Signal Processing 187 (2021) 108149

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Signal processing on higher-order networks: Livin' on the edge... and beyond

Michael T. Schaub[a,*], Yu Zhu[b,1], Jean-Baptiste Seby[c], T. Mitchell Roddenberry[b], Santiago Segarra[b]

[a] Department of Computer Science, RWTH Aachen University, Germany
[b] Department of Electrical and Computer Eng., Rice University, Houston, TX, USA
[c] Université Paris-Sud, Orsay, France

ARTICLE INFO

Article history:
Received 10 January 2021
Revised 9 April 2021
Accepted 5 May 2021
Available online 18 May 2021

Keywords:
Simplicial complex
Hypergraph
Higher-order network
Graph signal processing
Node embeddings

ABSTRACT

In this tutorial, we provide a didactic treatment of the emerging topic of signal processing on higher-order networks. Drawing analogies from discrete and graph signal processing, we introduce the building blocks for processing data on simplicial complexes and hypergraphs, two common higher-order network abstractions that can incorporate polyadic relationships. We provide brief introductions to simplicial complexes and hypergraphs, with a special emphasis on the concepts needed for the processing of signals supported on these structures. Specifically, we discuss Fourier analysis, signal denoising, signal interpolation, node embeddings, and nonlinear processing through neural networks, using these two higher-order network models. In the context of simplicial complexes, we specifically focus on signal processing using the Hodge Laplacian matrix, a multi-relational operator that leverages the special structure of simplicial complexes and generalizes desirable properties of the Laplacian matrix in graph signal processing. For hypergraphs, we present both matrix and tensor representations, and discuss the trade-offs in adopting one or the other. We also highlight limitations and potential research avenues, both to inform practitioners and to motivate the contribution of new researchers to the area.

1. Introduction

Graphs provide a powerful abstraction for systems consisting of (dynamically) interacting entities. By encoding these entities as nodes and the interaction between them as edges in a graph, we can model a large range of systems in an elegant, conceptually simple framework. Accordingly, graphs have been used as models in a broad range of application areas [1,2], including neuroscience [3,4], urban transportation [5], and social sciences [6]. Many of these applications may be understood in terms of graph signal processing (GSP), which provides a unifying framework for processing data supported on graphs. In GSP, we model complex data dependencies as the edges of graphs that relate signals on the nodes. In this way GSP extends and subsumes classical signal processing concepts and tools such as the Fourier transforms, filtering, sampling and reconstruction of signals, and others, to a graph-based setting [7–9].

To enable computations with graph-based data, we typically encode the graph structure in an adjacency matrix or its associated (normalized or combinatorial) Laplacian matrix. Rather than considering these matrices as a simple table that records pairwise coupling between nodes, it is fruitful to think of these matrices as linear operators that map data from the node space to itself. By analyzing the properties of these maps – e.g., their spectral properties – we can reveal important aspects both about the graphs themselves as well as signals defined on the nodes. Choosing an appropriate matrix operator associated with the graph structure is thus a key factor in gaining deeper insights about graphs and graph signals. In GSP, we call such maps that relate data associated with different nodes *graph shift operators*. Graph shift operators are natural generalizations of the classical time delay, and constitute the fundamental building blocks of graph filters and other more sophisticated processing architectures [10]. The rapid advancement of GSP has benefited significantly from spectral and algebraic graph theory [11], in which the properties of matrices such as the adjacency matrix and the Laplacian have been extensively studied.

By construction, graph-based representations do not account for interactions between more than two nodes, even though such

* Corresponding author.
E-mail address: michael.schaub@rwth-aachen.de (M.T. Schaub).
[1] Equal contribution

https://doi.org/10.1016/j.sigpro.2021.108149

multi-way interactions are widespread in complex systems: multiple neurons can fire at the same time [12], biochemical reactions usually include more than two proteins [13], and people interact in small groups [14]. To account for such polyadic interactions, a number of modeling frameworks have been proposed in the literature to represent higher-order relations, including simplicial complexes [15], hypergraphs [16], and others [17]. However, in comparison to this line of work on representing the *structure* of complex multi-relational systems, the literature on the *data processing* for signals defined on higher-order networks is comparatively sparse. In this tutorial paper, we focus on the topic of *signal processing on simplicial complexes and hypergraphs*. Following a high-level didactic style, we concentrate on the algebraic representations of these objects, and discuss how the choice of this algebraic representation can influence the way in which we analyze and model signals associated with higher-order networks.

Similarly to graphs, higher-order interactions can be encoded in terms of matrices or, more generally, tensors. Two of the most prominent abstractions for such polyadic data are simplicial complexes [15] and hypergraphs [16]. As we will see in the following, both of these abstractions have certain advantages and disadvantages: Hypergraphs are somewhat more flexible in terms of the relationships they can represent, which can be desirable in terms of modeling. Indeed a simplicial complex may be interpreted as a specific hypergraph for which only certain sets of hyperedges are allowed. The advantage of simplicial complexes, however, is that this additional structure provides deep links to computational geometry and algebraic topology, which can facilitate both the computation and interpretation of the processed signals [18].

Analogously to the graph case, we encode higher-order relations in terms of incidence matrices or tensors that provide an algebraic description of these two data models. Clearly, the choice of the linear (or multilinear) operator representing higher-order interactions will matter for revealing interesting properties about the data, leading to the key question of how to choose an appropriate abstraction for this kind of data. In comparison to graphs, the analysis of higher-order interaction data is more challenging due to several factors: (i) There exists a combinatorially large number of possible interactions: two-way, three-way, and so on. Hence, very large matrices and tensors are needed to capture all these relations; (ii) The large dimensionality of these representations gives rise to computational and statistical issues on how to efficiently extract information from higher-order data; and (iii) The theory on the structure of higher-order networks is largely unexplored relative to that of graphs. In the following, we will primarily focus on the question of choosing an appropriate algebraic descriptor to implement various signal processing tasks on simplicial complexes and hypergraphs. Specifically, we will consider the modeling assumptions inherent to an abstraction based on simplicial complexes versus hypergraphs, and discuss the relative advantages and disadvantages of a number of associated matrix and tensor descriptions that have been proposed. To make our discussions more concrete we provide a number of illustrative examples to demonstrate how the choice of an algebraic description can directly effect the type of results we can obtain.

