat{\mathbf{y}}(\ell)\boldsymbol{\chi}_\ell(n)$ for fixed t. In the context of a centralized image smoothing application, [22] discusses in detail the heat kernel and its relationship to classical Gaussian filtering. Similar to both the example at the end of Section II-C and distributed Tikhonov regularization, the main idea is that the multiplier $g(\lambda_\ell) = e^{-t\lambda_\ell}$ acts as a lowpass filter that attenuates the higher frequency (less smooth) components of $\mathbf{y}$. The distributed smoothing problem is to compute $\mathbf{R}\mathbf{y}$, with $\mathbf{R} = \mathbf{H}_t = e^{-t\mathcal{L}}$ and each vertex n beginning with only its observation y_n.

[2]This filter $g(\lambda_\ell)$ is the graph analog of a first-order Bessel filter from classical signal processing of functions on the real line.

C. Distributed Inverse Filtering

Next, we consider the situation where node n observes the n^{th} component of $\mathbf{y} = \boldsymbol{\Psi}\mathbf{f} + \boldsymbol{\nu}$, where $\boldsymbol{\Psi}$ is a graph Fourier multiplier operator with multiplier $g_\Psi(\cdot)$, and $\boldsymbol{\nu}$ is uncorrelated Gaussian noise. The task of the network is to recover $\mathbf{f}$ by inverting the effect of the graph multiplier operator $\boldsymbol{\Psi}$. This is the distributed graph analog to the deblurring problem in imaging, which is discussed in [45, Chapter 7]. As discussed in [45, Chapter 7], trying to recover $\mathbf{f}$ by simply applying the inverse filter in the graph Fourier domain, i.e., setting

$$\begin{aligned}\mathbf{f}_*(n) &= \sum_{\ell=0}^{N-1}\left(\frac{1}{g_\Psi(\lambda_\ell)}\right)\hat{\mathbf{y}}(\ell)\boldsymbol{\chi}_\ell(n)\\ &= \sum_{\ell=0}^{N-1}\left(\hat{\mathbf{f}}(\ell) + \frac{\hat{\boldsymbol{\nu}}(\ell)}{g_\Psi(\lambda_\ell)}\right)\boldsymbol{\chi}_\ell(n)\\ &= \mathbf{f}(n) + \sum_{\ell=0}^{N-1}\left(\frac{\hat{\boldsymbol{\nu}}(\ell)}{g_\Psi(\lambda_\ell)}\right)\boldsymbol{\chi}_\ell(n),\end{aligned} \tag{7}$$

does not work well when $g_\Psi(\cdot)$ is zero (or close to zero) for high frequencies, because the summation in (7) blows up, dominating $\mathbf{f}(n)$. Therefore, we again use the prior that the signal is smooth with respect to the underlying graph structure, and approximately solve the regularization problem

$$\operatorname*{argmin}_{\mathbf{f}} \frac{\tau}{2}\|\mathbf{y}-\boldsymbol{\Psi}\mathbf{f}\|_2^2 + \mathbf{f}^{\mathrm{T}}\mathcal{L}^r\mathbf{f} \tag{8}$$

in a distributed manner.

Proposition 3: The solution to (8) is given by $\mathbf{R}\mathbf{y}$, where $\mathbf{R}$ is a graph Fourier multiplier operator with multiplier

$$h(\lambda_\ell) = \frac{\tau g_\Psi(\lambda_\ell)}{\tau g_\Psi^2(\lambda_\ell) + 2\lambda_\ell^r}.$$

The proof of Proposition 3 is included in the Appendix of [44].

D. Distributed Semi-Supervised Classification

The goal of *semi-supervised classification* is to learn a mapping from the data points $X = \{x_1, x_2, \ldots, x_N\}$ to their corresponding labels $Y = \{y_1, y_2, \ldots, y_N\}$. The pairs (x_i, y_i) are independently and identically sampled from a joint distribution $p(x, y)$ over the sample space $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{Y} := \{1, 2, \ldots, \kappa\}$ is the space of κ classes. The transductive classification problem is to use the full set of data points $X = \{x_1, x_2, \ldots, x_N\}$ and the labels $Y_l = \{y_1, y_2, \ldots, y_l\}$ associated with a small portion of the data ($l \ll N$) to predict the labels $Y_u = \{y_{l+1}, y_{l+2}, \ldots, y_N\}$ associated with the unlabeled data $X_u = \{x_{l+1}, x_{l+2}, \ldots, x_N\}$.

Many semi-supervised learning methods represent the data X by an undirected, weighted graph, and then force the labels to be smooth with respect to the intrinsic structure of this graph. We show how a number of these centralized graph-based semi-supervised classification methods can be written as applications of graph multiplier operators. Throughout, we assume there is one data point at each node in the graph, and the nodes know the weights of the edges connecting them to their neighbors in the graph. For example, each data point could be at a different node in a sensor network, and the weights could be a function of the physical distance between the nodes.

For different choices of reproducing kernel Hilbert spaces (RKHS) $\mathcal{H}$, a number of centralized semi-supervised classification methods estimate the label of the n^{th} data point ($n \in \{l+1, \ldots, N\}$) by

$$\underset{j \in \{1,2,\ldots,\kappa\}}{\arg\max} \; F_{nj}^{\text{opt}}, \text{ where} \tag{9}$$

$$\mathbf{F}^{\text{opt}} = \underset{\mathbf{F} \in \mathbb{R}^{N \times \kappa}}{\operatorname{argmin}} \sum_{j=1}^{\kappa} \left\{ \tau \|\mathbf{F}_{:,j} - \mathbf{Y}_{:,j}\|_2^2 + \|\mathbf{F}_{:,j}\|_{\mathcal{H}}^2 \right\}. \tag{10}$$

In (10), $\mathbf{A}_{:,j}$ denotes the j^{th} column of a matrix $\mathbf{A}$; $\mathbf{Y}$ is an $N \times \kappa$ matrix with entries

$$Y_{ij} = \begin{cases} 1, & \text{if } i \in \{1, 2, \ldots, l\} \text{ and the label for point } i \text{ is } j \\ 0, & \text{otherwise} \end{cases};$$

and for some symmetric positive semi-definite matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$,

$$\|\mathbf{f}\|_{\mathcal{H}}^2 = \langle \mathbf{f}, \mathbf{f} \rangle_{\mathcal{H}} := \langle \mathbf{f}, \mathbf{S}\mathbf{f} \rangle = \mathbf{f}^{\text{T}}\mathbf{S}\mathbf{f}. \tag{11}$$

Note that for any symmetric positive semi-definite matrix $\mathbf{S}$, $\mathcal{H}$ endowed with the inner product defined in (11) is in fact a RKHS on the image of $\mathbb{R}^N$ under $\mathbf{S}$, and its kernel is $k(i,j) = \left(\mathbf{S}^{-1}\right)_{ij}$, where $\mathbf{S}^{-1}$ denotes the pseudoinverse if $\mathbf{S}$ is not invertible [8, Theorem 4].

The following graph-based centralized semi-supervised classification methods fall into this category:

- In Tikhonov regularization, $\mathbf{S} = \mathcal{L}^r$ (e.g., [14])
- Zhou *et al.* [15] take $\mathbf{S} = \mathcal{L}_{\text{norm}}^r$, where $\mathcal{L}_{\text{norm}} := \mathbf{D}^{-\frac{1}{2}} \mathcal{L} \mathbf{D}^{-\frac{1}{2}}$
- Smola and Kondor [8] consider a variety of kernel methods, including a diffusion process with $\mathbf{S} = \left[\exp\left(\frac{-\beta^2}{2}\mathcal{L}_{\text{norm}}\right)\right]^{-1}$, an inverse cosine with $\mathbf{S} = \left[\cos\left(\frac{\pi}{4}\mathcal{L}_{\text{norm}}\right)\right]^{-1}$, and an r-step random walk with $\mathbf{S} = (\beta \mathbf{I}_N - \mathcal{L}_{\text{norm}})^{-r}$, where $\beta \geq 2$ and $\mathbf{I}_N$ is the $N \times N$ identity matrix
- Ando and Zhang's K-scaling method [18], [19] takes
$$\mathbf{S} = (\gamma \mathbf{I}_N + \mathbf{D})^{-\frac{1}{2}} (\gamma \mathbf{I}_N + \mathcal{L}) (\gamma \mathbf{I}_N + \mathbf{D})^{-\frac{1}{2}},$$
which reduces to $\mathcal{L}_{\text{norm}}$ when $\gamma = 0$
- Zhu *et al.* [17, Chapter 15] take the kernel approach a step further by solving a convex optimization problem to find a good $\mathbf{S}$

Before moving on to the distributed semi-supervised classification problem, we note that in all of the examples above, we can write $\mathbf{S} = h(\mathbf{P})$ for some $h(\cdot)$, where $\mathbf{P}$ is either the combinatorial graph Laplacian, the normalized graph Laplacian, or the matrix $\mathbf{S}$ used in the K-scaling method, all of which have the same sparsity pattern as $\mathcal{L}$ and are easily computable from the weighted adjacency matrix.

Now, $\mathbf{F}^{\text{opt}}$ in (10) can be equivalently rewritten as the solution to κ separate minimization problems, with

$$\begin{aligned} \mathbf{F}_{:,j}^{\text{opt}} &= \underset{\mathbf{f} \in \mathbb{R}^N}{\operatorname{argmin}} \left\{ \tau \|\mathbf{f} - \mathbf{Y}_{:,j}\|_2^2 + \mathbf{f}^{\text{T}}\mathbf{S}\mathbf{f} \right\} \\ &= \underset{\mathbf{f} \in \mathbb{R}^N}{\operatorname{argmin}} \left\{ \tau \|\mathbf{f} - \mathbf{Y}_{:,j}\|_2^2 + \mathbf{f}^{\text{T}} h(\mathbf{P}) \mathbf{f} \right\}. \end{aligned} \tag{12}$$

We can write the solution to (12) as $\mathbf{R}\mathbf{Y}_{:,j}$, where $\mathbf{R}$ is a graph multiplier operator of the form outlined in Definition 1, with respect to $\mathbf{P}$. The optimal multiplier is $g(\lambda_\ell) = \frac{\tau}{\tau + h(\lambda_\ell)}$.

Therefore, the following is a method to distribute any of the centralized semi-supervised classification methods that can be written as (9) and (10):

1) Node n starts with or computes the entries of the n^{th} row of $\mathbf{P}$
2) Each node n forms the n^{th} row of $\mathbf{Y}$
3) For every $j \in \{1, 2, \ldots, \kappa\}$, the nodes approximately compute $\mathbf{F}_{:,j}^{\text{opt}} := \mathbf{R}\mathbf{Y}_{:,j}$ in a distributed manner via algorithms outlined in the subsequent sections.
4) Each node n with an unlabeled data point computes its label estimate according to $\arg\max_{j \in \{1,2,\ldots,\kappa\}} \left\{ F_{nj}^{\text{opt}} \right\}$

IV. Distributed Chebyshev Polynomial Approximation of Graph Multiplier Operators

Motivated by the fact that a number of distributed signal processing tasks can be viewed as applications of unions of graph multiplier operators, we proceed to the issue of how to approximately compute $\mathbf{\Phi}\mathbf{f}$, where $\mathbf{\Phi}$ is of the form (2), in a distributed setting. In this section, we introduce a computationally efficient approximation to unions of graph multiplier operators based on shifted Chebyshev polynomials.

