2020 54th Asilomar Conference on Signals, Systems, and Computers | 978-0-7381-3126-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/IEEECONF51394.2020.9443349

# Node-Asynchronous Implementation of Filter Banks on Graphs

Oguzhan Teke and P. P. Vaidyanathan
Department of Electrical Engineering
California Institute of Technology
oteke@caltech.edu, ppvnath@systems.caltech.edu

*Abstract*—**Filter banks on graphs are shown to be useful for analyzing data defined over networks, as they decompose a graph signal into components with low variation and high variation. Based on recent node-asynchronous implementation of graph filters, this study proposes an asynchronous implementation of filter banks on graphs. In the proposed algorithm nodes follow a randomized collect-compute-broadcast scheme: if a node is in the passive stage it collects the data sent by its incoming neighbors and stores only the most recent data. When a node gets into the active stage at a random time instance, it does the necessary filtering computations locally, and broadcasts a state vector to its outgoing neighbors. When the underlying filters (of the filter bank) are rational functions with the same denominator, the proposed filter bank implementation does not require additional communication between the neighboring nodes. However, computations done by a node increase linearly with the number of filters in the bank. It is also proven that the proposed asynchronous implementation converges to the desired output of the filter bank in the mean-squared sense under mild stability conditions. The convergence is verified also with numerical experiments.**

## I. Introduction

One of the key aspects in the field of graph signal processing is the use of graph filters, which provides a versatile tool that can be utilized in order to smooth out graph signals (low-pass filters), or detect anomalies (high-pass filters) [1]–[3]. More importantly, graph filters are implicitly related to distributed signal processing due to their localized structures on graphs. Due to their importance, design and implementation of graph filters have been of interest in recent years. The papers [4]–[12] (and references therein) made explicit connections between polynomial graph filters and distributed computation, and studied various problems including smoothing, regularization, and consensus.

Although graph filters (either polynomial, or rational) can be implemented in a distributed fashion requiring only localized communication between the neighboring nodes, most of the proposed implementations rely on consecutive and *synchronous* communication between the neighboring nodes of the graph. In the case of large scale distributed graph processing frameworks, e.g., [13]–[16], synchronization becomes an important limitation, as it can cause delays in the system.

In order to eliminate the need for synchronization, the study in [17] recently proposed a node-asynchronous implementation of a given graph filter. In the proposed algorithm, neighboring nodes communicate with each other at random time instances asynchronously from each other. So, no synchronization is required when implementing the distributed filtering operations. Furthermore, the implementation was proven to converge under mild conditions on the graph, filter and the random behavior of the nodes.

This work was supported in parts by the ONR grant N00014-18-1-2390, the NSF grant CCF-1712633, and the Electrical Engineering Carver Mead Research Seed Fund of the California Institute of Technology.

We note that [17] considered the node-asynchronous implementation of a *single* filter. However, in practice, it can be more desirable to implement several filters in parallel similar to a filter bank structure [18]. In this regard, here, we will extend the results of [17] to the case of multiple parallel filters. In doing so, we will preserve the node-asynchronous behavior of the implementation. Furthermore, we will keep the *communication cost* between the neighboring nodes the same. Only the *local computation cost* will increase linearly with the number of filters implemented in parallel. Based on the analysis in [17], we will show that the node-asynchronous implementation preserves its convergence behavior even when implementing several filters in parallel.

The rest of the paper is organized as follows: Section II presents graph filter banks, and Section III presents a node-asynchronous implementation of such graph filter banks. Section IV proves the convergence of the proposed implementation (Lemma 1). Section V provides a numerical example that visualizes the convergence behavior of the randomized asynchronous recursions.

### A. Preliminaries and Notation

We will use $\mathbb{E}[\cdot]$ to denote the expectation. For a matrix $\mathbf{X}$ we will use $\mathbf{X}^*$ and $\mathbf{X}^{\rm H}$ to denote its element-wise conjugate and conjugate transpose, respectively. We will use $\otimes$ to denote the Kronecker product. We will use $\mathbf{I}_M$ to denote the identity matrix of size $M$, and $\mathbf{e}_i$ to denote the $i^{th}$ standard vector that has 1 at the $i^{th}$ index and 0 elsewhere.