Outline. We first briefly recap selected concepts from signal processing and GSP in Section 2. In Section 3, we present tools from algebraic topology and their use in representing higher-order interactions with simplicial complexes. In Section 4, we describe methods to analyze signals defined on simplicial complexes. We then turn our attention to hypergraphs in Section 5, and focus on the modeling of higher-order interactions via hypergraphs. Section 6 then builds on these models and outlines some of the existing methods for signal processing and learning on hypergraphs. Finally, in Section 7, we close with a brief discussion summarizing the main takeaways and laying out directions for future research.

2. Signal processing on graphs: a selective overview

Before discussing signal processing on higher-order networks, we revisit principles from signal processing and GSP [7–9] and recall some important problem setups, which will later guide our discussion on higher-order signal processing. In this tutorial, we focus on undirected graphs (and higher-order networks), although signal processing on directed graphs has been studied as well [19,20].

2.1. Central tenets of discrete signal processing

In discrete signal processing (DSP), signals are processed by filters. A linear filter $\mathbf{H}$ is an operator that takes a signal as input and produces a transformed signal as output. This linear filtering operation is represented by a matrix-vector multiplication $\mathbf{s}_{out} = \mathbf{H}\mathbf{s}_{in}$ and defines a *linear* system. A special role is played by the circular *time shift* filter $\mathbf{S}$, a linear operator that delays the signal by one sample. This so-called shift operator underpins the class of time shift-invariant filters, which is arguably the most important class of linear filters in practice. Specifically, in classical DSP, every linear time shift-invariant filter can be built based on a matrix polynomial of the time-shift $\mathbf{S}$ [21].

A filter represented by the matrix $\mathbf{H}$ is shift-invariant if it commutes with the shift operator, i.e., $\mathbf{SH} = \mathbf{HS}$. This implies that $\mathbf{H}$ and $\mathbf{S}$ preserve each others eigenspaces. Since the cyclic shift $\mathbf{S}$ is a circulant matrix that is diagonalizable by discrete Fourier modes, this implies that the action of any shift-invariant linear filter in DSP can be understood by means of a Fourier transform. Specifically, the eigenvectors of the cyclic time-shift operator provide an orthogonal basis for linear time shift-invariant processing of discrete-time signals. Thus time-shift invariant filters are naturally interpretable by Fourier analysis [21].

2.2. Graphs, incidence matrices, and the graph Laplacian

An undirected graph $\mathcal{G}$ is defined by a set of nodes $\mathcal{V} = \{v_1, \cdots, v_N\}$ with cardinality N and a set of edges $\mathcal{E}$ with cardinality E composed of unordered pairs of nodes in $\mathcal{V}$. Edges can be stored in the symmetric adjacency matrix $\mathbf{A}$ whose entries are given by $A_{ij} = A_{ji} = 1$ if $\{i, j\} \in \mathcal{E}$ and 0 otherwise. Given the degree matrix $\mathbf{D} = \mathrm{diag}(\mathbf{A1})$, the graph Laplacian associated with $\mathcal{G}$ is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Alternatively to the adjacency matrix $\mathbf{A}$, we can collect interactions between the nodes in the graph via the incidence matrix $\mathbf{B} \in \mathbb{R}^{N \times E}$. For each edge e we define an arbitrary orientation, which we denote by $e = (i, j)$. We think of such an edge e as being oriented from tail node i to its head node j. Based on this orientation, the incidence matrix $\mathbf{B}$ is defined such that $B_{ie} = -B_{je} = -1$ and $B_{ke} = 0$ otherwise. Using this definition we can provide an equivalent expression for the graph Laplacian as $\mathbf{L} = \mathbf{B}\mathbf{B}^\top$. In the remainder of this paper, we choose an edge-orientation induced by the lexicographic ordering of the nodes, i.e., edges will always be oriented such that they point from a node with lower index to a node with higher index. However, we emphasize that this orientation is arbitrary and is distinct from the notion of a *directed* edge.

2.3. Graph signal processing

GSP generalizes the concepts and tools from DSP to signals defined on the nodes of graphs. A *graph signal* $\mathbf{s} : \mathcal{V} \to \mathbb{R}$ is a map from the set of nodes $\mathcal{V}$ to the set of real numbers $\mathbb{R}$. This defines an isomorphism between the set of nodes and the set of real-valued vectors of length N, so any graph signal may be represented