A. The Centralized Chebyshev Polynomial Approximation

Exactly computing $g(\mathbf{P})\mathbf{f}$ requires explicit computation of the entire set of eigenvectors and eigenvalues of $\mathbf{P}$, which becomes computationally challenging as the size of the network, N, increases, even in a centralized setting. Druskin and Knizhnerman [26] introduce a method to approximate $g(\mathbf{P})\mathbf{f}$ by $\tilde{g}(\mathbf{P})\mathbf{f}$, where $\tilde{g}(\cdot)$ is a polynomial approximation of $g(\cdot)$ computed by truncating a shifted Chebyshev series expansion of the function $g(\cdot)$ on the interval $[\lambda_{\min}, \lambda_{\max}]$. Doing so circumvents the need to compute the full set of eigenvectors and eigenvalues of $\mathbf{P}$. This idea is extended to unions of graph Fourier multipliers in [25, Section 6]; that is, a computationally efficient approximation $\tilde{\mathbf{\Phi}}\mathbf{f}$ of $\mathbf{\Phi}\mathbf{f}$ can be computed by approximating each multiplier $g_j(\cdot)$ by a truncated series of shifted Chebyshev polynomials. We summarize this approach below.

For $y \in [-1, 1]$, the Chebyshev polynomials $\{T_k(y)\}_{k=0,1,2,\ldots}$ are generated by

$$T_k(y) := \begin{cases} 1, & \text{if } k = 0 \\ y, & \text{if } k = 1 \\ 2yT_{k-1}(y) - T_{k-2}(y), & \text{if } k \geq 2 \end{cases}.$$

These Chebyshev polynomials form an orthogonal basis for $L^2\left([-1,1], \frac{dy}{\sqrt{1-y^2}}\right)$. So every function h on $[-1, 1]$ that is square integrable with respect to the measure $dy/\sqrt{1-y^2}$ can be represented as $h(y) = \frac{1}{2}b_0 + \sum_{k=1}^{\infty} b_k T_k(y)$, where $\{b_k\}_{k=0,1,\ldots}$ is a sequence of Chebyshev coefficients that depends on $h(\cdot)$. For a detailed overview of Chebyshev polynomials, including the above definitions and properties, see [46]–[50].

By shifting the domain of the Chebyshev polynomials to $[0, \lambda_{\max}]$ via the transformation $x = \frac{\lambda_{\max}}{2}(y+1)$, we can represent each multiplier as

$$g_j(x) = \frac{1}{2}c_{j,0} + \sum_{k=1}^{\infty} c_{j,k}\overline{T}_k(x), \text{ for all } x \in [0, \lambda_{\max}], \quad (13)$$

where $\overline{T}_k(x) := T_k\left(\frac{x-\alpha}{\alpha}\right)$, $\alpha := \frac{\lambda_{\max}}{2}$, and[3]

$$c_{j,k} := \frac{2}{\pi}\int_0^{\pi} \cos(k\phi)\; g_j\Big(\alpha\big(\cos(\phi)+1\big)\Big)\; d\phi. \quad (14)$$

For $k \geq 2$, the shifted Chebyshev polynomials satisfy

$$\overline{T}_k(x) = \frac{2}{\alpha}(x-\alpha)\overline{T}_{k-1}(x) - \overline{T}_{k-2}(x).$$

Thus, for any $\mathbf{f} \in \mathbb{R}^N$, we have

$$\overline{T}_k(\mathbf{P})\mathbf{f} = \frac{2}{\alpha}(\mathbf{P} - \alpha\mathbf{I})\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{f}\right) - \overline{T}_{k-2}(\mathbf{P})\mathbf{f}, \quad (15)$$

where $\overline{T}_k(\mathbf{P}) \in \mathbb{R}^{N\times N}$ and, by (3), the n^{th} element of $\overline{T}_k(\mathbf{P})\mathbf{f}$ is given by

$$\left(\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n) = \sum_{\ell=0}^{N-1} \overline{T}_k(\lambda_\ell)\langle \mathbf{f}, \mathbf{u}_\ell\rangle \mathbf{u}_\ell(n). \quad (16)$$

Now, to approximate the operator Φ, we can approximate each multiplier $g_j(\cdot)$ by the first $K+1$ terms in its Chebyshev polynomial expansion (13). Then, for every $j \in \{1, 2, \ldots, \eta\}$ and $n \in \{1, 2, \ldots, N\}$, we have

$$\begin{aligned}
&\left(\tilde{\mathbf{\Phi}}\mathbf{f}\right)((j-1)N+n) \\
&:= \left(\frac{1}{2}c_{j,0}\mathbf{f} + \sum_{k=1}^{K} c_{j,k}\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n) \quad (17) \\
&\overset{(16)}{=} \sum_{\ell=0}^{N-1}\left[\frac{1}{2}c_{j,0} + \sum_{k=1}^{K} c_{j,k}\overline{T}_k(\lambda_\ell)\right]\langle \mathbf{f}, \mathbf{u}_\ell\rangle \mathbf{u}_\ell(n) \\
&\approx \sum_{\ell=0}^{N-1}\left[\frac{1}{2}c_{j,0} + \sum_{k=1}^{\infty} c_{j,k}\overline{T}_k(\lambda_\ell)\right]\langle \mathbf{f}, \mathbf{u}_\ell\rangle \mathbf{u}_\ell(n) \\
&\overset{(13)}{=} \sum_{\ell=0}^{N-1} g_j(\lambda_\ell)\langle \mathbf{f}, \mathbf{u}_\ell\rangle \mathbf{u}_\ell(n) \\
&\overset{(3)}{=} (\mathbf{\Phi}\mathbf{f})((j-1)N+n).
\end{aligned}$$

To recap, we propose to compute $\tilde{\mathbf{\Phi}}\mathbf{f}$ by first computing the Chebyshev coefficients $\{c_{j,k}\}_{j=1,2,\ldots,\eta;\ k=1,2,\ldots,K}$ according to (14), and then computing the sum in (17). The computational benefit of the Chebyshev polynomial approximation arises in (17) from the fact the vector $\overline{T}_k(\mathbf{P})\mathbf{f}$ can be computed recursively from $\overline{T}_{k-1}(\mathbf{P})\mathbf{f}$ and $\overline{T}_{k-2}(\mathbf{P})\mathbf{f}$ according to (15). The computational cost of doing so is dominated by the cost of matrix-vector multiplication with $\mathbf{P}$, which is proportional to the number of edges, $|\mathcal{E}|$ [25]. Therefore, if the underlying communication graph is sparse (i.e., $|\mathcal{E}|$ scales linearly with the network size N), it is far more computationally efficient to compute $\tilde{\mathbf{\Phi}}\mathbf{f}$ than $\mathbf{\Phi}\mathbf{f}$.

The approximation order K required to achieve a desired approximation error depends on the smoothness of the function $g(\cdot)$ and the magnitudes of its derivatives. In practice, for smooth functions $g(\cdot)$ (say twice continuously differentiable without large derivatives), choosing K on the order of 20 to 30 has led to close enough approximations for the applications we have examined. Alternatively, one can implement an adaptive procedure to choose K based on the magnitudes of the computed Chebyshev coefficients, by increasing K until these magnitudes decay below some pre-specified tolerance level. Such a procedure is implemented in the Chebfun package [50]. For ideal filters, [52, Section 4.3] presents a similar method for choosing K. See Section IV-E for more details on theoretical bounds on the approximation error.

B. *Distributed Computation of* $\tilde{\mathbf{\Phi}}\mathbf{f}$

We now discuss the second benefit of the Chebyshev polynomial approximation: it is easily distributable. We consider the following scenario. There is a network of N nodes, and each node n begins with the following knowledge:

- f_n, the n^{th} component of the signal $\mathbf{f}$
- The identity of its neighbors, and the weights of the graph edges connecting itself to each of its neighbors
- The Chebyshev coefficients, $c_{j,k}$, for $j \in \{1, 2, \ldots, \eta\}$ and $k \in \{0, 1, 2, \ldots, K\}$. These can either be computed centrally according to (14) and then transmitted throughout the network, or each node can begin with knowledge of the multipliers, $\{g_j(\cdot)\}_{j=1,2,\ldots,\eta}$, and precompute the Chebyshev coefficients according to (14)
- An upper bound $\overline{\lambda_{\max}}$ on $\lambda_{\max}$, the largest eigenvalue of $\mathbf{P}$. This bound need not be tight. For example, when $\mathbf{P}$ is the graph Laplacian $\mathcal{L}$, we can precompute a bound such as $\lambda_{\max} \leq \max\{d(m)+d(n); m \sim n\}$, where $d(n)$ is the degree of node n [53][54, Corollary 3.2]. More generally, an upper bound can be generated from a few steps of the Lanczos algorithm [55]

The task is for each network node n to compute

$$\left\{\left(\tilde{\mathbf{\Phi}}\mathbf{f}\right)\Big((j-1)N+n\Big)\right\}_{j=1,2,\ldots,\eta} \quad (18)$$

by iteratively exchanging messages with its local neighbors in the network and performing some computations.[4]

As a result of (17), for node n to compute the desired sequence in (18), it suffices to learn $\left\{\left(\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n)\right\}_{k=1,2,\ldots,K}$. Note that $\left(\overline{T}_1(\mathbf{P})\mathbf{f}\right)(n) = \left(\frac{1}{\alpha}(\mathbf{P}-\alpha\mathbf{I})\mathbf{f}\right)(n)$ and $P_{n,m} = 0$ for all nodes m that are not neighbors of node n. Thus, to compute $\left(\overline{T}_1(\mathbf{P})\mathbf{f}\right)(n)$, node n just needs to receive $\mathbf{f}(m)$ from all neighbors m. So once all nodes send their component

[3]The integral on the right-hand side of (14) often needs to be approximated numerically. The default method to do so in the Graph Signal Processing Toolbox (GSPBox) [51] is the general purpose trapezium quadrature rule described in [46, Section 5.2.2]. However, the GSPBox also allows the user to specify that this integral should be approximated with the open source Chebfun toolbox [50], which uses more nuanced quadrature methods that can, e.g., split the integral into multiple parts to perform the numerical approximation. We leverage the Chebfun toolbox in the numerical experiments in this paper.

[4]In practical implementations, it is important to ensure that the nodes remain in sync and/or the computations are robust to synchronization errors. These synchronization issues fall outside the scope of this work, but are considered in detail elsewhere [56]-[58].