When discussing graphs, we will use $\mathbf{G} \in \mathbb{C}^{N\times N}$ to denote a graph operator for the graph with $N$ nodes. Here $G_{i,j}$ denotes the weight of the edge from node $j$ to node $i$. In particular, $G_{i,j} = 0$ when nodes $i$ and $j$ are not neighbors. Examples of such local graph operators include the adjacency matrix, the graph Laplacian, etc. *The graph is allowed to be directed possibly with a non-diagonalizable adjacency matrix.* We will use $\mathcal{N}_{\rm in}(i)$ and $\mathcal{N}_{\rm out}(i)$ to denote the incoming and outgoing neighbors of the node $i$. More precisely we have:

$$\mathcal{N}_{\rm in}(i) = \{j \mid G_{i,j} \neq 0\}, \qquad \mathcal{N}_{\rm out}(i) = \{j \mid G_{j,i} \neq 0\}. \quad (1)$$

## II. Rational Graph Filter Banks

In this section, we will overview the notion of graph filters, which play a central role in graph signal processing [2]. More specifically, we will describe the setup, which will be useful later in Section III when describing the node-asynchronous implementation. In this regard, we start by assuming that we are given $M$ filters in the following form:

$$h_0(x) = \frac{p_0(x)}{q(x)}, \qquad \cdots, \qquad h_{M\text{-}1}(x) = \frac{p_{M\text{-}1}(x)}{q(x)}, \tag{2}$$

where it is assumed that filters share the same denominator polynomial $q(x)$, but numerators are assumed to be arbitrary polynomials of order $L$. More precisely, they are assumed to be in the following form:

$$p_i(x) = \sum_{n=0}^{L} p_{i,n}\, x^n, \qquad q(x) = 1 + \sum_{n=1}^{L} q_n\, x^n. \tag{3}$$

The coefficients are allowed to be complex in general, and polynomial (FIR) graph filters, which correspond to the case of $q_1 = \cdots = q_L = 0$, are not excluded.

When implemented as graph filters, having the same denominator polynomial $q(x)$ in (2) brings an advantage in terms of the communication cost between the neighboring nodes of the graph. That is, any number of filters can be implemented in parallel without increasing the communication as long as the filters share the same denominator. We will elaborate on this point later in Section III. We also note that the assumption in (2) is not a restriction in general, as arbitrary rational functions can be equivalently represented with a common denominator.

### A. State-Space Descriptions

Regarding the filters in (2), we will assume that the quadruple $(\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d})$ represents an $L$-dimensional state-space realization. That is, the filters have the following infinite order polynomial representation:

$$\begin{bmatrix} h_0(x) \\ h_1(x) \\ \vdots \\ h_{M\text{-}1}(x) \end{bmatrix} = \mathbf{d} + \sum_{n=1}^{\infty} \mathbf{C}\,\mathbf{A}^{n\text{-}1}\,\mathbf{b}\, x^n, \tag{4}$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d})$ have the following dimensions:

$$\mathbf{A} \in \mathbb{C}^{L\times L}, \quad \mathbf{b} \in \mathbb{C}^{L}, \quad \mathbf{C} \in \mathbb{C}^{M\times L}, \quad \mathbf{d} \in \mathbb{C}^{M}. \tag{5}$$

Given the coefficients of the polynomials of $p_i(x)$ and $q(x)$, the direct form representation of the filters in (2) can be obtained as follows (see [18, Section 13.4]):

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \text{-}q_L & \text{-}q_{L\text{-}1} & \cdots & \cdots & \text{-}q_1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} p_{0,\,0} \\ p_{1,\,0} \\ \vdots \\ p_{M\text{-}1,\,0} \end{bmatrix},$$

$$(\mathbf{C})_{i,j} = p_{i\text{-}1,\,L\text{-}j+1} - p_{i\text{-}1,\,0}\; q_{L\text{-}j+1}, \qquad \begin{matrix} 1 \leqslant i \leqslant M, \\ 1 \leqslant j \leqslant L. \end{matrix} \tag{6}$$

We note that state-space realization of the filters will be useful when considering their node-asynchronous implementations in Section III. We also note that state-space representations are not unique, and different (but equivalent) representations of (2) can be obtained by applying similarity transforms on (6).

### B. Filters on Graphs

In the context of graph signal processing a rational graph filter based on the filters in (2) has the following form:

$$h_i(\mathbf{G}) = p_i(\mathbf{G})\; q(\mathbf{G})^{\text{-}1}, \tag{7}$$

where $\mathbf{G} \in \mathbb{C}^{N\times N}$ denotes the operator of the graph at hand, and it is implicitly assumed that $q(\mathbf{G})$ is an invertible matrix.