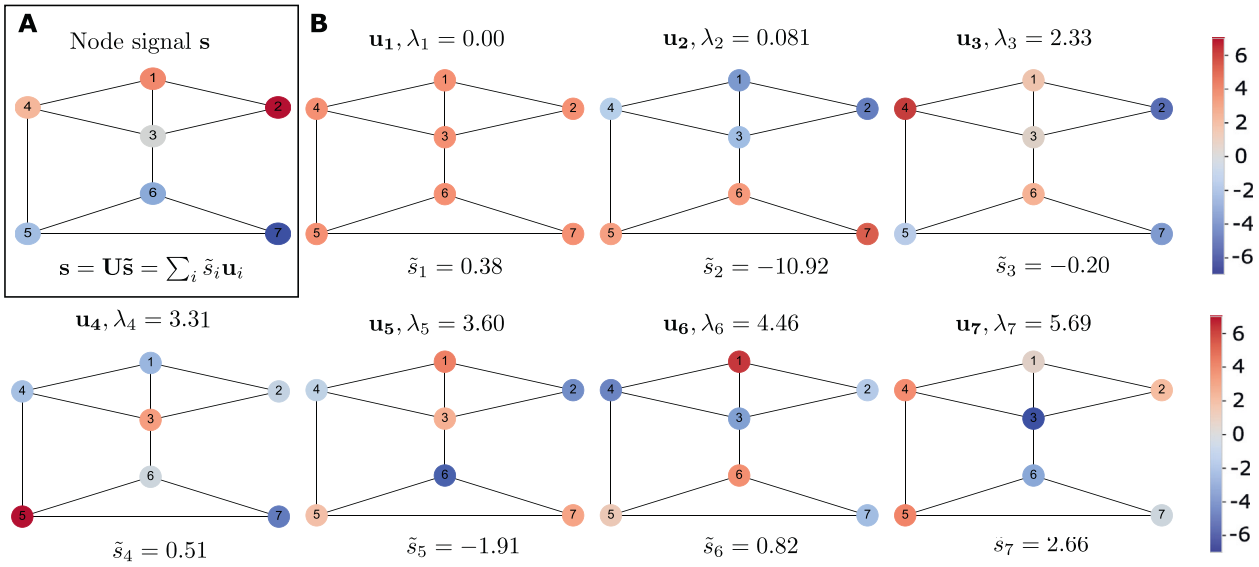

Fig. 1. Graph signal and its Fourier decomposition. A Graph signal defined on the nodes of the graph. **B** Eigenvector and eigenvalue pairs of the graph Laplacian **L**. We visualize each of the eigenvectors in terms of a graph signal and order them from low to high graph frequencies, corresponding to a decrease in "smoothness". The decomposition of the node signal **s** into this basis provides the Fourier coefficients in $\tilde{\mathbf{s}}$ as indicated at the bottom of each eigenvector representation.

as a vector $\mathbf{s} = [s_1, s_2, \ldots, s_N]^\top \in \mathbb{R}^N$. An example of a graph signal can be seen in Fig. 1A, where the signal values at each node are indicated by the node color. Similarly to DSP, filtering in GSP can be represented by a matrix-vector multiplication operation $\mathbf{s}_{out} = \mathbf{H}\mathbf{s}_{in}$. The analog of the shift operator **S** in the GSP setting is any operator that captures the relational dependencies between nodes, including the adjacency matrix **A**, the Laplacian matrix **L**, or variations of these operators [8,9].

As we are considering undirected graphs here, the choice of a shift operator imparts a natural orthogonal basis **U** in which to represent the signal. Given the eigenvalue decomposition of the shift operator $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ and a filtering weight function $h : \mathbb{R} \to \mathbb{R}$, we can express any shift-invariant filter in this basis as:

$$\mathbf{H} = \sum_{k=1}^{N} h(\lambda_k)\mathbf{u}_k\mathbf{u}_k^\top = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\top, \tag{1}$$

where we have used the shorthand notation $h(\mathbf{\Lambda}) = \mathrm{diag}(h(\lambda_1), \cdots, h(\lambda_N))$. By analogy to the Fourier basis in DSP, the eigenvectors **U** of the shift operator are said to define a *graph Fourier transform* (GFT), and $h(\mathbf{\Lambda})$ is called the frequency response of the filter **H**. Specifically, the GFT of a graph signal **s** is given by $\tilde{\mathbf{s}} = \mathbf{U}^\top\mathbf{s}$, while the inverse GFT is given by $\mathbf{s} = \mathbf{U}\tilde{\mathbf{s}}$ [7,9].

As our discussion emphasizes, any filtered signal $\mathbf{s}_{out} = \mathbf{H}\mathbf{s}_{in}$ on an undirected graph can be understood in terms of three steps: (i) project the signal into the graph Fourier domain, i.e., express it in the orthogonal basis **U** (via multiplication with $\mathbf{U}^\top$); (ii) amplify certain modes and attenuate others (via multiplication with $h(\mathbf{\Lambda})$), and (iii) push back the signal to the original node domain (via multiplication with **U**). The choice of an appropriate shift operator is thus crucial, as its eigenvectors define the basis for any shift-invariant graph filter for undirected graphs. We will encounter this aspect again when considering signal processing on higher-order networks.

In the context of GSP, we focus on the graph Laplacian as a shift operator. This choice has the following advantages. First, **L** is positive semidefinite, so that all the graph frequencies (eigenvalues) are real and non-negative. This enables us to order the GFT basis vectors (eigenvectors) in a natural way. Second, by considering the variational characterization of the eigenvalues of the Laplacian in terms of the Rayleigh quotient $r(\mathbf{s}) = \mathbf{s}^\top\mathbf{L}\mathbf{s}/\mathbf{s}^\top\mathbf{s} = \sum_{ij} A_{ij}(s_i - s_j)^2/(2\|\mathbf{s}\|^2)$, it can be shown that eigenvectors associated with small eigenvalues have small variation along the edges of the graph (low frequency) and eigenvectors associated with large eigenvalues have large variation along edges (high frequency). In particular, eigenvectors associated with eigenvalue 0 are constant over connected components. An illustration of this is given in Figure 1B, which displays the individual basis vectors of the graph Laplacian, and the coefficients with which these basis vectors would have to be weighted to obtain the previously considered graph signal in Figure 1A.

2.4. Graph signal processing: illustrative problems and applications

Over the last few years, several relevant problems have been addressed using GSP tools including sampling and reconstruction of graph signals [22–24], (blind) deconvolution [25,26], and network topology inference [27–30], to name a few. We now introduce a subset of illustrative problems and application scenarios that we will revisit in the context of higher-order signal processing.

2.4.1. Fourier analysis: node embeddings and Laplacian eigenmaps

As discussed above, the GFT of a graph signal provides a fundamental tool of GSP. While we are often interested in filtering a signal and representing it in the vertex space, the Fourier representation can also be used to gain insight about specific graph components by considering a frequency domain representation of the indicator vector associated with the vertices of interest. In particular, by considering a truncated Fourier domain representation of the indicator vectors of individual nodes, we can recover a number of spectral node embeddings that have found a broad range of applications (see also [31] for a related discussion). Specifically, by considering a truncated Fourier domain representation based on the normalized Laplacian as a shift operator, we recover a variant of the so-called Laplacian eigenmaps [32], and by additionally incorporating a scaling associated with the eigenvalues, we can recover the diffusion map embedding [31,33].