Algorithm 1 Distributed Computation of $\tilde{\mathbf{\Phi}}\mathbf{f}$

Inputs at node n: f_n, $P_{n,m}\ \forall m$, $\{c_{k,j}\}_{j=1,2,\ldots,\eta;\ k=0,1,\ldots,K}$, and $\overline{\lambda_{\max}}$

Outputs at node n: $\left\{\left(\tilde{\mathbf{\Phi}}\mathbf{f}\right)((j-1)N+n)\right\}_{j=1,2,\ldots,\eta}$

```
1: Set $\alpha = \frac{\overline{\lambda_{\max}}}{2}$
2: Set $\left(\overline{T}_0(\mathbf{P})\mathbf{f}\right)(n) = \mathbf{f}(n)$
3: Transmit $f_n$ to all neighbors $\mathcal{N}_n := \{m : P_{n,m} \neq 0\}$
4: Receive $f_m$ from all neighbors $\mathcal{N}_n$
5: Compute and store
     $$\left(\overline{T}_1(\mathbf{P})\mathbf{f}\right)(n) = \left[\sum_{m\in\mathcal{N}_n\cup n} \frac{1}{\alpha} P_{n,m}\mathbf{f}(m)\right] - \mathbf{f}(n)$$
6: for $k = 2,\ldots,K$ do
7:   Transmit $\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{f}\right)(n)$ to all neighbors $\mathcal{N}_n$
8:   Receive $\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{f}\right)(m)$ from all neighbors $\mathcal{N}_n$
9:   Compute and store
       $$\begin{aligned}\left(\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n) &= \sum_{m\in\mathcal{N}_n\cup n} \frac{2}{\alpha} P_{n,m}\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{f}\right)(m) \\ &\quad - 2\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{f}\right)(n) - \left(\overline{T}_{k-2}(\mathbf{P})\mathbf{f}\right)(n)\end{aligned}$$
10: end for
11: for $j \in \{1,2,\ldots,\eta\}$ do
12:   Output
       $$\begin{aligned}\left(\tilde{\mathbf{\Phi}}\mathbf{f}\right)((j-1)N+n) &= \frac{1}{2}c_{j,0}\mathbf{f}(n) + \sum_{k=1}^{K} c_{j,k}\left(\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n)\end{aligned}$$
13: end for
```

of the signal to their neighbors, they are able to compute their respective components of $\overline{T}_1(\mathbf{P})\mathbf{f}$. In the next step, each node n sends the newly computed quantity $\left(\overline{T}_1(\mathbf{P})\mathbf{f}\right)(n)$ to all of its neighbors, enabling the distributed computation of $\overline{T}_2(\mathbf{P})\mathbf{f}$ according to (15). The iterative process of local communication and computation continues for K rounds until each node n has computed the required sequence $\left\{\left(\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n)\right\}_{k=1,2,\ldots,K}$. In all, since each edge results in one message passed in each direction in each iteration, $2K|\mathcal{E}|$ messages of length 1 are required for every node n to compute its sequence of coefficients in (18) in a distributed fashion. This distributed computation process is summarized in Algorithm 1.

An important point to emphasize again is that although the operator Φ and its approximation $\tilde{\Phi}$ are defined through the eigenvectors of $\mathbf{P}$, the Chebyshev polynomial approximation helps the nodes apply the operator to the signal without explicitly computing (individually or collectively) the eigenvalues or eigenvectors of $\mathbf{P}$, other than the upper bound on its spectrum. Rather, they initially communicate their component of the signal to their neighbors, and then communicate simple weighted combinations of the messages received in the previous stage in subsequent iterations. In this way, information about each component of the signal $\mathbf{f}$ diffuses through the network without direct communication between non-neighboring nodes.

Algorithm 2 Distributed Computation of $\tilde{\mathbf{\Phi}}^*\mathbf{a}$

Inputs at node n: $\{\mathbf{a}_j(n)\}_{j=1,2,\ldots,\eta}$, $P_{n,m}\ \forall m$, $\overline{\lambda_{\max}}$, and $\{c_{k,j}\}_{j=1,2,\ldots,\eta;\ k=0,1,\ldots,K}$,

Output at node n: $\left(\tilde{\mathbf{\Phi}}^*\mathbf{a}\right)(n)$

```
1: Set $\alpha = \frac{\overline{\lambda_{\max}}}{2}$
2: for $j = 1,2,\ldots,\eta$ do
3:   Set $\left(\overline{T}_0(\mathbf{P})\mathbf{a_j}\right)(n) = \mathbf{a}_j(n)$
4: end for
5: Transmit $\{\mathbf{a}_j(n)\}_{j=1,2,\ldots,\eta}$ to all neighbors $\mathcal{N}_n := \{m : P_{n,m} \neq 0\}$
6: Receive $\{\mathbf{a}_j(m)\}_{j=1,2,\ldots,\eta}$ from all neighbors $\mathcal{N}_n$
7: for $j = 1,2,\ldots,\eta$ do
8:   Compute and store
       $$\left(\overline{T}_1(\mathbf{P})\mathbf{a_j}\right)(n) = \left[\sum_{m\in\mathcal{N}_n\cup n} \frac{2}{\alpha} P_{n,m}\mathbf{a}_j(m)\right] - 2\mathbf{a}_j(n)$$
9: end for
10: for $k = 2,\ldots,K$ do
11:   Transmit $\left\{\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{a_j}\right)(n)\right\}_{j=1,2,\ldots,\eta}$ to all neighbors $\mathcal{N}_n$
12:   Receive $\left\{\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{a_j}\right)(m)\right\}_{j=1,2,\ldots,\eta}$ from all neighbors $\mathcal{N}_n$
13:   for $j = 1,2,\ldots,\eta$ do
14:     Compute and store
          $$\begin{aligned}\left(\overline{T}_k(\mathbf{P})\mathbf{a_j}\right)(n) &= \sum_{m\in\mathcal{N}_n\cup n} \frac{2}{\alpha} P_{n,m}\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{a_j}\right)(m) \\ &\quad - 2\left(\overline{T}_{k-1}(\mathbf{P})\mathbf{a_j}\right)(n) - \left(\overline{T}_{k-2}(\mathbf{P})\mathbf{a_j}\right)(n)\end{aligned}$$
15:   end for
16: end for
17: Output
       $$\begin{aligned}\left(\tilde{\mathbf{\Phi}}^*\mathbf{a}\right)(n) &= \sum_{j=1}^{\eta}\left\{\frac{1}{2}c_{j,0}\mathbf{a}_j(n) + \sum_{k=1}^{K} c_{j,k}\left(\overline{T}_k(\mathbf{P})\mathbf{a_j}\right)(n)\right\}.\end{aligned}$$
```

C. Distributed Computation of $\tilde{\mathbf{\Phi}}^*\mathbf{a}$ and $\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{\Phi}}\mathbf{f}$

In some tasks, such as the distributed lasso presented in Section VI, we not only need to apply unions of graph multiplier operators, but we also need to apply their adjoints. The application of the adjoint $\tilde{\Phi}^*$ of the Chebyshev polynomial approximate operator $\tilde{\Phi}$ can also be computed in a distributed manner. Let $\mathbf{a} = [\mathbf{a}_1; \mathbf{a}_2; \ldots; \mathbf{a}_\eta] \in \mathbb{R}^{\eta N}$, where $\mathbf{a}_j \in \mathbb{R}^N$.

Then it is straightforward to show that

$$\left(\tilde{\mathbf{\Phi}}^{*}\mathbf{a}\right)(n)=\sum_{j=1}^{\eta}\left(\frac{1}{2}c_{j,0}\mathbf{a}_j+\sum_{k=1}^{K}c_{j,k}\overline{T}_k(\mathbf{P})\mathbf{a_j}\right)(n). \quad (19)$$

We assume each node n starts with knowledge of $\mathbf{a}_j(n)$ for all $j\in\{1,2,\dots,\eta\}$. For each $j\in\{1,2,\dots,\eta\}$, the distributed computation of the corresponding term on the right-hand side of (19) is done in an analogous manner to the distributed computation of $\tilde{\mathbf{\Phi}}\mathbf{f}$ discussed above. Since this has to be done for each j, $2K|\mathcal{E}|$ messages, each a vector of length η, are required for every node n to compute $(\tilde{\mathbf{\Phi}}^{*}\mathbf{a})(n)$. The distributed computation of $\tilde{\mathbf{\Phi}}^{*}\mathbf{a}$ is summarized in Algorithm 2.

Using the property of the Chebyshev polynomials that $T_k(x)T_{k'}(x)=\frac{1}{2}\left[T_{k+k'}(x)+T_{|k-k'|}(x)\right]$, we can write

$$\left(\tilde{\mathbf{\Phi}}^{*}\tilde{\mathbf{\Phi}}\mathbf{f}\right)(n)=\left(\frac{1}{2}d_0\mathbf{f}+\sum_{k=1}^{2K}d_k\overline{T}_k(\mathbf{P})\mathbf{f}\right)(n).$$

See [25, Section 6.1] for a similar calculation and an explicit formula for the coefficients $\{d_k\}_{k=0,1,\dots,2K}$. Thus, with each node n starting with $\mathbf{f}(n)$ as in Section IV-B, $\tilde{\mathbf{\Phi}}^{*}\tilde{\mathbf{\Phi}}\mathbf{f}$ can be distributedly computed using $4K|\mathcal{E}|$ messages of length 1, with each node n finishing with knowledge of $\left(\tilde{\mathbf{\Phi}}^{*}\tilde{\mathbf{\Phi}}\mathbf{f}\right)(n)$.

D. Numerical Example

We place 500 sensors randomly in the $[0,1]\times[0,1]$ square. We then construct a weighted graph according to a thresholded Gaussian kernel weighting function based on the physical distance between nodes. The weight of edge e connecting nodes i and j that are a distance $d(i,j)$ apart is

$$w(e)=\begin{cases}\exp\left(-\frac{[d(i,j)]^2}{2\sigma^2}\right) & \text{if } d(i,j)\le\kappa\\ 0 & \text{otherwise}\end{cases},$$

with parameters $\sigma=0.074$ and $\kappa=0.075$. We create a smooth 500-dimensional signal with the n^{th} component given by $h_n=n_x^2+n_y^2-1$, where n_x and n_y are node n's x and y coordinates in $[0,1]\times[0,1]$. Next, we corrupt each component of the signal $\mathbf{h}$ with uncorrelated additive Gaussian noise with mean zero and standard deviation 0.5, resulting in a noisy signal $\mathbf{y}$. Then we apply the graph Fourier multiplier operator $\tilde{\mathbf{R}}$, the Chebyshev polynomial approximation to $\mathbf{R}$ from Proposition 2, with $\tau=r=1$ and $K=20$. The original signal $\mathbf{h}$, noisy signal $\mathbf{y}$, and denoised signal $\tilde{\mathbf{R}}\mathbf{y}$ are shown in Figure 1(a)-(c). The Chebyshev polynomial approximation errors are shown in Figure 1(d), and the resulting approximation errors for the graph Fourier multiplier operator and denoised signal are shown in Figure 1(e). We repeated this entire experiment 1000 times, with a new random graph and random noise each time, and the average mean square error for the denoised signals was 0.013, as compared to 0.250 average mean square error for the noisy signals.[5]

[5]The reported errors are averaged over the 441 random graph realizations that were connected.