For a given graph signal $\mathbf{u} \in \mathbb{C}^N$, the filtered version of the signal with the graph filter $h_i(\mathbf{G})$ is given as follows:

$$\tilde{\mathbf{u}}_i = h_i(\mathbf{G})\,\mathbf{u}. \tag{8}$$

We note that when the graph operator $\mathbf{G}$ is selected as the Laplacian matrix and the polynomial coefficients are selected appropriately, the filtering operation in (8) corresponds to Laplacian smoothing [4], [19], [20]. In this regard, rational graph filters can be considered as extensions of the Laplacian smoothing to arbitrary graph operators.

## III. Node-Asynchronous Graph Filter Banks

Since we consider graphs of finite size $N$, any rational graph filter can be written as a polynomial graph filter of order at most $N$-1. (See [21, Theorem 6.2.9].) Therefore, one straightforward way to implement a rational graph filter is to compute its polynomial representation, and then implement the polynomial filter through consecutive graph shifts. This approach has two limitations: 1) A rational graph filter of order $L$ typically has an order $N$-1 polynomial representation even when $L \ll N$. This limits the practicality especially when the graph is large. 2) As discussed in [22], [23], a graph shift (multiplication with $\mathbf{G}$) requires a synchronization over the network, which introduces delays, or it may not be even possible in the case of autonomous networks.

In order to eliminate these limitations, the study [17] considered a node-asynchronous implementation of a given rational graph filter, in which nodes interact with their neighbors randomly and asynchronously. Here, we will show that it can be extended to implement different filters in parallel.

The implementation proposed in [17] assumes that each node has the following four local variables:

*1) An input signal:* It is assumed that the $i^{th}$ node has an input signal denoted by $u_i \in \mathbb{C}$. Collection of all these input signals, denoted as $\mathbf{u} \in \mathbb{C}^N$, will be referred to as the input graph signal.

*2) A state-vector:* The $i^{th}$ node is assumed to have a local state vector $\mathbf{x}_i \in \mathbb{C}^L$, which will be updated recursively and broadcast to outgoing neighbors $\mathcal{N}_{\text{out}}(i)$ at random time instances. We note that the size of the state vector is determined only by the *order* of the filters $L$, but not by the number of filters $M$.

*3) An output signal:* The $i^{th}$ node will have a local output signal $\mathbf{y}_i \in \mathbb{C}^M$, whose size will be determined by the number of filters $M$.

*4) A buffer:* In order to allow asynchronous behavior, each node is assumed to have a buffer in order to store the state vectors of its incoming neighbors. So, the $i^{th}$ node is assumed to have a buffer of size $L\,|\mathcal{N}_{\text{in}}(i)|$, which will be used to update the state-vector.

Among these four local variables, only the input signal stays the same over time, and the remaining variables get updated randomly. In fact, we will show that the output variable $\mathbf{y}_i$ converges to the $i^{th}$ component of all $M$ output graph signals.

The schematic description of the proposed implementation is presented in Figure 1, in which a node is either in the "active," or "passive" stage. When a node is in the passive stage, it only receives state vectors from its incoming neighbors and updates its buffer accordingly. We note that a node stores only the most recent state vector coming from an incoming neighbor. When a node gets into the active stage at a random time instance independently and asynchronously from the remaining nodes, it updates its state vector and output signal using the input signal and the data already available in its buffer. Once the computations are done, the node broadcasts the new value of its state vector to its outgoing neighbors.