We remark that while most of these spectral node embeddings focus on low frequency eigenvectors, high frequency components can also be of interest for embeddings. For instance, if the graph to be analyzed is almost bipartite, then the eigenvectors associated

with the highest frequencies of the graph Laplacian will reveal the two (almost) independent node sets in the graph. Other types of (nonlinear) node embeddings may also be viewed through a GSP lens, e.g., certain node embeddings derived from graph neural networks (cf. Section 2.4.4). We refer to [34] for an extensive discussion on the highly active area of node representation learning on graphs.

2.4.2. Signal smoothing and denoising

A canonical task in GSP is to denoise (smooth out) a noisy signal $\mathbf{y} = \mathbf{y}_0 + \boldsymbol{\epsilon} \in \mathbb{R}^N$, where $\mathbf{y}_0$ is the true signal we aim to recover and $\boldsymbol{\epsilon}$ is a vector of zero-mean white Gaussian noise [35]. A natural assumption is that the signal should be *smooth* on nearby nodes in terms of the underlying graph, so that neighboring nodes will tend to take on similar values. Following our above discussion, this amounts to assuming that the signal has a low-pass characteristic, i.e., can be well-represented by the low frequency eigenvectors of the Laplacian. Indeed, the eigenvectors of the Laplacian associated with low eigenvalues are smooth on clusters, i.e. their total variation is low within clusters and high over edges between clusters.

We formalize the above problem in terms of the following optimization problem [27,36]

$$\min_{\hat{\mathbf{y}}} \{ \left\| \hat{\mathbf{y}} - \mathbf{y} \right\|_2^2 + \alpha \hat{\mathbf{y}}^\top \mathbf{L} \hat{\mathbf{y}} \}, \tag{2}$$

where $\hat{\mathbf{y}}$ is the estimate of the true signal $\mathbf{y}_0$. The coefficient $\alpha > 0$ can be interpreted as a regularization parameter that trades-off the smoothness promoted by minimizing the quadratic form $\hat{\mathbf{y}}^\top \mathbf{L} \hat{\mathbf{y}} = \sum_{ij} A_{ij} (\hat{y}_i - \hat{y}_j)^2 / 2$ and the fit to the observed signal in terms of the squared 2-norm. The optimal solution for (2) is given by [27]

$$\hat{\mathbf{y}} = (\mathbf{I} + \alpha \mathbf{L})^{-1} \mathbf{y}. \tag{3}$$

A different procedure to obtain a signal estimate is the iterative smoothing operation

$$\hat{\mathbf{y}} = (\mathbf{I} - \mu \mathbf{L})^k \mathbf{y}, \tag{4}$$

for a certain fixed number of iterations k and a suitably chosen update parameter μ. This may be interpreted in terms of k gradient descent steps of the cost function $\hat{\mathbf{y}}^\top \mathbf{L} \hat{\mathbf{y}}$.

Matching the signal modeling assumption of a smooth signal, the denoising and smoothing operators defined in (3) and (4) are instances of *low-pass filters*, i.e., filters whose frequency responses $h(\boldsymbol{\lambda}) = \mathrm{diag}(\mathbf{U}^\top \mathbf{H} \mathbf{U})$ are vectors of non-increasing (decreasing) values. In the GSP context, the low-pass filtering operation guarantees that variations over neighboring nodes are smoothed out, in line with the intuition of the optimization problem defined in (2).

2.4.3. Graph signal interpolation

Another common task in GSP is signal interpolation, which can alternatively be interpreted in terms of graph-based *semi-supervised learning* [23,37]. Suppose that we are given signal values (labels) for a subset of the nodes $\mathcal{V}^L \subset \mathcal{V}$ of a graph. Our goal is to interpolate these assignments and to provide a label to all unlabeled nodes $\mathcal{V}^U = \mathcal{V} \setminus \mathcal{V}^L$.

As in the signal denoising case, it is natural to adopt a smoothness assumption that posits that well-connected nodes have similar labels [38]. This motivates the following constrained optimization problem [39]

$$\min_{\hat{\mathbf{y}}} \left\| \mathbf{B}^\top \hat{\mathbf{y}} \right\|_2^2, \tag{5}$$
$$\text{s.t. } \hat{y}_i = y_i, \text{ for all } v_i \in \mathcal{V}^L,$$

which aims to minimize the sum-of-squares label difference between connected nodes under the constraint that the observed node labels y_i should be accounted for in the optimal solution. Notice that the objective function in (5) can again be written in terms of the quadratic form of the graph Laplacian $\left\| \mathbf{B}^\top \hat{\mathbf{y}} \right\|_2^2 = \sum_{(i,j) \in \mathcal{E}} (\hat{y}_i - \hat{y}_j)^2 = \hat{\mathbf{y}}^\top \mathbf{L} \hat{\mathbf{y}}$, highlighting the low-pass modeling assumption inherent in the optimization problem (5).

2.4.4. Graph neural networks

Motivated by spectral interpretations of filters and shift operators in the domain of graph signal processing, *graph neural networks* [40,41] have emerged as a popular approach to incorporate nonlinearities in the graph signal processing pipeline for purposes of node embedding [42–44], node classification [45,46], and graph classification [46]. Graph neural network architectures combine notions of graph filtering, permutation invariance, and graph Fourier analysis with nonlinear models from the design of neural networks.