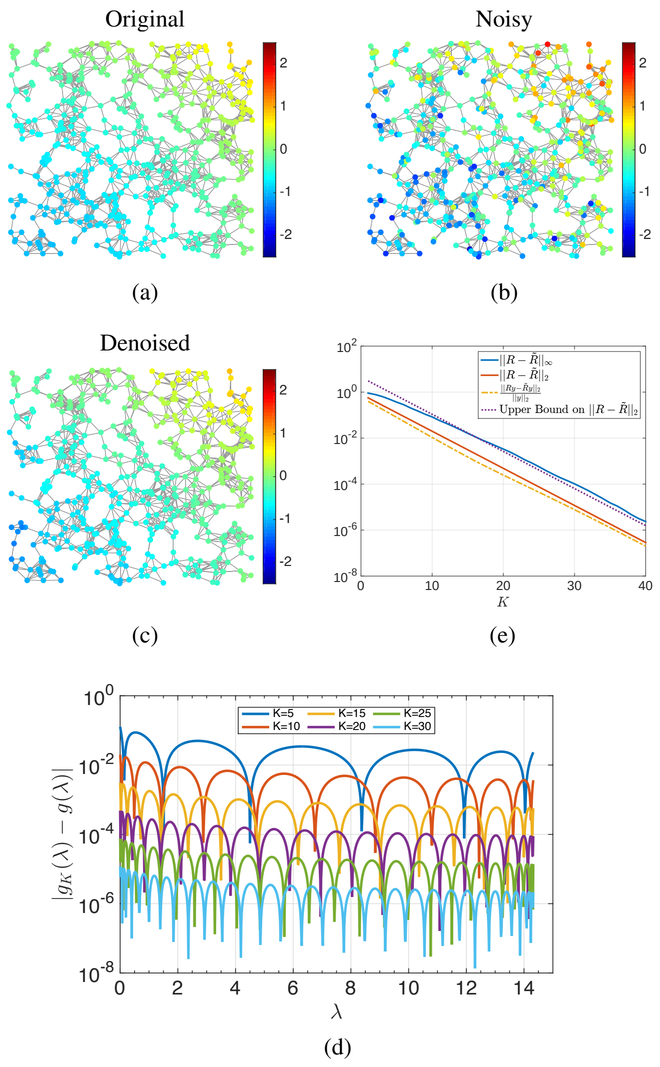

Fig. 1. Distributed denoising example. (a) The original signal with $h_n=n_x^2+n_y^2-1$, where n_x and n_y are the x and y coordinates of sensor node n. (b) The noisy signal $\mathbf{y}$. (c) The denoised signal $\tilde{\mathbf{R}}\mathbf{y}$, the Chebyshev polynomial approximation (order $K=20$) to $\mathbf{R}\mathbf{y}=\sum_{\ell=0}^{N-1}\frac{1}{1+2\lambda_\ell}\hat{y}(\ell)\chi_\ell(n)$. (d) Approximation errors of shifted Chebyshev polynomial expansions of different orders for the filter $\hat{g}(\lambda_\ell)=\frac{1}{1+2\lambda_\ell}$. (e) Resulting approximation errors for the graph Fourier multiplier operator and graph filtered signal, along with the upper bound on $||\mathbf{R}-\tilde{\mathbf{R}}||_2$ from (21)-(23).

E. Approximation Error

We use the following result, which bounds the spectral norm of the difference between a union of graph multiplier operators and its Chebyshev polynomial approximation, to analyze the distributed lasso problem in Section VI.

Proposition 4: Let $\mathbf{\Phi}$ be a union of η graph multiplier operators; i.e., it has the form given in (2) for a real symmetric positive semi-definite matrix $\mathbf{P}$. Let $\tilde{\mathbf{\Phi}}$ be the order K Chebyshev polynomial approximation of $\mathbf{\Phi}$. Define

$$B(K):=\max_{j=1,2,\dots,\eta}\left\{\sup_{\lambda\in[0,\lambda_{\max}]}\left\{\left|g_j(\lambda)-p_j^K(\lambda)\right|\right\}\right\}, \quad (20)$$

where $\lambda_{\max}$ is the largest eigenvalue of $\mathbf{P}$, and $p_j^K(\cdot)$ is the order K Chebyshev polynomial approximation of $g_j(\cdot)$. Then

$$\|\mathbf{\Phi}-\tilde{\mathbf{\Phi}}\|_2 := \max_{\mathbf{f}\neq\mathbf{0}} \frac{\|(\mathbf{\Phi}-\tilde{\mathbf{\Phi}})\mathbf{f}\|_2}{\|\mathbf{f}\|_2} \leq B(K)\sqrt{\eta}. \tag{21}$$

The proof of Proposition 4 is included in the Appendix of [44].

When the multipliers $g_j(\cdot)$ are smooth, the Chebyshev approximations $p_j^K(\cdot)$ converge to the multipliers rapidly as K increases. The following proposition characterizes this convergence for continuously differentiable multipliers.

***Proposition** 5 (Theorem 5.14 in [46]):* If $g_j(\cdot)$ has $M+1$ continuous derivatives for all j, then $B(K)=\mathcal{O}\left(K^{-M}\right)$.

When each multiplier $g_j(\cdot)$ is real analytic on $[0,\lambda_{\max}]$, we can make a stronger statement about the convergence. Using the parametrization $\lambda=\frac{\lambda_{\max}}{2}(x+1)$ with $x\in[-1,1]$ and setting $h_j(x):=g_j(\frac{\lambda_{\max}}{2}(x+1))$, the function $h_j(\cdot)$ admits an analytic extension to an open Bernstein ellipse of radius $\rho_j>1$. Setting $\rho=\min\{\rho_1,\ldots,\rho_\eta\}>1$ and letting C denote the supremum of all $|h_j(\cdot)|$ on the open Bernstein ellipse of radius $\rho>1$, we have

$$B(K)\leq\frac{2C}{\rho-1}\rho^{-K}=\mathcal{O}\left(\rho^{-K}\right); \tag{22}$$

see, e.g., [46, Theorem 5.16], [49, Theorem 8.2]. For example, for $g(\lambda)=\frac{1}{1+2\lambda}$, using the formula in [49, Equation 8.4] for computing ρ when there is a real singularity,

$$\rho=1+\frac{1}{\lambda_{\max}}\left(1+\sqrt{1+2\lambda_{\max}}\right). \tag{23}$$

This upper bound on $B(K)$ (and in turn by (21) on $||\mathbf{R}-\tilde{\mathbf{R}}||_2$) is shown in Figure 1(e).

V. Other Distributed Methods for Computing $g(\mathbf{P})\mathbf{y}$

In this section, we discuss some other methods for computing $g(\mathbf{P})\mathbf{y}$ in a distributed setting. Most of these variations are not distributed computation methods *per se*, but rather centralized computational methods that can be distributed in the context of the applications mentioned above.

Higham [39, Chapter 13], as well as Frommer and Simoncini [59] provide excellent introductory overviews of centralized methods to compute $g(\mathbf{P})\mathbf{y}$ for large, sparse $\mathbf{P}$. Of the methods mentioned there, we do not consider contour integral or Krylov subspace methods, which are not readily amenable to distributed computation. For example, in a distributed setting, the Lanczos method [26], [60] would require a significant amount of extra communication at each iteration to compute vector norms. We also do not consider conjugate gradient or algebraic multigrid methods, which could be used for a few specific choices of $g(\cdot)$. For example, if $g(\lambda)=\frac{1}{1+2\lambda}$, then we can compute $\mathbf{x}=g(\mathbf{P})\mathbf{f}$ by solving the linear system of equations $g^{-1}(\mathbf{P})\mathbf{x}=(\mathbf{I}+2\mathbf{P})\mathbf{x}=\mathbf{f}$ via these methods.

A. Jacobi's Iterative Method

For $\mathbf{S}=\mathbf{P}=\mathcal{L}_{\text{norm}}$, Zhou *et al.* [15] propose to solve the semi-supervised classification problem (10) through the iteration

$$\mathbf{F}^{(t+1)}=\frac{1}{1+\tau}\left[\left(\mathbf{I}_N-\mathbf{P}\right)\mathbf{F}^{(t)}+\tau\mathbf{Y}\right], \quad t=0,1,\ldots,T-1, \tag{24}$$

where $\mathbf{F}^{(0)}$ is arbitrary (set to $\mathbf{Y}$ in [15]).[6] The iteration (24) is in fact just a particular instance of Jacobi's iterative method (see, e.g., [61, Chapter 4]) to solve the set of linear equations

$$\left(\tau\mathbf{I}_N+\mathbf{P}\right)\mathbf{F}^{\text{opt}}=\tau\mathbf{Y}. \tag{25}$$

So one alternative distributed semi-supervised classification method with $\mathbf{S}=\mathbf{P}=\mathcal{L}_{\text{norm}}$ is to compute the iterations (24) in a distributed manner, with each node starting with knowledge of its row of $\mathbf{P}$ and $\mathbf{Y}$. In fact, the communication cost of one iteration of (24) is the same as the communication cost of one iteration of the distributed computation of $\tilde{\mathbf{R}}\mathbf{Y}$ (lines 6 and 7 of Algorithm 1).

For graph multiplier operators whose multipliers have the property $g(\lambda_\ell)\neq 0$ for all ℓ, the Jacobi method generalizes as follows. Suppose we wish to compute $\mathbf{R}\mathbf{y}$, where $\mathbf{R}$ is a graph multiplier operator with respect to $\mathbf{P}$ and with multiplier $g(\cdot)$. This is equivalent to solving the linear system of equations $g(\mathbf{P})^{-1}\mathbf{x}=\mathbf{y}$. Assuming that the entries of the matrix $\mathbf{Q}=g(\mathbf{P})^{-1}$ are convenient to evaluate (e.g., for certain rational functions g), let $\mathbf{Q}=\mathbf{Q}_D-\mathbf{Q}_O$, where $\mathbf{Q}_D$ contains the diagonal part of $\mathbf{Q}$. Then the Jacobi iteration is

$$\mathbf{x}^{(t+1)}=\mathbf{Q}_D^{-1}\mathbf{Q}_O\mathbf{x}^{(t)}+\mathbf{Q}_D^{-1}\mathbf{y},\ t=0,1,\ldots,T-1. \tag{26}$$

One immediate drawback of Jacobi's method, as compared with the Chebyshev polynomial method of Section IV, is that it does not always converge. The iterations in (26) converge for any $\mathbf{x}^{(0)}$ if and only if the spectral radius of $\mathbf{Q}_D^{-1}\mathbf{Q}_O$ is less than one [61, Theorem 4.1]. One sufficient condition for the latter to be true is that $\mathbf{Q}$ is strictly diagonally dominant, as is the case for example when $\mathbf{P}=\mathcal{L}$ and $g(\lambda_\ell)=\frac{\tau}{\tau+\lambda_\ell}$. Additionally, it may be too expensive computationally to evaluate the matrix $\mathbf{Q}$, or it may be a dense matrix, in which case the communication cost of a distributed method becomes prohibitive. For example, if $g=e^{-t\lambda}$, it is not efficient to fully evaluate $\mathbf{Q}$ and so this method is not applicable.

B. Jacobi's Iterative Method with Chebyshev Acceleration

When Jacobi's method does converge, we can accelerate (26) using the following algorithm [62, Algorithm 6.7]. Let ρ be an upper bound on the spectral radius of $\mathbf{Q}_D^{-1}\mathbf{Q}_O$, and define $\xi^{(0)}:=1$, $\xi^{(1)}:=\rho$, and $\mathbf{x}^{(1)}:=\mathbf{Q}_D^{-1}\mathbf{Q}_O\mathbf{x}^{(0)}+\mathbf{Q}_D^{-1}\mathbf{y}$. Then for $t\geq 1$, let

$$\begin{aligned}\xi^{(t+1)}&=\frac{1}{\frac{2}{\rho\xi^{(t)}}-\frac{1}{\xi^{(t-1)}}}, \text{ and}\\ \mathbf{x}^{(t+1)}&=\frac{2\xi^{(t+1)}}{\rho\xi^{(t)}}\mathbf{Q}_D^{-1}\mathbf{Q}_O\mathbf{x}^{(t)}-\frac{\xi^{(t+1)}}{\xi^{(t-1)}}\mathbf{x}^{(t-1)}\\ &\quad+\frac{2\xi^{(t+1)}}{\rho\xi^{(t)}}\mathbf{Q}_D^{-1}\mathbf{y}.\end{aligned} \tag{27}$$

[6]In [17, Chapter 11], similar iterative label propagation methods from [12] and [16] are also compared with the method of [15].