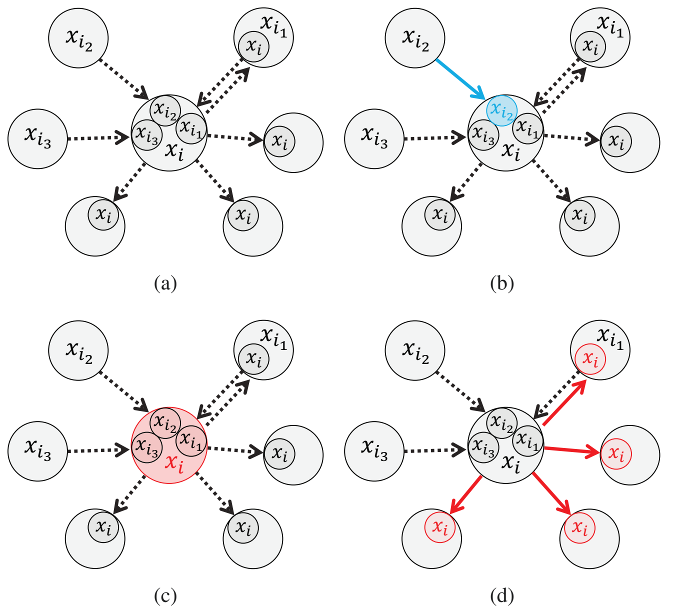


Fig. 1. Visual illustration of the proposed asynchronous implementation of a given graph filter [17]. Edges can be directed in the network. (a) The node $i$ waits and listens in the passive stage. (b) When the node $i$ receives a message, it updates its buffer. (c) When the node $i$ gets into the active stage at a random time instance, it first updates its state vector. (d) After the update, the node $i$ broadcasts its state vector to its outgoing neighbors.

During the active stage, the node updates its state-vector in a two-step procedure. The node first updates its state vector using a local *graph shift* using the data already available in its buffer according to the graph operator $\mathbf{G}$. More precisely, when the $i^{th}$ node is updating its state vector, the node does the following computation first:

$$\mathbf{x}_i' \leftarrow \sum_{j\in\mathcal{N}_{\text{in}}(i)} G_{i,j}\ \mathbf{x}_j, \tag{9}$$

where $\mathbf{x}_i' \in \mathbb{C}^L$ denotes the "graph shifted version" of the state vector $\mathbf{x}_i$. It is important to note that the computation in (9) can be done locally and *asynchronously* by the $i^{th}$ node, as the node is assumed to have all the state vectors of its incoming neighbors already available in its buffer.

After the graph shift step, the state vector is updated one more time using the input signal and the state-space representation of the graph filters. More precisely, the $i^{th}$ node executes the following updates locally in the filtering step:

$$\begin{aligned}\mathbf{y}_i &\leftarrow \mathbf{C}\ \mathbf{x}_i' + \mathbf{d}\ u_i,\\ \mathbf{x}_i &\leftarrow \mathbf{A}\ \mathbf{x}_i' + \mathbf{b}\ u_i,\end{aligned} \tag{10}$$

where $\mathbf{x}_i' \in \mathbb{C}^L$ is as in (9), and the quadruple $(\mathbf{A},\ \mathbf{b},\ \mathbf{C},\ \mathbf{d})$ is a state-space representation of the filters as in (6). We note that the output signal $\mathbf{y}_i$ is also updated in the filtering stage.

When the filtering stage described in (10) is completed, the node broadcasts its most recent state vector $\mathbf{x}_i$ to its outgoing neighbors $\mathcal{N}_{\text{out}}(i)$. We note that (9) and (10) describe the behavior of a single node in the active stage. The graph consists of $N$ nodes, and we note that each node follows these procedures asynchronously from each other at random time instances. The proposed implementation is presented formally in Algorithm 1.

**Algorithm 1** Node-Asynchronous Graph Filter Bank

```
1:  procedure INITIALIZATION($i$)
2:      Initialize the state vector $\mathbf{x}_i \in \mathbb{C}^L$ as $\mathbf{x}_i = \mathbf{0}$.
3:  procedure PASSIVE STAGE($i$)
4:      if $\mathbf{x}_j$ is received from the node $j \in \mathcal{N}_{\text{in}}(i)$ then
5:          Store the most recent value of $\mathbf{x}_j$.
6:  procedure ACTIVE STAGE($i$)
7:      $\mathbf{x}_i' \leftarrow \sum_{j\in\mathcal{N}_{\text{in}}(i)} G_{i,j}\ \mathbf{x}_j$.          ▷ graph shift
8:      $\mathbf{y}_i \leftarrow \mathbf{C}\ \mathbf{x}_i' + \mathbf{d}\ u_i$.          ▷ filtering
9:      $\mathbf{x}_i \leftarrow \mathbf{A}\ \mathbf{x}_i' + \mathbf{b}\ u_i$.          ▷ filtering
10:     Broadcast $\mathbf{x}_i$ to all $j \in \mathcal{N}_{\text{out}}(i)$.
```

In the presented algorithm we emphasize that a node getting into the active stage is independent of the values in its buffer. In general, a node does *not* wait until it receives state vectors from all of its neighbors. In between two activations (i.e., in the passive stage), some values in the buffer may be updated more than once, and some may not be updated at all. Nodes use the most recent update only.