One such architecture is the well-known *graph convolutional network* [45], which resembles the functional form of (4) with interleaved nonlinear, elementwise activation functions, i.e., for a set of F_0 input features gathered in the columns of a matrix $\mathbf{Y}_0 \in \mathbb{R}^{N \times F_0}$,

$$\mathbf{Y}_k = \sigma(\mathbf{H} \mathbf{Y}_{k-1} \mathbf{W}_k), \tag{6}$$

where we take $\mathbf{Y}_K$ for some integer K as the output, $\{\mathbf{W}_k \in \mathbb{R}^{F_{k-1} \times F_k}\}_{k=1}^K$ are learnable weight matrices that perform linear transformations in the feature space, $\mathbf{H}$ is a certain graph filter, and $\sigma(\cdot)$ is a generally nonlinear activation function applied elementwise. Specifically, [45] uses a normalized version of the graph Laplacian as a first-order filter $\mathbf{H}$, and the ReLU activation function for $\sigma(\cdot)$.

A closer look at (6) reveals a connection with the iterative smoothing method of (4). Taking $\sigma(\cdot)$ to be the identity mapping, we see that (6) can be expressed as a linear graph filter independently applied to each of the F_0 features, with output defined as linear combinations of these filtered features at each node via the matrices $\{\mathbf{W}_k\}$. That is,

$$\mathbf{Y}_K = (\mathbf{H}^K \mathbf{Y}_0)(\mathbf{W}_1 \mathbf{W}_2 \dots \mathbf{W}_K), \tag{7}$$

where $\mathbf{H}^K$ itself represents a shift-invariant graph filter, due to the assumed shift-invariance of $\mathbf{H}$. Taking $F_0 = F_K = 1$ and $\mathbf{H} = (\mathbf{I} - \mu \mathbf{L})$ recovers the iterative smoothing procedure of (4). However, by interleaving nonlinear functions as in (6) and taking linear combinations of features via $\{\mathbf{W}_k\}$, we allow the architecture to *learn* more sophisticated, nonlinear relationships between the nodes and node features by finding optimal weights $\{\mathbf{W}_k\}$ for a suitable loss function.

There are many variants of the graph neural network architecture, designed for tasks ranging from semi-supervised learning [45] to graph classification [46]. We refer the reader to the survey paper [41] for further details, as well as [47] for a view focused on graph signal processing in particular.

3. Modeling higher-order interactions with simplicial complexes

In this section, we recap some of the mathematical underpinnings of simplicial complexes. We focus in particular on the Hodge Laplacian [15,48,49], which extends the graph Laplacian as a natural shift operator for simplicial complexes. Specifically, we discuss how the eigenvectors of the Hodge Laplacian provide an interpretable orthogonal basis for signals defined on simplicial complexes by means of the Hodge decomposition.

3.1. Background on simplicial complexes

Given a finite set of vertices $\mathcal{V}$, a k-simplex $\mathcal{S}^k$ is a subset of $\mathcal{V}$ with cardinality $k+1$. A simplicial complex $\mathcal{X}$ is a set of simplices

such that for any k-simplex $\mathcal{S}^k$ in $\mathcal{X}$, any subset of $\mathcal{S}^k$ must also be in $\mathcal{X}$. A *face* of a simplex $\mathcal{S}^k$ is a subset of $\mathcal{S}^k$ with cardinality k. A *co-face* $\mathcal{S}^{k+1}$ of a simplex $\mathcal{S}^k$ is a $(k+1)$-simplex such that $\mathcal{S}^k$ is a subset of $\mathcal{S}^{k+1}$. More detailed discussions and definitions can, e.g., be found in [48,50,51].

Example 1. Figure 2A provides an example of a simplicial complex. Here, simplices of order 0 are depicted as nodes, simplices of order 1 as edges, and simplices of order 2 are displayed as gray, filled triangles. Note how the edges $\{1,3\},\{1,4\}$ and $\{3,4\}$ are faces of the 2-simplex $\{1,3,4\}$. The 2-simplex $\{5,6,7\}$ is a co-face of the edges $\{5,6\},\{5,7\}$ and $\{6,7\}$.

For computational purposes, we define an orientation for each simplex by fixing an ordering of its vertices. This ordering induces a reference orientation by increasing vertex label. Based on the reference orientation for each simplex, we introduce a book-keeping of the relationships between $(k-1)$-simplices and k-simplices via linear maps called *boundary operators* that record higher-order interactions in networks. As the simplicial complexes we consider are all of finite order, these boundary operators can be represented by matrices $\mathbf{B}_k$. The rows of $\mathbf{B}_k$ are indexed by $(k-1)$-simplices and the columns of $\mathbf{B}_k$ are indexed by k-simplices. For instance, $\mathbf{B}_1$ is nothing but the node-to-edge incidence matrix denoted $\mathbf{B}$ in Section 2, while $\mathbf{B}_2$ is the edge-to-triangle incidence matrix.

Example 2. We adopt the lexicographic order to define the reference orientation of simplices in Fig. 2. The corresponding boundary maps $\mathbf{B}_1$ and $\mathbf{B}_2$ are then given by

$$\mathbf{B}_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccccc} (1,2) & (1,3) & (1,4) & (2,3) & (3,4) & (3,6) & (4,5) & (5,6) & (5,7) & (6,7) \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

$$\mathbf{B}_2 = \begin{array}{c} \\ (1,2) \\ (1,3) \\ (1,4) \\ (2,3) \\ (3,4) \\ (3,6) \\ (4,5) \\ (5,6) \\ (5,7) \\ (6,7) \end{array} \begin{array}{c} \begin{array}{cc} (1,3,4) & (5,6,7) \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ -1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} \end{array}$$

We may consider signals defined on any k-simplices (nodes, edges, triangles, etc.) of a simplicial complex as illustrated in Figure 2 B-D. Just like for graph signals, we need to establish an appropriate shift operator to process such signals. While there are many possibilities, we will show in the next section that a natural choice for the shift operator is the Hodge Laplacian, a generalization of the graph Laplacian rooted in algebraic topology.