To distribute (27), each node n must first learn Q_{nn} and the n^{th} row of $\mathbf{Q}_O$. For example, when $\mathbf{P} = \mathcal{L}_{\text{norm}}$ and $g(\lambda_\ell) = \frac{\tau}{\tau+\lambda_\ell}$, as in (24), $Q_{nn} = \frac{\tau+1}{\tau}$ for all n, and the n^{th} row of $\mathbf{Q}_O$ is just $-\frac{1}{\tau}$ times the n^{th} row of $\mathcal{L}_{\text{norm}}$. An additional challenge in a distributed setting may be to calculate the bound ρ.

Note that while this method and the method of Section IV share the same namesake, the use of the Chebyshev polynomials in the two is different. In Section IV, we use Chebyshev polynomials to approximate the multiplier, whereas this method improves the convergence speed of the Jacobi method by using Chebyshev polynomials to choose the weights it uses to form the iterates in (27) as weighted linear combinations of the iterates in (26). See Section 6.5.6 of [62] for more details.

C. Polynomial Approximation Variants

When a graph spectral filter $g(\cdot)$ is a polynomial, it is often referred to as a *finite impulse response* (FIR) filter [11], and we can write

$$g(\mathbf{P}) = \beta_0 \mathbf{I} + \sum_{k=1}^{K} \beta_k \mathbf{P}^k. \tag{28}$$

In this case, we can compute $g(\mathbf{P})\mathbf{f}$ in a distributed fashion through a nested multiplication iteration [63, Section 9.2.4], letting $\mathbf{x}^{(0)} = \beta_K \mathbf{f}$, and then iterating

$$\mathbf{x}^{(l)} = \beta_{K-l}\mathbf{f} + \mathbf{P}\mathbf{x}^{(l-1)},\ l = 1, 2, \ldots, K. \tag{29}$$

The result is $\mathbf{x}^{(K)} = g(\mathbf{P})\mathbf{f}$. Since each iteration of (29) requires the distributed computation of $\mathbf{P}\mathbf{x}$, this method requires the same amount of communication as the Chebyshev polynomial approximation of Algorithm 1.

When a graph spectral filter $g(\cdot)$ is not a polynomial (or a polynomial of order greater than K), any polynomial approximation method can be used to generate an order K polynomial approximation $\tilde{g}(\cdot)$ such that $\tilde{g}(\mathbf{P})$ is of the form (28). Sandryhaila and Moura suggest in [11] to find an order K polynomial $\tilde{g}(\cdot)$ such that the (usually overdetermined) system of equations $\{\tilde{g}(\lambda_\ell) = g(\lambda_\ell)\}_{\ell=0,1,\ldots,N-1}$ is approximately solved in the least squares sense. The drawback of this approach is that one needs access to all eigenvalues of $\mathbf{P}$. The continuous analog of this discrete approach is to compute an order K Legendre polynomial approximation $\tilde{g}(\cdot)$ to $g(\cdot)$ on the interval $[0, \lambda_{\max}]$. Then $\tilde{g}(\mathbf{P})\mathbf{f}$ can be computed in a distributed manner via (29) or through a three-term recurrence similar to step 9 of Algorithm 1 (c.f., [49, Equation 17.6]).

Other orthogonal polynomials can also be used to generate approximations via truncated expansions. For example, [64] uses Laguerre polynomials to approximate matrix exponentials. One advantage of this method is that Laguerre polynomials are orthonormal on $[0, \infty)$, so no upper bound on the spectrum is required. However, in the applications we consider, it is usually not hard to generate the upper bound $\overline{\lambda_{\max}}$, as discussed earlier in Section IV-B.

In [65], Chen et al. first approximate the filter g by a polynomial spline, and then compute orthogonal expansion coefficients of the spline in order to avoid the numerical integration involved in computing, e.g., the Chebyshev coefficients $\{c_k\}$ in (14). The conjugate residual-type algorithm of [66] also uses the spline approach. However, in [66], the order $K+1$ polynomial approximation of a highpass filter g takes the form $\tilde{g}(\lambda) = \lambda\varphi(\lambda)$, where φ is an order K polynomial, forcing $\tilde{g}(0)$ to be equal to zero. If g is a lowpass filter such as $g(\lambda) = e^{-\tau\lambda}$, then [66] takes the approximation to be of the form $\tilde{g}(\lambda) = 1 - \lambda\varphi(\lambda)$, with φ an order K polynomial, once again guaranteeing zero approximation error at $\lambda = 0$. This technique can be extended to bandpass filters by splitting the spectrum up into separate intervals, eventually leading to a three term recurrence with new weights that can be computed offline. A distributed implementation then carries the same communication cost as the single Chebyshev polynomial approximation.

This discussion begs the question of which polynomial approximation is best to use in these distributed signal processing tasks. Broadly speaking, the numerical analysis literature demonstrates that different families of orthogonal polynomials have similar approximation properties. For example, [49, p. 21] reports "Legendre points and polynomials are neither much better than Chebyshev for approximating functions, nor much worse; they are essentially the same." However, the main advantages of the Chebyshev polynomials are the ability to use the fast Fourier transform to compute the Chebyshev coefficients [49, p. 21 and p. 125], and the good conditioning of the Chebyshev polynomial basis [46, Section 6.3.4], [67].

D. Rational Approximations

An alternative to a polynomial approximation is a rational approximation (see, e.g., [59, Section 3.4]) of the form

$$g(\lambda) \approx \frac{\mathcal{N}_\mu(\lambda)}{\mathcal{D}_\nu(\lambda)} =: \tilde{\tilde{g}}(\lambda), \tag{30}$$

where $\mathcal{N}_\mu$ and $\mathcal{D}_\nu$ are polynomials of degree μ and ν, respectively. In the graph signal processing literature, references such as [36], [37] refer to filters of the form (30) as *infinite impulse response filters*, since we can not write them in the form of (28) for any choice of the order K and series of coefficients $\{\beta_k\}$.

One benefit of rational approximations of the form (30) is that they tend to provide better approximations than polynomials of lower orders, especially when g features a singularity close to the spectrum of $\mathbf{P}$. However, a major drawback is they tend to require extra subiterations, resulting in increased communication cost. For example, to compute $\mathbf{x} = \mathcal{D}_\nu^{-1}(\mathbf{P})\mathbf{y}$, [36] uses gradient descent to iteratively solve

$$\operatorname*{argmin}_{\mathbf{x}} ||\mathcal{D}_\nu(\mathbf{P})\mathbf{x} - \mathbf{y}||^2. \tag{31}$$

Yet, [36] estimates the number of iterations required to solve (31) as $\frac{\max_{\ell=0,1,\ldots,N-1}\{\mathcal{D}_\nu(\lambda_\ell)^2\}}{\min_{\ell=0,1,\ldots,N-1}\{\mathcal{D}_\nu(\lambda_\ell)^2\}}$. Each of these iterations requires twice as much communication as the full distributed computation of an order ν matrix polynomial computation via Algorithm 1 (with $\eta = 1$). So even when $\mathcal{N}_\mu$ and $\mathcal{D}_\nu$ are taken to be lower order polynomials, the communication requirements may still be significantly higher than a higher order polynomial approximation (where $\mathcal{D}_\nu(\lambda) = 1$).

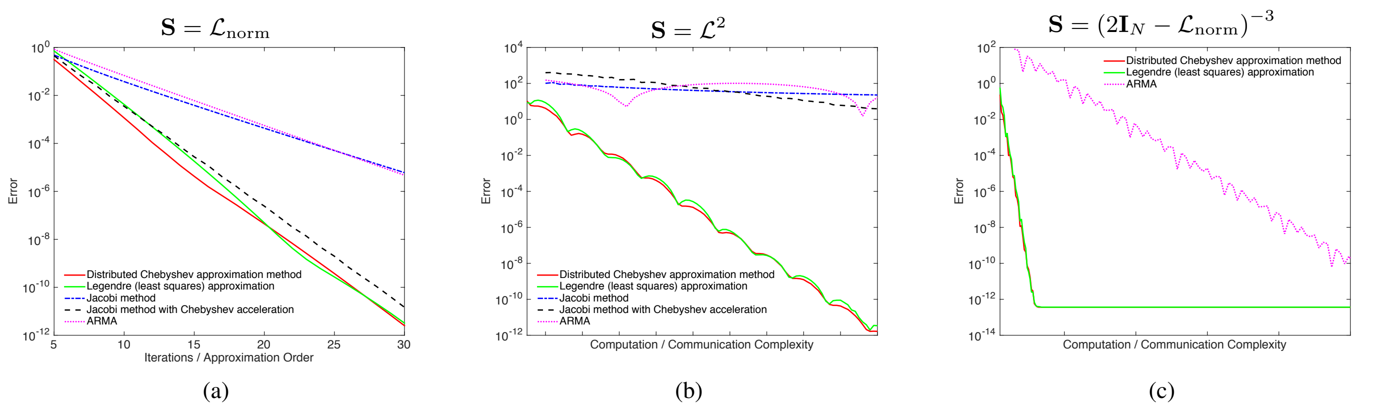

Fig. 2. Five different distributed methods to approximately compute $\mathbf{Ry}$, where $\mathbf{R}$ is a graph multiplier operator with respect to $\mathbf{P}$ for different choices of $\mathbf{P}$. In all cases, the multiplier is $g(\lambda_\ell) = \frac{\tau}{\tau + h(\lambda_\ell)}$. In (a), the error shown is $\|\mathbf{f}^{(K)} - \mathbf{f}\|_2$, where $\mathbf{f}^{(K)}$ is either $\tilde{\mathbf{R}}\mathbf{y}$ with an order K approximation for our distributed Chebyshev approximation method, or the result of the K^{th} iteration for the Jacobi and ARMA methods. In (b) and (c), since iterations of the different algorithms have different computation and communication complexities, we normalize to a common scale.

Some filters of the form (30) with $\mu \leq K$ and $\nu = K$ can also be written as

$$\tilde{\tilde{g}}(\lambda) = \sum_{k=1}^{K} \frac{2r_k}{\lambda_{\max} - \lambda_{\min} - 2\lambda - 2p_k}, \tag{32}$$

for some coefficient sequences $\{r_k\}$ and $\{p_k\}$. Loukas et al. [37] refer to such filters as parallel autoregressive moving average graph filters (ARMA) of order K, and show that if for all k, $|p_k| > \frac{\lambda_{\max} - \lambda_{\min}}{2}$, then $\mathbf{x} = \tilde{\tilde{g}}(\mathbf{P})\mathbf{y}$ can be computed by iterating the following recursion for each term in the summation on the right-hand side of (32):

$$\mathbf{x}_k^{(t+1)} = \frac{1}{p_k} \left[\left(\frac{\lambda_{\max} - \lambda_{\min}}{2} \right) \mathbf{I} - \mathbf{P} \right] \mathbf{x}_k^{(t)} - \frac{r_k}{p_k} \mathbf{y}, \quad t = 0, 1, \ldots, T-1 \tag{33}$$

and then summing these results to find $\mathbf{x} = \sum_{k=1}^{K} \mathbf{x}_k$. Once again, these ARMA filters have the potential to yield a better approximation than a finite impulse response filter of the form on the right-hand side of (28) with the same order K; however, they require T times the communication, where T is the number of times one must iterate (33) to convergence.