## IV. Convergence of The Algorithm

In this section we will show that Algorithm 1 is indeed a valid implementation of the rational graph filters given in (2). We note that in the case of a single rational filter, $M = 1$, the study [17] proved that the output variables in Algorithm 1 converges to the filtered graph signal in the mean-squared sense under some mild conditions on the graph, the filter matrix and the random behavior of the nodes. Based on the result of [17], this section will show that the same convergence behavior holds true even when implementing several filters in parallel.

In this regard, we start by assuming a stochastic model for the random behavior of the nodes in Algorithm 1. In particular, whenever a node gets updated we will assume that an iteration has passed. We note that more than one node may get updated in a single iteration. Furthermore, we will assume that the $i^{th}$ node gets into the active stage randomly with probability $p_i$ independently at any iteration, and we will use the following diagonal matrix to denote the update probabilities of all nodes:

$$\mathbf{P} = \text{diag}\Big(\begin{bmatrix} p_1 & p_2 & \cdots & p_N\end{bmatrix}\Big) \in \mathbb{R}^{N\times N}. \tag{11}$$

In the case of a single rational filter, $M = 1$, let $\mathbf{y}_{(k)} \in \mathbb{C}^N$ denote the combined output of all the nodes after $k$ iterations of the algorithm. More precisely,

$$\mathbf{y}_{(k)} = \begin{bmatrix} y_1 & y_2 & \cdots & y_N\end{bmatrix}^{\text{T}}, \tag{12}$$

where we note that the index $k$ is a global counter that we use to enumerate the iterations. In general, nodes are unaware of the value of $k$, which is why the variables corresponding to individual nodes are not indexed by $k$.

When the node update probability matrix $\mathbf{P}$, the state transition matrix $\mathbf{A}$ of the filter, and the operator $\mathbf{G}$ of the graph satisfy the following:

$$\|\mathbf{A}\|_2^2 \, \mathbf{G}^{\mathrm{H}} \, \mathbf{P} \, \mathbf{G} \prec \mathbf{P}, \tag{13}$$

it is shown in [17, Theorem 3] that

$$\lim_{k\to\infty} \mathbb{E}\big[\big\|\mathbf{y}_{(k)} - \tilde{\mathbf{u}}\big\|_2^2\big] = 0. \tag{14}$$

That is, the output variables in Algorithm 1 converges to the desired output graph signal in the mean-square sense as the iterations progress.

In the case of multiple filters, $M \geqslant 1$, we will show that the condition (17) is still sufficient to ensure the convergence of the algorithm. In this regard, we first define $\mathbf{Y}_{(k)}$ as follows:

$$\mathbf{Y}_{(k)} = [\mathbf{y}_1 \quad \mathbf{y}_2 \quad \cdots \quad \mathbf{y}_N]^{\mathrm{T}} \in \mathbb{C}^{N\times M}, \tag{15}$$

which denotes the combined output vectors of all nodes of the graph. Furthermore, we will use $\widetilde{\mathbf{U}}$ to denote the filtered graph signals by the filters in (2). More precisely,

$$\widetilde{\mathbf{U}} = \big[\tilde{\mathbf{u}}_0 \quad \tilde{\mathbf{u}}_1 \quad \cdots \quad \tilde{\mathbf{u}}_{M\text{-}1}\big] \in \mathbb{C}^{N\times M}, \tag{16}$$

where $\tilde{\mathbf{u}}_i$'s are as in (8). Then, we present the following:

**Lemma 1.** *In Algorithm 1, let* $\mathbf{P}$ *denote the node update probability matrix. If the state transition matrix* $\mathbf{A}$ *of the filter, and the operator* $\mathbf{G}$ *of the graph satisfy the following:*

$$\|\mathbf{A}\|_2^2 \, \mathbf{G}^{\mathrm{H}} \, \mathbf{P} \, \mathbf{G} \prec \mathbf{P}, \tag{17}$$

*then*

$$\lim_{k\to\infty} \mathbb{E}\big[\big\|\mathbf{Y}_{(k)} - \widetilde{\mathbf{U}}\big\|_F^2\big] = 0. \tag{18}$$