3.2. The Hodge Laplacian as a shift operator for simplicial complexes

Based on the incidence matrices defined above, we can define a sequence of so-called *Hodge Laplacians* [48]. Specifically, the *k-th combinatorial Hodge Laplacian*, originally introduced in [52], is given by [48,52]:

$$\mathbf{L}_k = \mathbf{B}_k^\top \mathbf{B}_k + \mathbf{B}_{k+1}\mathbf{B}_{k+1}^\top. \tag{8}$$

Notice that, according to this definition, the graph Laplacian corresponds to $\mathbf{L}_0 = \mathbf{B}_1\mathbf{B}_1^\top$ with $\mathbf{B}_0 = 0$. More generally, by equipping all spaces with an inner product induced by positive diagonal matrices, we can define a weighted version of the Hodge Laplacian (see, e.g., [48–50,53]). This weighted Hodge Laplacian encapsulates operators such as the random walk graph Laplacian or the normalized graph Laplacian as special cases. For simplicity, in this paper we concentrate on the unweighted case.

Just like the graph Laplacian provides a useful choice for a shift operator for node signals defined on a graph due to its (spectral) properties, the Hodge Laplacian and its weighted variants provide a natural shift operator for signals defined on the edges of a simplicial complex (or graph). As the edges in our simplicial complexes are equipped with a chosen reference orientation, the Hodge Laplacian is in particular relevant as shift operator if the signals considered are indeed oriented, e.g., correspond to some kind of edge-flow in case of a signal on edges.

Similar to the graph Laplacian, the Hodge Laplacian is positive semi-definite, which ensures that we can interpret its eigenvalues in terms of non-negative frequencies. Moreover, these frequencies are again aligned with a specific type of signal-smoothness displayed by the eigenvectors of the Hodge Laplacian. For signals on general k-simplices, this notion of smoothness can be understood by means of the so called *Hodge decomposition* [48–50], which states that the space of k-simplex signals can be decomposed into three orthogonal subspaces

$$\mathbb{R}^{N_k} = \mathrm{im}(\mathbf{B}_{k+1}) \oplus \mathrm{im}(\mathbf{B}_k^\top) \oplus \ker(\mathbf{L}_k), \tag{9}$$

where $\mathrm{im}(\cdot)$ and $\ker(\cdot)$ are shorthand for the *image* and *kernel* spaces of the respective matrices, $\oplus$ represents the union of orthogonal subspaces, and N_k is the cardinality of the space of signals on k-simplices (i.e., $N_0 = N$ for the node signals, and $N_1 = E$ for edge signals). Here we have (i) made use of the fact that a signal on a finite dimensional set of N_k simplices is isomorphic to $\mathbb{R}^{N_k}$; and (ii) implicitly assumed that we are only interested in real-valued signals and thus a Hodge decomposition for a real valued vector space (see [48] for a more detailed discussion).

To facilitate the discussion on how the Hodge decomposition (9) can be related to a notion of smooth signals let us consider the concrete case $k = 1$ with Hodge Laplacian $\mathbf{L}_1 = \mathbf{B}_1^\top\mathbf{B}_1 + \mathbf{B}_2\mathbf{B}_2^\top$ for illustration [49,54,55]. In this case, we can provide the following meaning to the three subspaces considered in (9). First, the space $\mathrm{im}(\mathbf{B}_1^\top)$ can be considered as the space of gradient flows (or potential flows). Specifically, since $\mathrm{im}(\mathbf{B}_1^\top) = \{\mathbf{f} = \mathbf{B}_1^\top\mathbf{v}$, for some $\mathbf{v} \in \mathbb{R}^N\}$ we may create any such flow according to the following recipe: (i) assign a scalar potential to all the nodes; (ii) induce a flow along the edges by considering the difference of the potentials on the respective endpoints. Clearly, we cannot create a positive net-flow along any closed path within a complex if the flow at every edge is computed according to the gradient (difference) of the node potentials in the chosen reference orientation: the difference between the potentials along any closed path has to sum to zero, by construction. orientation. Accordingly, the space $\ker(\mathbf{B}_1) = \mathrm{im}(\mathbf{B}_2) \oplus \ker(\mathbf{L}_1)$ that is orthogonal to $\mathrm{im}(\mathbf{B}_1^\top)$ is the so-called cycle space. As indicated, the cycle space is spanned by two types of cyclic flows. The space $\mathrm{im}(\mathbf{B}_2)$ consists of curl flows and its elements are flows that can be composed of combinations of local circulations along any 2-simplex. Specifically, we may assign a scalar potential to each 2-simplex and consider the induced flows $\mathbf{f} = \mathbf{B}_2\mathbf{t}$, where $\mathbf{t}$ is the vector of 2-simplex potentials. Note that every column of $\mathbf{B}_2$ creates a triangular circulation around the respective 2-simplex along its chosen reference orientation. Hence, these flows correspond to local cycles associated with the 2-simplices present in the simplicial complex. Finally $\ker(\mathbf{L}_1)$

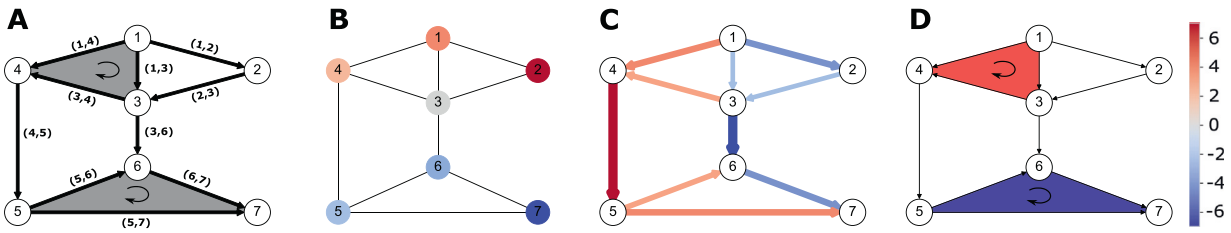

Fig. 2. Signals on simplicial complexes of different order. A: Structure of the simplicial complexes used as a running example in the text. Arrows represent the chosen reference orientation. Shaded areas correspond to the 2-simplices $\{1,3,4\}$ and $\{5,6,7\}$. **B**: Signal on 0-simplices (nodes). **C**: Signal on 1-simplices (edges). **D**: Signal on 2-simplices (triangles).