E. Numerical Comparison

We consider the same random sensor network shown in Figure 1, and we generate a signal $\mathbf{f}$ on the vertices of the graph with the components of $\mathbf{f}$ independently and identically sampled from a uniform distribution on $[-10, 10]$. For different choices of $h(\cdot)$ and $\mathbf{P} = \mathcal{L}$ or $\mathbf{P} = \mathcal{L}_{\text{norm}}$, we define

$$\mathbf{y} := \left(\mathbf{I}_{500} + \frac{1}{\tau} h(\mathbf{P}) \right) \mathbf{f} = g(\mathbf{P})\mathbf{f},$$

with $\tau = 0.5$ and $g(\lambda) = \frac{\tau + h(\lambda)}{\tau}$. Then, starting with $\mathbf{y}$, we iteratively compute an approximation to $\mathbf{f}$ in five different distributable ways: 1) $\tilde{\mathbf{R}}\mathbf{y}$, where $\tilde{\mathbf{R}}$ is the Chebyshev approximation to $\mathbf{R} = g(\mathbf{P})^{-1}$; 2) the same, but with the Legendre approximation to the filter $g^{-1}(\cdot)$; 3) with the Jacobi iteration (26); 4) with the Jacobi iteration with Chebyshev acceleration (27); and 5) the ARMA iteration (33).

When $\mathbf{P} = \mathcal{L}_{\text{norm}}$ and $\mathbf{S} = h(\mathbf{P}) = \mathcal{L}_{\text{norm}}$, the filter $g^{-1}(\cdot)$ is the ratio of a constant and a first order polynomial, so we can take $K = 1$ in (32). Taking the initial guess $\mathbf{x}^{(0)}$ to be $\mathbf{y}$ and $\lambda_{\min} = 0$, the iteration (33) becomes

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \frac{2}{\tau + \lambda_{\max}} \left[\left(\frac{\lambda_{\max}}{2} \mathbf{I} - \mathbf{P} \right) \mathbf{x}^{(t)} + \tau \mathbf{y} \right] \\ &= \frac{2\tau}{\tau + \lambda_{\max}} \mathbf{y} + \frac{\lambda_{\max}}{\tau + \lambda_{\max}} \mathbf{x}^{(t)} - \frac{2}{\tau + \lambda_{\max}} \mathbf{P}\mathbf{x}^{(t)}.\end{aligned}$$

In this case, the communication requirements of the polynomial methods with approximation order K are equal to the communication requirements of $T = K$ iterations of the latter three methods, so we plot the errors $\|\mathbf{f}^{(K)} - \mathbf{f}\|_2$ (where $\mathbf{f}^{(K)}$ corresponds to $\tilde{\mathbf{R}}\mathbf{y}$ with an order K approximation in the first two cases or the result of the K^{th} iteration in the latter three cases) on the same axes in Figure 2(a).

When $\mathbf{P} = \mathcal{L}$ and $\mathbf{S} = h(\mathbf{P}) = \mathcal{L}^2$, computing $\mathbf{Q}_O\mathbf{x}^{(t)}$ in (26) and (27) requires computing $\mathbf{W}\mathbf{x}^{(t)}$, which requires twice the communication and computation of a single iteration of Algorithm 1 for the polynomial approximations. For the ARMA approach, we can write the filter $g(\lambda) = \frac{\tau}{\tau + \lambda^2}$ exactly in the form of (32) with $p_1 = \sqrt{\tau} i + \frac{\lambda_{\max}}{2}$, $p_2 = -\sqrt{\tau} i + \frac{\lambda_{\max}}{2}$, $r_1 = -\frac{\sqrt{\tau} i}{2}$, and $r_2 = \frac{\sqrt{\tau} i}{2}$.

When $\mathbf{P} = \mathcal{L}_{\text{norm}}$ and $\mathbf{S} = h(\mathbf{P}) = (2\mathbf{I}_{500} - \mathcal{L}_{\text{norm}})^{-3}$ (a three-step random walk process), the Jacobi method does not converge. We have $h(\lambda) = (2 - \lambda)^{-3}$, and thus

$$g(\lambda) = \frac{\tau}{\tau + h(\lambda_\ell)} = 1 - \frac{2}{(2 - \lambda)^3 + 2},$$

the last term of which can be written as a third order ARMA filter.

Figure 2 compares the approximation error to the communication/computation complexity for each of these methods and choices of $\mathbf{S}$. In these experiments, not only do the polynomial methods always converge, but they converge faster and with less communication than the alternative methods we tested. Not surprisingly, the errors resulting from the Chebyshev and Legendre polynomial approximations are similar.

VI. Distributed Lasso

In Section III, we presented a number of distributed signal processing tasks that could be represented as a single application of a union of graph multiplier operators. In this section, we present a distributed wavelet denoising example that requires repeated applications of unions of graph multiplier operators and their adjoints. Recall that the distributed Tikhonov regularization method from Section III-A is an efficient way to denoise a signal when we have *a priori* information that the underlying signal is globally smooth. The distributed wavelet denoising method is better suited to situations where we start with a prior belief that the signal is not globally smooth, but rather piecewise smooth, which corresponds to the signal being sparse in the spectral graph wavelet domain [25].

The spectral graph wavelet transform, defined in [25] is precisely of the form of $\mathbf{\Phi}$ in (3). Namely, it is composed of one multiplier, $h(\cdot)$, that acts as a lowpass filter to stably represent the signal's low frequency content, and J wavelet operators, defined by $g_j(\lambda_\ell) = g(t_j\lambda_\ell)$, where $\{t_j\}_{j=1,2,\ldots,J}$ is a set of scales and $g(\cdot)$ is the wavelet multiplier that acts as a bandpass filter.

The most common way to incorporate a sparse prior in a centralized setting is to regularize via a weighted version of the *least absolute shrinkage and selection operator (lasso)* [68], also called *basis pursuit denoising* [69]:

$$\underset{\mathbf{a}}{\operatorname{argmin}}\ \frac{1}{2}\|\mathbf{y}-\mathbf{\Phi}^*\mathbf{a}\|_2^2 + \|\mathbf{a}\|_{1,\boldsymbol{\mu}}\ , \tag{34}$$

where $\|\mathbf{a}\|_{1,\boldsymbol{\mu}} := \sum_{i=1}^{N(J+1)} \mu_i\,|a_i|$ and $\mu_i > 0$ for all i. The optimization problem in (34) can be solved for example by iterative soft thresholding [70]. The initial estimate of the wavelet coefficients $\mathbf{a}^{(0)}$ is arbitrary, and at each iteration of the soft thresholding algorithm, the update of the estimated wavelet coefficients is given by

$$\mathbf{a}^{(\beta)}(i) = \mathcal{S}_{\mu_i\gamma}\left(\left(\mathbf{a}^{(\beta-1)} + \gamma\mathbf{\Phi}\left[\mathbf{y}-\mathbf{\Phi}^*\mathbf{a}^{(\beta-1)}\right]\right)(i)\right),$$
$$i = 1,2,\ldots,N(J+1);\ \ \beta = 1,2,\ldots \tag{35}$$

where γ is the step size and $\mathcal{S}_{\mu_i\gamma}$ is the shrinkage or soft thresholding operator

$$\mathcal{S}_{\mu_i\gamma}(z) := \begin{cases} 0 & ,\ \text{if } \mid z \mid \leq \mu_i\gamma \\ z - \operatorname{sgn}(z)\mu_i\gamma & ,\ \text{o.w.} \end{cases}.$$

The iterative soft thresholding algorithm converges to $\mathbf{a}_*$, the minimizer of (34), if $\gamma < \frac{2}{\|\mathbf{\Phi}^*\|^2}$ [71]. The final denoised estimate of the signal is then given by $\mathbf{\Phi}^*\mathbf{a}_*$.

We now turn to the issue of how to implement the above algorithm in a distributed fashion by sending messages between neighbors in the network. One option would be to use the distributed lasso algorithm of [31], [32], which is a special case of the alternating direction method of multipliers [72, p. 253]. In every iteration of that algorithm, each node transmits its current estimate of *all* the wavelet coefficients to its local neighbors. With the spectral graph wavelet transform, that method requires $2|\mathcal{E}|$ total messages at every iteration, with each message being a vector of length $N(J+1)$. A method where the amount of communicated information does not grow with N (beyond the number of edges, $|\mathcal{E}|$) would be highly preferable.

Algorithm 3 Distributed lasso

Inputs at node n: y_n, $\mathcal{L}_{n,m}\ \forall m$, $\left\{\mu_{(j-1)N+n}\right\}_{j=1,2,\ldots,J+1}$, $\overline{\lambda_{\max}}$, γ, and $\{c_{k,j}\}_{j=1,2,\ldots,J+1;\ k=0,1,\ldots,K}$
Outputs at node n: $\mathbf{y}_*(n)$, the denoised estimate of f_n^0

```
1: Arbitrarily initialize $\left\{\tilde{\mathbf{a}}^{(0)}((j-1)N+n)\right\}_{j=1,2,\ldots,J+1}$
2: Set $\beta = 1$
3: Compute and store $\{(\mathbf{\Phi}\mathbf{y})\,((j-1)N+n)\}_{j=1,2,\ldots,J+1}$
   via Algorithm 1
4: while stopping criterion not satisfied do
5:    Compute and store
          $\left\{(\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}^{(\beta-1)})((j-1)N+n)\right\}_{j=1,2,\ldots,J+1}$
      via Algorithm 2, followed by Algorithm 1
6:    for $j = 1,2,\ldots,J+1$ do
7:       Compute and store
            $\tilde{\mathbf{a}}^{(\beta)}((j-1)N+n)$
            $= \mathcal{S}_{\left(\mu_{(j-1)N+n}\right)\gamma}\begin{pmatrix} \tilde{\mathbf{a}}^{(\beta-1)}((j-1)N+n) \\ +\gamma\left(\tilde{\mathbf{\Phi}}\mathbf{y}\right)((j-1)N+n) \\ -\gamma\left(\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}^{(\beta-1)}\right)((j-1)N+n) \end{pmatrix}$
8:    end for
9:    Set $\beta = \beta + 1$
10: end while
11: for $j = 1,2,\ldots,J+1$ do
12:    Set $(\tilde{\mathbf{a}}_*)\,((j-1)N+n) = \left(\tilde{\mathbf{a}}^{(\beta)}\right)((j-1)N+n)$
13: end for
14: Compute and store $\mathbf{y}_*(n) = (\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}_*)(n)$ via Algorithm 2
15: Output $\mathbf{y}_*(n)$
```

The Chebyshev polynomial approximation of the spectral graph wavelet transform allows us to accomplish this goal. Our approach, which is summarized in Algorithm 3, is to approximate $\mathbf{\Phi}$ by $\tilde{\mathbf{\Phi}}$, and use the distributed implementation of the approximate wavelet transform and its adjoint to perform iterative soft thresholding in order to solve

$$\underset{\tilde{\mathbf{a}}}{\operatorname{argmin}}\ \frac{1}{2}\|\mathbf{y}-\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}\|_2^2 + \|\tilde{\mathbf{a}}\|_{1,\boldsymbol{\mu}}. \tag{36}$$

In the first soft thresholding iteration, each node n must learn $(\tilde{\mathbf{\Phi}}\mathbf{y})((j-1)N+n)$ at all scales j, via Algorithm 1. These coefficients are then stored for future iterations. In the β^{th} iteration, each node n must learn the $J+1$ coefficients of $\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}^{(\beta-1)}$ centered at n, by sequentially applying the operators $\tilde{\Phi}^*$ and $\tilde{\Phi}$ in a distributed manner via Algorithms 2 and 1, respectively. When a stopping criterion for the soft thresholding is satisfied, the adjoint operator $\tilde{\Phi}^*$ is applied again in a distributed manner to the resulting coefficients $\tilde{\mathbf{a}}_*$, and node n's denoised estimate of its signal is $(\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}_*)(n)$. The stopping criterion may simply be a fixed number of iterations, or it may be when $\left|(\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}^{(\beta)})(n) - (\tilde{\mathbf{\Phi}}^*\tilde{\mathbf{a}}^{(\beta-1)})(n)\right| < \epsilon$, for all n and some small ϵ. Finally, note that we could also optimize the weights $\boldsymbol{\mu}$ by performing distributed cross-validation, as discussed in [31], [32].