*Proof:* Since it is assumed that the filters in (2) share the same denominator polynomial, the state vector $\mathbf{x}_i$ of each node is updated the same irrespective of the number of filters. On the other hand, entries of the output vector $\mathbf{y}_i$ are updated according to the corresponding numerator polynomials. More precisely, consider the $j^{th}$ entry of the output vector of the $i^{th}$ node in Line 8 of the algorithm:

$$\begin{aligned}(\mathbf{y}_i)_j \leftarrow [p_{j\text{-}1,\,L} - p_{j\text{-}1,\,0}\, q_L \quad \cdots \quad p_{j\text{-}1,\,1} - p_{j\text{-}1,\,0}\, q_1]\; \mathbf{x}_i' \\ + p_{j\text{-}1,\,0}\; u_i,\end{aligned} \tag{19}$$

which follows from the direct form representations of $\mathbf{C}$ and $\mathbf{d}$ given in (6).

Since the $j^{th}$ entry of the output vector is updated according to the $j^{th}$ filter only, we can consider each entry of the output separately. More precisely, let the following be the combined $j^{th}$ entry of all the output vectors:

$$\mathbf{y}_{(k)}^{j} = [(\mathbf{y}_1)_j \quad (\mathbf{y}_2)_j \quad \cdots \quad (\mathbf{y}_N)_j]^{\mathrm{T}} \in \mathbb{C}^{N}. \tag{20}$$

Under the condition in (17), [17, Theorem 3] shows that

$$\lim_{k\to\infty} \mathbb{E}\big[\big\|\mathbf{y}_{(k)}^{j} - \tilde{\mathbf{u}}_{j\text{-}1}\big\|_2^2\big] = 0. \tag{21}$$

We also note that definitions of $\mathbf{Y}_{(k)}$ and $\widetilde{\mathbf{U}}$ imply that

$$\big\|\mathbf{Y}_{(k)} - \widetilde{\mathbf{U}}\big\|_F^2 = \sum_{j=1}^{M} \big\|\mathbf{y}_{(k)}^{j} - \tilde{\mathbf{u}}_{j\text{-}1}\big\|_2^2, \tag{22}$$

which shows that the condition (17) is sufficient to ensure the mean-square convergence in (18). ■

Two remarks are in order regarding Lemma 1:

1) The implementation presented in Algorithm 1 considers the noise-free case. Hence, Lemma 1 shows that the combined output vector *converges* to the desired output of graph filters in the mean-square sense. In the presence of noise, we note that the combined output vector $\mathbf{Y}_{(k)}$ does not converge to the desired output exactly. Instead, the difference between $\mathbf{Y}_{(k)}$ and $\widetilde{\mathbf{U}}$ reaches an error floor, whose value is determined by the input noise statistics, update probabilities, state-transition matrix of the filter, and the graph operator. In fact, [17, Theorem 3] provides an upper bound on the error floor. Similar bounds can also be obtained for the case of multiple filters.

2) We note that condition (17) in Lemma 1 is only *sufficient* to ensure the convergence. So, the algorithm may still converge even when the condition (17) is not satisfied. Additionally, the study [17] also demonstrated that it is possible to construct examples in which Algorithm 1 converges only when it has sufficient amount of asynchronicity. (So, it may diverge in the synchronous mode of operation.) Although Lemma 1 does not explain these phenomenas, the study [24] presented the *necessary and sufficient* condition for the mean-square convergence of the algorithm in the case of a single filter. More precisely, consider the following matrix $\mathbf{S}$ of size $(NL)^2 \times (NL)^2$:

$$\begin{aligned}&\mathbf{S} = \bar{\mathbf{A}}^{*} \otimes \bar{\mathbf{A}} + \\ &\Big(\mathbf{I}_L \otimes (\mathbf{P}^{\text{-}1} - \mathbf{I}_N) \otimes \mathbf{I}_{NL}\Big)\, \mathbf{J}\, \Big((\bar{\mathbf{A}}^{*} - \mathbf{I}_{NL}) \otimes (\bar{\mathbf{A}} - \mathbf{I}_{NL})\Big),\end{aligned} \tag{23}$$

where * denotes element-wise conjugation, and $\mathbf{J}$ is a diagonal matrix as follows:

$$\mathbf{J} = \sum_{i=1}^{N} \mathbf{I}_L \otimes (\mathbf{e}_i\, \mathbf{e}_i^{\mathrm{H}}) \otimes \mathbf{I}_L \otimes (\mathbf{e}_i\, \mathbf{e}_i^{\mathrm{H}}) \in \mathbb{R}^{(NL)^2\times(NL)^2}, \tag{24}$$

and the matrix $\bar{\mathbf{A}}$ is given as follows:

$$\bar{\mathbf{A}} = \mathbf{I}_{NL} + (\mathbf{I}_L \otimes \mathbf{P})\,(\mathbf{A}\otimes\mathbf{G} - \mathbf{I}_{NL}). \tag{25}$$