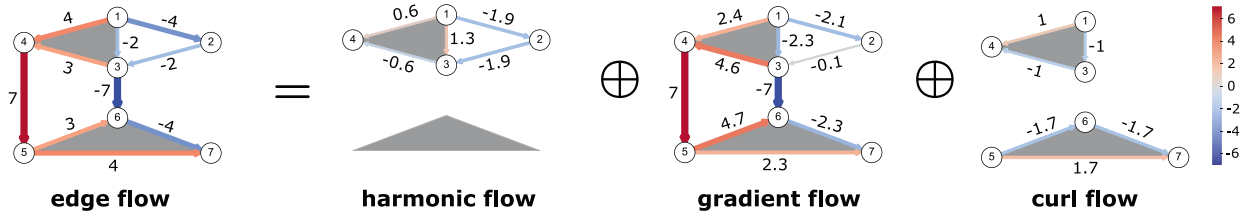

Fig. 3. Hodge decomposition of the edge flow in the example from Fig. 2. Any edge flow (left) can be decomposed into a harmonic flow, a gradient flow and a curl flow.

is the harmonic space, whose elements correspond to (global) circulations that are not representable as a linear combination of curl flows.

Example 3. In Fig. 3, we consider the edge flow $\mathbf{c} = [-4,-2,4,-2,3,-7,7,3,4,-4]^\top$. The Hodge decomposition $\mathbf{c} = \mathbf{h} \oplus \mathbf{g} \oplus \mathbf{r}$ enables us to decompose the edge flow $\mathbf{c}$ into a harmonic, gradient and curl part, respectively denoted by $\mathbf{h}, \mathbf{g}$ and $\mathbf{r}$. Components $\mathbf{g}$ and $\mathbf{r}$ are given by

$$\mathbf{g} = \mathbf{B}_1^\top \mathbf{p}, \qquad \mathbf{r} = \mathbf{B}_2 \mathbf{w}. \tag{10}$$

Since the Hodge decomposition is orthogonal, $\mathbf{p}$ and $\mathbf{w}$ are the solutions of the following least squares problems

$$\min_{\mathbf{p}} \left\| \mathbf{B}_1^\top \mathbf{p} - \mathbf{c} \right\|_2, \qquad \min_{\mathbf{w}} \left\| \mathbf{B}_2 \mathbf{w} - \mathbf{c} \right\|_2. \tag{11}$$

The harmonic component satisfies $\mathbf{L}_1 \mathbf{h} = \mathbf{0}$, and by the orthogonality of the Hodge decomposition, it can be obtained by $\mathbf{h} = \mathbf{c} - \mathbf{g} - \mathbf{r}$. As explained in the text, $\mathbf{g}$ is an element of the space $\mathrm{im}(\mathbf{B}_1^\top)$, i.e., the gradient space or space of cycle-free flows. Components $\mathbf{h} \in \ker(\mathbf{L}_1)$ and $\mathbf{r} \in \mathrm{im}(\mathbf{B}_2)$ are elements of the cycle space $\ker(\mathbf{B}_1) = \mathrm{im}(\mathbf{B}_2) \oplus \ker(\mathbf{L}_1)$. As can be seen in Fig. 3, the curl component $\mathbf{r}$ can be decomposed into two local circulations, of absolute magnitude 1 and 1.7, respectively.

Importantly the gradient, curl and harmonic subspaces are spanned by certain subsets of eigenvectors of $\mathbf{L}_1$ as the following lemma, which can be verified by direct computation [49,56], shows.

Lemma 4. *Let $\mathbf{L}_1 = \mathbf{B}_1^\top \mathbf{B}_1 + \mathbf{B}_2 \mathbf{B}_2^\top$ be the Hodge 1-Laplacian of a simplicial complex. Then the eigenvectors associated with nonzero eigenvalues of $\mathbf{L}_1$ comprise two groups that span the gradient space and the curl space respectively.*

- *Consider any eigenvector $\mathbf{v}_i$ of the graph Laplacian $\mathbf{L}_0$ associated with a nonzero eigenvalue λ_i. Then $\mathbf{u}_{\mathrm{grad}}^{(i)} = \mathbf{B}_1^\top \mathbf{v}_i$ is an eigenvector of $\mathbf{L}_1$ with the same eigenvalue λ_i. Moreover $\mathbf{U}_{\mathrm{grad}} = [\mathbf{u}_{\mathrm{grad}}^{(1)}, \mathbf{u}_{\mathrm{grad}}^{(2)}, \ldots]$ spans the space of all gradient flows.*
- *Consider any eigenvector $\mathbf{t}_i$ of the "2-simplex coupling matrix" $\mathbf{T} = \mathbf{B}_2^\top \mathbf{B}_2$ associated with a nonzero eigenvalue θ_i. Then $\mathbf{u}_{\mathrm{curl}}^{(i)} = \mathbf{B}_2 \mathbf{t}_i$ is an eigenvector of $\mathbf{L}_1$ with the same eigenvalue θ_i. Moreover $\mathbf{U}_{\mathrm{curl}} = [\mathbf{u}_{\mathrm{curl}}^{(1)}, \mathbf{u}_{\mathrm{curl}}^{(2)}, \ldots]$ spans the space of all curl flows.*

The above result shows that, unlike for node signals, edge-flow signals can have a high frequency contribution, reflected by a high component in the corresponding projected space, due to two different types of (orthogonal) basis components being present in the signal: a high frequency may arise both due to a curl component as well as a strong gradient component present in the edge-flow. This has certain consequences for the filtering of edge signals that we will discuss in more detail in the following section.