We now examine the communication requirements of this approach. Recall from Section IV-B that $2K|\mathcal{E}|$ messages of length 1 are required to compute $\tilde{\boldsymbol{\Phi}}\mathbf{y}$ in a distributed fashion. Distributed computation of $\tilde{\boldsymbol{\Phi}}\tilde{\boldsymbol{\Phi}}^*\tilde{\mathbf{a}}^{(\beta-1)}$, the other term needed in the iterative thresholding update (35), requires $2K|\mathcal{E}|$ messages of length $J+1$ and $2K|\mathcal{E}|$ messages of length 1. The final application of the adjoint operator $\tilde{\Phi}^*$ to recover the denoised signal estimates requires another $2K|\mathcal{E}|$ messages, each a vector of length $J+1$. Therefore, the Chebyshev polynomial approximation to the spectral graph wavelet transform enables us to iteratively solve the weighted lasso in a distributed manner where the communication workload only scales with the size of the network through $|\mathcal{E}|$, and is otherwise independent of the network dimension N.

The reconstructed signal in Algorithm 3 is $\tilde{\boldsymbol{\Phi}}^*\tilde{\mathbf{a}}_*$, where $\tilde{\mathbf{a}}_*$ is the solution to the lasso problem (36). A natural question is how good of an approximation $\tilde{\boldsymbol{\Phi}}^*\tilde{\mathbf{a}}_*$ is to $\boldsymbol{\Phi}^*\mathbf{a}_*$, where $\mathbf{a}_*$ is the solution to the original lasso problem (34). The following proposition bounds the squared distance between these two quantities by a term proportional to the spectral norm of the difference between the exact and approximate spectral graph wavelet operators.

Proposition *6:* $\|\tilde{\boldsymbol{\Phi}}^*\tilde{\mathbf{a}}_* - \boldsymbol{\Phi}^*\mathbf{a}_*\|_2^2 \leq C|\!|\!|\tilde{\boldsymbol{\Phi}} - \boldsymbol{\Phi}|\!|\!|_2$, where $|\!|\!|\cdot|\!|\!|_2$ is the spectral norm, and the constant $C = \frac{\|\mathbf{y}\|_2^3}{\min_i \mu_i}$.

Combining Proposition 6, whose proof is included in the Appendix of [44], with (21), we have

$$\|\tilde{\boldsymbol{\Phi}}^*\tilde{\mathbf{a}}_* - \boldsymbol{\Phi}^*\mathbf{a}_*\|_2^2 \leq \frac{\|\mathbf{y}\|_2^3}{\min_i \mu_i} B(K)\sqrt{J+1}. \tag{37}$$

Thus, as we increase the approximation order K, $B(K)$ and the right-hand side of (37) tend toward zero (at a speed dependent on the smoothness of the graph wavelet multipliers $g(\cdot)$ and $h(\cdot)$).

Finally, to illustrate the distributed lasso, we consider a numerical example. We use the same 500 node sensor network as in Section IV-D. This time, however, the underlying signal is piecewise smooth, but not globally smooth, with the n^{th} component given by

$$f_n^0 = \begin{cases} -2n_x + 0.5, & \text{if } n_y \geq 1 - n_x \\ n_x^2 + n_y^2 + 0.5, & \text{if } n_y < 1 - n_x \end{cases}.$$

We corrupt each component of the signal $\mathbf{f}^0$ with uncorrelated additive Gaussian noise with mean zero and standard deviation 0.5. We then solve problem (36) in a distributed manner using Algorithm 3. We use a spectral graph wavelet transform with 6 wavelet scales, implemented by the Graph Signal Processing Toolbox [51]. In Algorithm 3, we run 300 soft thresholding iterations and take $\gamma = 0.2$, $\mu_i = 0.75$ for all the wavelet coefficients, and $\mu_i = 0.01$ for all the scaling coefficients.[7] We do not perform any distributed cross-validation to optimize the weights $\boldsymbol{\mu}$. We repeated this entire experiment 1000 times, with a new random graph and random noise each time.[8] The average mean square errors were 0.250 for the noisy signals, 0.098 for the estimates produced by the Tikhonov regularization method (6), 0.088 for the denoised estimates produced by the distributed lasso with the exact wavelet operator, and 0.080 for the denoised estimates produced by the distributed lasso with the approximate wavelet operator with $K = 15$. Note that the approximate solution does not necessarily result in a higher mean square error than the exact solution.

[7]The scaling coefficients in the spectral graph wavelet transform are not expected to be sparse.

[8]The reported errors are averaged over the 441 random graph realizations that were connected.

VII. Concluding Remarks

We presented a novel method to distribute a class of linear operators called unions of graph multiplier operators. The main idea is to approximate the graph multipliers by Chebyshev polynomials, whose recurrence relations make them readily amenable to distributed computation. Key takeaways from the discussion and application examples include:

- A number of distributed signal processing tasks can be represented as distributed applications of unions of graph multiplier operators (and their adjoints) to signals on weighted graphs. Examples include distributed smoothing, denoising, inverse filtering, and semi-supervised learning.
- Graph Fourier multiplier operators are the graph analog of filter banks, as they reshape functions' frequencies through multiplication in the Fourier domain.
- The amount of communication required to perform the distributed computations only scales with the size of the network through the number of edges of the communication graph, which is usually sparse. Therefore, the method is well suited to large-scale networks.
- The approximate graph multiplier operators closely approximate the exact operators in practice, and for graph multiplier operators with smooth multipliers, an upper bound on the spectral norm of the difference of the approximate and exact operators decreases rapidly as we increase the Chebyshev approximation order.

While the focus in this paper has been on distributed signal processing tasks, the methods we presented can also be used to compute matrix function-vector products of the form $g(\mathbf{P})\mathbf{f}$ in a distributed fashion for other applications. Such computations are used, e.g., to solve ordinary and partial differential equations [73]-[75], approximate spectral densities of large matrices [76], estimate the numerical rank of large matrices [52], [77], and approximate spectral sums such as the log-determinant of a large matrix or the trace of a matrix inverse for applications in physics, biology, information theory, and other disciplines [78].

References

[1] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, Berkeley, CA, Apr. 2004, pp. 20–27.

[2] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, pp. 56–69, Jul. 2006.

[3] R. Olfati-Saber, J. Fax, and R. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.

[4] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. IEEE*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.

[5] A. Sandryhaila, S. Kar, and J. M. F. Moura, "Finite-time distributed consensus through graph filters," in *Proc. IEEE Int. Conf. Acc., Speech, and Signal Process.*, Florence, Italy, May 2014, pp. 1080–1084.

[6] 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, May 2013.

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

[8] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Proc. Ann. Conf. Comp. Learn. Theory*, ser. Lect. Notes Comp. Sci., B. Schölkopf and M. Warmuth, Eds. Springer, 2003, pp. 144–158.

[9] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data." in *Proc. ICML Workshop Stat. Relat. Learn. and Its Connections to Other Fields*, Jul. 2004, pp. 132–137.

[10] ——, "Regularization on discrete spaces," in *Pattern Recogn.*, ser. Lect. Notes Comp. Sci., W. G. Kropatsch, R. Sablatnig, and A. Hanbury, Eds. Springer, 2005, vol. 3663, pp. 361–368.

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

[12] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Carnegie Mellon University, Technical Report CMU-CALD-02-107, 2002.

[13] ——, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proc. Int. Conf. Mach. Learn.*, Washington, D.C., Aug. 2003, pp. 912–919.

[14] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *Learn. Theory*, ser. Lect. Notes Comp. Sci. Springer-Verlag, 2004, pp. 624–638.

[15] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Adv. Neural Inf. Process. Syst.*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16. MIT Press, 2004, pp. 321–328.

[16] O. Delalleau, Y. Bengio, and N. Le Roux, "Efficient non-parametric function induction in semi-supervised learning," in *Proc. Int. Wkshp. on Artif. Intell. Stat.*, Barbados, Jan. 2005, pp. 96–103.

[17] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. MIT Press, 2006.

[18] R. K. Ando and T. Zhang, "Learning on graph with Laplacian regularization," in *Adv. Neural Inf. Process. Syst.*, B. Schölkopf, J. Platt, and T. Hofmann, Eds., vol. 19. MIT Press, 2007, pp. 25–32.

[19] R. Johnson and T. Zhang, "On the effectiveness of Laplacian normalization for graph semi-supervised learning," *J. Mach. Learn. Res.*, vol. 8, pp. 1489–1517, 2007.

[20] S. Bougleux, A. Elmoataz, and M. Melkemi, "Discrete regularization on weighted graphs for image and mesh filtering," in *Scale Space Var. Methods Comp. Vision*, ser. Lect. Notes Comp. Sci., F. Sgallari, A. Murli, and N. Paragios, Eds. Springer, 2007, vol. 4485, pp. 128–139.

[21] 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, pp. 1047–1060, Jul. 2008.

[22] F. Zhang and E. R. Hancock, "Graph spectral image smoothing using the heat kernel," *Pattern Recogn.*, vol. 41, pp. 3328–3342, Nov. 2008.

[23] G. Peyré, S. Bougleux, and L. Cohen, "Non-local regularization of inverse problems," in *Proc. ECCV'08*, ser. Lect. Notes Comp. Sci., D. A. Forsyth, P. H. S. Torr, and A. Zisserman, Eds. Springer, 2008, pp. 57–68.

[24] L. Rosasco, E. De Vito, and A. Verri, "Spectral methods for regularization in learning theory," DISI, Università degli Studi di Genova, Italy, Technical Report DISI-TR-05-18, 2005.

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

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

[27] R. Wagner, V. Delouille, and R. Baraniuk, "Distributed wavelet de-noising for sensor networks," in *Proc. IEEE Int. Conf. Dec. and Contr.*, San Diego, CA, Dec. 2006, pp. 373–379.

[28] S. Barbarossa, G. Scutari, and T. Battisti, "Distributed signal subspace projection algorithms with maximum convergence rate for sensor networks with topological constraints," in *Proc. IEEE Int. Conf. Acc., Speech, and Signal Process.*, Taipei, Apr. 2009, pp. 2893–2896.

[29] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *Proc. Int. Symp. Inf. Process. Sensor Netw.*, Berkeley, CA, Apr. 2004, pp. 1–10.

[30] J. B. Predd, S. R. Kulkarni, and H. V. Poor, "A collaborative training algorithm for distributed learning," *IEEE Trans. Inf. Theory*, vol. 55, no. 4, pp. 1856–1871, Apr. 2009.

[31] J. A. Bazerque, G. Mateos, and G. B. Giannakis, "Distributed lasso for in-network linear regression," in *Proc. IEEE Int. Conf. Acc., Speech, and Signal Process.*, Dallas, TX, Mar. 2010, pp. 2978–2981.