Then, it is shown in [24] that Algorithm 1 converges in the mean-square sense *if and only if* the spectral radius of matrix $\mathbf{S}$ in (23) is strictly less than unity. Additionally, we can argue that stability of the matrix $\mathbf{S}$ determines the convergence of the algorithm even when implementing several filters in parallel.

## V. Numerical Simulations

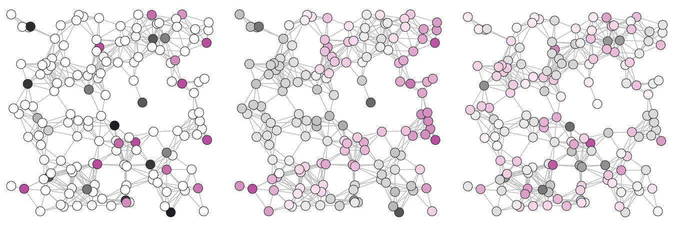
In this section, we will numerically demonstrate the convergence of the algorithm for the case of multiple filters running in parallel on the graph demonstrated in Figure 2, which is an undirected random geometric graph on $N = 150$ nodes. We also note that we will use the graph Laplacian as the graph operator in this section.

In order to demonstrate the convergence behavior of node-asynchronous filtering operations, we consider $M = 2$ polynomial (FIR) graph filters running in parallel. So, we take $q(x) = 1$ in (2), and select the numerators as follows:

$$p_0(x) = -0.0002\,x^3 + 0.0091\,x^2 - 0.1650\,x + 1, \tag{26}$$

$$p_1(x) = 0.0002\,x^3 + 0.0003\,x^2 + 0.0009\,x. \tag{27}$$

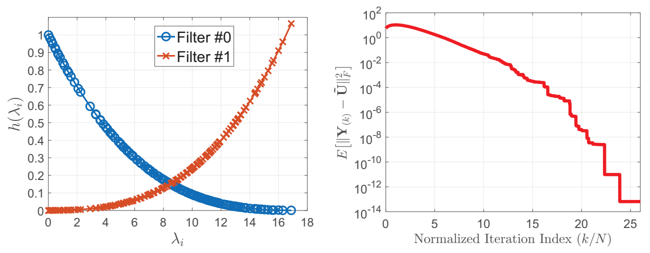
On the graph visualized in Figure 2, the response (with respect to the graph Laplacian) of the filters in (26) and (27) are visualized in Figure 3(a). So, $h_0(x)$ has a low-pass behavior, and $h_1(x)$ has a high-pass behavior. When the



(a) (b) (c)

Fig. 2. Visualization of the signals on the graph. Colors black and pink represent positive and negative values, respectively. Intensity of a color represents the magnitude. (a) The input graph signal $\mathbf{u}$ that has nonzero values on 30 nodes. (b) The filtered signal $\tilde{\mathbf{u}}_0$ with the filter in (26). (c) The filtered signal $\tilde{\mathbf{u}}_1$ with the filter in (27).

signal visualized in Figure 2(a) is used as the input to these filters, the corresponding output graph signals are presented in Figures 2(b) and 2(c), respectively.



(a) (b)

Fig. 3. (a) Response of the FIR filters in (26) and (27). (b) The average error in Algorithm 1 throughout the iterations when implementing the filters in (26) and (27). The average behavior is obtained by empirically averaging over $10^3$ independent realizations.

In Figure 3(b), we demonstrate the convergence behavior of Algorithm 1 throughout the iterations when implementing the filters in (26) and (27). As the iterations progress, the figure clearly shows the convergence of the algorithm, which can also be theoretically verified by checking the stability of the matrix $\mathbf{S}$ in (23). We also note that a visualization of the iterations of the algorithm is provided as a supplementary file in [25].

## VI. Conclusions

In this paper, we extended the node-asynchronous implementation of a single graph filter to the case of multiple graph filters running in parallel. By assuming that all the filters share the same denominator polynomial, we kept the communication cost between the neighboring nodes the same irrespective of the number of filters implemented. Only the local computation cost increases linearly with the number of filters. Following the analysis done for the case of a single filter, we proved that the node-asynchronous implementation converges to the desired output of the filter bank in the mean-squared sense under mild stability conditions involving the graph, the filter, and the update probabilities of the nodes. We also provided numerical examples in order to verify the convergence of the implementation.

Although the filters are assumed to share the same denominator polynomial, it is, in fact, possible to implement arbitrary rational filters within the node-asynchronous framework considered in this paper. This extension is left as a future direction.

## References

[1] A. Sandryhaila and J. M. F. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 80–90, Sept. 2014.
[2] D. Shuman, S. 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.
[3] A. Ortega, P. Frossard, J. Kovacevic, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, May 2018.
[4] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via chebyshev polynomial approximation," *IEEE Trans. on Sig. and Inf. Process. Net.*, vol. 4, no. 4, pp. 736–751, Dec. 2018.
[5] S. Safavi and U. A. Khan, "Revisiting finite-time distributed algorithms via successive nulling of eigenvalues," *IEEE Sig. Process. Letters*, vol. 22, no. 1, pp. 54–57, Jan. 2015.
[6] A. Sandryhaila, S. Kar, and J. M. F. Moura, "Finite-time distributed consensus through graph filters," in *Proc. Int. Conf. Acoust. Speech, Signal Process. (ICASSP)*, May 2014, pp. 1080–1084.
[7] S. Segarra, A. G. Marques, and A. Ribeiro, "Distributed implementation of linear network operators using graph filters," in *Allerton Conference on Communication, Control, and Computing*, Sept. 2015, pp. 1406–1413.
[8] X. Shi, H. Feng, M. Zhai, T. Yang, and B. Hu, "Infinite impulse response graph filters in wireless sensor networks," *IEEE Sig. Process. Letters*, vol. 22, no. 8, pp. 1113–1117, Aug. 2015.
[9] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. on Sig. Process.*, vol. 65, no. 2, pp. 274–288, Jan. 2017.
[10] ——, "Filtering random graph processes over random time-varying graphs," *IEEE Trans. Signal Process.*, vol. 65, no. 16, pp. 4406–4421, Aug. 2017.
[11] A. Loukas, A. Simonetto, and G. Leus, "Distributed autoregressive moving average graph filters," *IEEE Sig. Process. Letters*, vol. 22, no. 11, pp. 1931–1935, Nov. 2015.
[12] A. Loukas, "Distributed graph filters," Ph.D. dissertation, Delft University of Technology, March 2015.
[13] (2019) Giraph. [Online]. Available: https://giraph.apache.org
[14] (2019) Spark. [Online]. Available: https://spark.apache.org
[15] (2019) Dgraph. [Online]. Available: https://dgraph.io
[16] B. Awerbuch, "Complexity of network synchronization," *J. ACM*, vol. 32, no. 4, p. 804–823, Oct. 1985.
[17] O. Teke and P. P. Vaidyanathan, "IIR filtering on graphs with random node-asynchronous updates," *IEEE Transactions on Signal Processing*, vol. 68, pp. 3945–3960, 2020.
[18] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, N.J. Prentice Hall, 1993.
[19] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *International Conference on Machine Learning (ICML)*, 2003, pp. 912–919.
[20] K. Avrachenkov, A. Mishenin, P. Gonçalves, and M. Sokol, "Generalized optimization framework for graph-based semi-supervised learning," in *SIAM International Conf. Data Mining*, 2012, pp. 966–974.
[21] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge University Press, 1994.
[22] O. Teke and P. P. Vaidyanathan, "Random node-asynchronous updates on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 11, pp. 2794–2809, June 2019.
[23] ——, "Random node-asynchronous graph computations," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 64–73, November 2020.
[24] ——, "Joint vertex-time filtering on graphs with random node-asynchronous updates," *Submitted to IEEE Transactions on Signal Processing*, 2020.
[25] ——. (2020) Real-time visualization of node-asynchronous graph filtering. [Online]. Available: http://systems.caltech.edu/dsp/students/oteke/files/asyncPoly.mp4