[32] G. Mateos, J.-A. Bazerque, and G. B. Giannakis, "Distributed sparse linear regression," *IEEE Trans. Signal Process.*, vol. 58, no. 10, pp. 5262–5276, Oct. 2010.

[33] D. Thanou and P. Frossard, "Distributed signal processing with graph spectral dictionaries," in *Proc. Allerton Conf. Comm., Contr., and Comp.*, Sep. 2015, pp. 1391–1398.

[34] S. Segarra, A. G. Marques, and A. Ribeiro, "Distributed implementation of linear network operators using graph filters," in *Proc. Allerton Conf. Comm., Contr., and Comp.*, Sep. 2015, pp. 1406–1413.

[35] ——, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[36] 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, Aug. 2015.

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

[38] D. I Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *Proc. Int. Conf. Distr. Comput. in Sensor Syst.*, Barcelona, Spain, June 2011.

[39] N. J. Higham, *Functions of Matrices*. Society for Industrial and Applied Mathematics, 2008.

[40] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 1990.

[41] F. K. Chung, *Spectral Graph Theory*. Vol. 92 of the CBMS Regional Conference Series in Mathematics, AMS Bokstore, 1997.

[42] D. Spielman, "Spectral graph theory," in *Combinatorial Scientific Computing*. Chapman and Hall / CRC Press, 2012.

[43] T. Bıyıkoğlu, J. Leydold, and P. F. Stadler, *Laplacian Eigenvectors of Graphs*. Lecture Notes in Mathematics, vol. 1915, Springer, 2007.

[44] D. I Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via Chebyshev polynomial approximation," *arXiv ePrints*, 2017. [Online]. Available: http://arxiv.org/abs/1111.5239

[45] G. Peyré, *Advanced Signal, Image and Surface Processing*, 2010, http://www.ceremade.dauphine.fr/∼peyre/numerical-tour/book/.

[46] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*. Chapman and Hall, 2003.

[47] G. M. Phillips, *Interpolation and Approximation by Polynomials*. CMS Books in Mathematics, Springer-Verlag, 2003.

[48] T. J. Rivlin, *Chebyshev Polynomials*. Wiley-Interscience, 1990.

[49] L. N. Trefethen, *Approximation Theory and Approximation Practice*. SIAM, 2013.

[50] T. A. Driscoll, N. Hale, and L. N. Trefethen, Eds., *Chebfun Guide*. Oxford: Pafnuty Publications, 2014.

[51] N. Perraudin, J. Paratte, D. I Shuman, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, "GSPBOX: A toolbox for signal processing on graphs," *arXiv ePrints*, 2014, https://lts2.epfl.ch/gsp/. [Online]. Available: http://arxiv.org/abs/1408.5781

[52] S. Ubaru and Y. Saad, "Fast methods for estimating the numerical rank of large matrices," in *Proc. Int. Conf. Mach. Learn.*, New York, NY, Jun. 2016, pp. 468–477.

[53] W. N. Anderson and T. D. Morley, "Eigenvalues of the Laplacian of a graph," *Linear Multilinear Algebra*, vol. 18, no. 2, pp. 141–145, 1985.

[54] K. C. Das and R. P. Bapat, "A sharp upper bound on the largest Laplacian eigenvalue of weighted graphs," *Lin. Alg. Appl.*, vol. 409, pp. 153–165, Nov. 2005.

[55] Y. Zhou and R. C. Li, "Bounding the spectrum of large Hermitian matrices." *Linear Algebra Appl.*, vol. 435, no. 3, pp. 480–493, 2011.

[56] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, Aug. 2004.

[57] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281–323, May 2005.

[58] Y.-C. Wu, Q. Chaudhari, and E. Serpedin, “Clock synchronization of wireless sensor networks,” *IEEE Signal Process. Mag.*, vol. 28, no. 1, pp. 124–138, Jan. 2011.
[59] A. Frommer and V. Simoncini, “Matrix functions,” in *Model Order Reduction: Theory, Research Aspects and Applications*. Springer, 2008, pp. 275–303.
[60] A. Šušnjara, N. Perraudin, D. Kressner, and P. Vandergheynst, “Accelerated filtering on graphs using Lanczos method,” *arXiv ePrints*, 2015. [Online]. Available: https://arxiv.org/abs/1509.04537
[61] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.
[62] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.
[63] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 2013.
[64] B. N. Sheehan, Y. Saad, and R. B. Sidje, “Computing exp(-τa)b with Laguerre polynomials,” *Electron. Trans. Numer. Anal.*, vol. 37, pp. 147–165, 2010.
[65] J. Chen, M. Anitescu, and Y. Saad, “Computing $f(A)b$ via least squares polynomial approximations,” *SIAM J. Sci. Comp.*, vol. 33, no. 1, pp. 195–222, Feb. 2011.
[66] Y. Saad, “Filtered conjugate residual-type algorithms with applications,” *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 845–870, 2006.
[67] W. Gautschi, “Questions of numerical condition related to polynomials,” *Studies in Numerical Analysis*, vol. 24, pp. 140–177, 1984.
[68] R. Tibshirani, “Regression shrinkage and selection via the Lasso,” *J. Royal. Statist. Soc. B*, vol. 58, no. 1, pp. 267–288, 1996.
[69] S. Chen, D. Donoho, and M. Saunders, “Atomic decomposition by basis pursuit,” *SIAM J. Sci. Comp.*, vol. 20, no. 1, pp. 33–61, Aug. 1998.
[70] I. Daubechies, M. Defrise, and C. De Mol, “An iterative thresholding algorithm for linear inverse problems with a sparsity constraint,” *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Nov. 2004.
[71] P. L. Combettes and V. R. Wajs, “Signal recovery by proximal forward-backward splitting,” *Multiscale Model. Sim.*, vol. 4, no. 4, pp. 1168–1200, Nov. 2005.
[72] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.
[73] M. Hochbruck, C. Lubich, and H. Selhofer, “Exponential integrators for large systems of differential equations,” *SIAM J. Sci. Comput.*, vol. 19, no. 5, pp. 1552–1574, 1998.
[74] R. A. Friesner, L. Tuckerman, B. Dornblaser, and T. V. Russo, “A method for exponential propagation of large systems of stiff nonlinear differential equations,” *J. Sci. Comput.*, vol. 4, no. 4, pp. 327–354, Dec. 1989.
[75] E. Gallopoulos and Y. Saad, “Efficient solution of parabolic equations by Krylov approximation methods,” *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 5, pp. 1236–1264, 1992.
[76] L. Lin, Y. Saad, and C. Yang, “Approximating spectral densities of large matrices,” *SIAM Review*, vol. 58, no. 1, pp. 34–65, 2016.
[77] S. Ubaru, Y. Saad, and A.-K. Seghouane, “Fast estimation of approximate matrix ranks using spectral densities,” *Neural Computation*, vol. 29, no. 5, pp. 1317–1351, May 2017.
[78] I. Han, D. Malioutov, H. Avron, and J. Shin, “Approximating spectral sums of large-scale matrices using stochastic Chebyshev approximations,” *SIAM J. Sci. Comput.*, vol. 39, no. 4, pp. A1558–A1585, 2017.

David I Shuman received the B.A. degree in economics and the M.S. degree in engineering-economic systems and operations research from Stanford University, Stanford, CA, in 2001 and the M.S. degree in electrical engineering: systems, the M.S. degree in applied mathematics, and the Ph.D. degree in electrical engineering: systems from the University of Michigan, Ann Arbor, in 2006, 2009, and 2010, respectively.

He is currently an Assistant Professor in the Department of Mathematics, Statistics, and Computer Science, Macalester College, St. Paul, Minnesota, which he joined in January 2014. From 2010 to 2013, he was a Postdoctoral Researcher at the Institute of Electrical Engineering, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. His research interests include signal processing on graphs, computational harmonic analysis, and stochastic scheduling and resource allocation problems.

Dr. Shuman is an Associate Editor for the IEEE Signal Processing Letters (2017-), and has served on the Technical Program Committee for the IEEE Global Conference on Signal and Information Processing (2015-2017). He received the 2016 IEEE Signal Processing Magazine Best Paper Award, and was a 2014-2015 Project NExT Fellow of the Mathematical Association of America.

Pierre Vandergheynst Pierre Vandergheynst is Professor of Electrical Engineering at the Ecole Polytechnique Fédérale de Lausanne (EPFL) and Director of the Signal Processing Laboratory (LTS2). A theoretical physicist by training, Pierre is a renown expert in the mathematical modeling of complex data. His current research focuses on data processing with graph-based methods with a particular emphasis on machine learning and network science.

Pierre Vandergheynst has served as associate editor of multiple flagship journals, such as the IEEE Transactions on Signal Processing and SIAM Imaging Sciences. He is the author or co-author of more than 100 published technical papers and has received several best paper awards from technical societies. He was awarded the Apple ARTS award in 2007 and the De Boelpaepe prize of the Royal Academy of Sciences of Belgium in 2010.

As of January 1st 2017, Prof. Vandergheynst is EPFL’s Vice-President for Education.

Daniel Kressner received the Diploma and the Ph.D. degrees in mathematics from TU Chemnitz and TU Berlin, Germany, in 2001 and 2004, respectively. After stays at the Universities of Umea, Sweden, and Zagreb, Croatia, as a DFG Emmy Noether PostDoctoral fellow, he was an assistant professor at ETH Zurich (2007-2010). Since 2011, he is at EPFL, Switzerland, where he is currently full professor. Most of his research is in numerical linear algebra, with a focus on low-rank matrix and tensor approximation techniques. He received the John Todd Award from the Oberwolfach Foundation in 2011 and a SIAM outstanding paper prize in 2013.

Pascal Frossard (S96,M01,SM04,F18) received the M.S. and Ph.D. degrees, both in electrical engineering, from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 1997 and 2000, respectively. Between 2001 and 2003, he was a member of the research staff at the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he worked on media coding and streaming technologies. Since 2003, he has been a faculty at EPFL, where he heads the Signal Processing Laboratory (LTS4). His research interests include graph signal processing, image representation and coding, visual information analysis, and distributed signal processing and communications.

Dr. Frossard has been the General Chair of IEEE ICME 2002 and Packet Video 2007. He has been the Technical Program Chair of IEEE ICIP 2014 and EUSIPCO 2008, and a member of the organizing or technical program committees of numerous conferences. He has been a senior Area Editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING (2015-), an Associate Editor of the IEEE TRANSACTIONS ON BIG DATA (2015-), IEEE TRANSACTIONS ON IMAGE PROCESSING (2010-2013), the IEEE TRANSACTIONS ON MULTIMEDIA (2004-2012), and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (2006-2011). He is an elected member of the IEEE Multimedia Signal Processing Technical Committee (2004-2007, 2016-), the IEEE Visual Signal Processing and Communications Technical Committee (2006-) and the IEEE Multimedia Systems and Applications Technical Committee (2005-). He has served as Chair of the IEEE Image, Video and Multidimensional Signal Processing Technical Committee (2014-2015), and Steering Committee Chair (2012-2014) and Vice-Chair (2004-2006) of the IEEE Multimedia Communications Technical Committee. He received the Swiss NSF Professorship Award in 2003, the IBM Faculty Award in 2005, the IBM Exploratory Stream Analytics Innovation Award in 2008, the Google Faculty Award 2017, the IEEE Transactions on Multimedia Best Paper Award in 2011, and the IEEE Signal Processing Magazine Best Paper Award 2016.