4. Signal processing and learning on simplicial complexes

Using the algebraic framework of simplicial complexes as discussed in Section 3, in this section we revisit the four signal processing setups considered in Section 2.4—Fourier analysis and embeddings, smoothing and denoising, signal interpolation, and nonlinear (graph) neural networks—and discuss how these can be extended to simplicial complexes by means of the Hodge Laplacian and associated boundary maps. For concreteness, we concentrate primarily on edge signals, though the results presented here can be extended to signals on any type of simplices.

4.1. Fourier analysis: edge-flow and trajectory embeddings

In the same way that the (normalized) graph Laplacian provides a node embedding of the graph, the eigenvectors of the Hodge Laplacian $\mathbf{L}_1$ can be used to induce a low-frequency edge embedding. As a concrete example, let us consider the harmonic embedding, i.e., the projection of an edge signal $\mathbf{f}$ into the harmonic subspace, corresponding to signal with zero frequency

$$\mathbf{f}_{\mathrm{emb}} = \mathbf{U}_{\mathrm{harm}}^\top \mathbf{f}, \tag{12}$$

where $\mathbf{U}_{\mathrm{harm}} = [\mathbf{u}_{\mathrm{harm}}^{(1)}, \mathbf{u}_{\mathrm{harm}}^{(2)}, \ldots]$ corresponds to eigenvectors of the Hodge Laplacian $\mathbf{L}_1$ associated with zero eigenvalues. As explained in Section 3, the harmonic space spanned by the vectors $\mathbf{U}_{\mathrm{harm}}$ corresponds to (globally) cyclic flows that cannot be composed from locally cyclic flows (curl flows). Analogously to the embedding of nodes via indicator signals projected onto the low frequency eigenvectors (i.e., eigenvectors associated with low eigenvalues) of the graph Laplacian, we can construct embeddings of individual edges using (12). Unlike for graphs where such node embeddings can indicate a clustering of the nodes [57], an edge embedding into the harmonic subspace characterizes the position

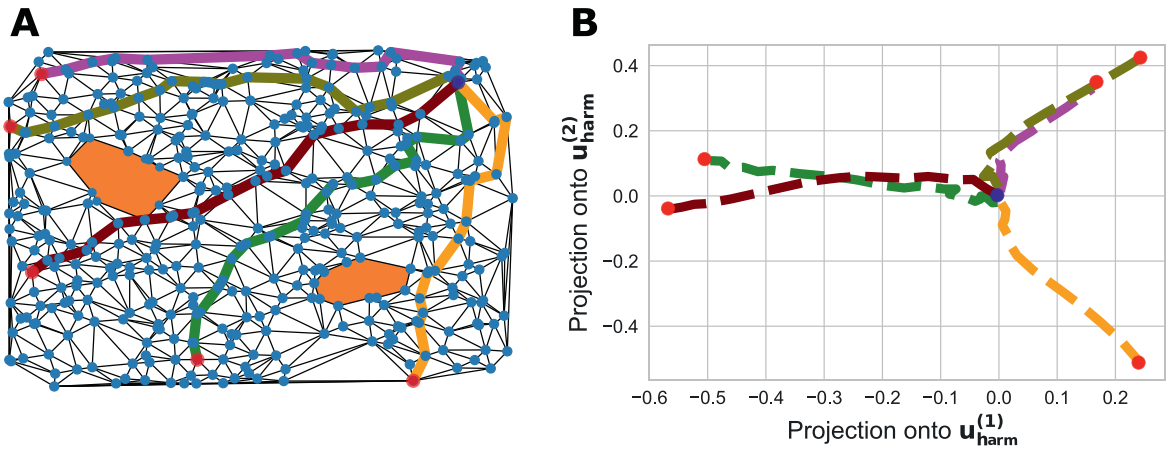

Fig. 4. Embedding of trajectories defined on a simplicial complex. A Five trajectories defined on a simplicial complex containing two obstacles, indicated by orange color. The simplicial complex is constructed by creating a triangular lattice from a random set of points and then introducing two "holes" in this lattice. All triangles in the lattices are assumed to correspond to 2-simplices. **B** The projection of the trajectories displayed in **A** into the two dimensional harmonic space of the simplicial complex. Notice that the trajectories that move around the obstacles in a topologically similar way have a similar embedding [49].

of an edge relative to the harmonic flows. Since the harmonic flows are in one-to-one correspondence with the 1-homology of the simplicial complex, i.e., the "holes" in the complex that are not filled with faces, such an embedding may be used to identify edges whose location is in accordance with particular harmonic cycles [49,58]. However, as the edges are equipped with an arbitrary reference orientation, the sign of the projection into the harmonic space is arbitrary. This is a consequence of the fact that, unlike the graph Laplacian, the Hodge Laplacian is in general not invariant, but equivariant under a change of the reference orientation of the edges (cf. section 4.4). To account for this fact, one may use a clustering approach that is invariant to this arbitrary choice of sign. For instance, we can use subspace clustering as in [58], or consider the absolute value of the projection as discussed in [49].

Rather than aiming at grouping edges together into clusters according to their relative position with respect to the 1-homology [58], we may be interested in grouping sequences of edges corresponding to trajectories on a simplicial complex by projecting appropriate signal indicator vectors of such trajectories into the harmonic space [49]. Here we represent a trajectory by a vector $\mathbf{f}$ with entries $\mathbf{f}_{(i,j)} = 1$ if the edge (i, j) is part of the trajectory and traversed along the chosen reference orientation, $\mathbf{f}_{(i,j)} = -1$ if the edge (i, j) is part of the trajectory and traversed opposite to the chosen reference orientation, and $\mathbf{f}_{(i,j)} = 0$ otherwise.

Example 5. In Fig. 4A, we construct a simplicial complex by drawing 400 random points in the unit square and generating a triangular lattice by Delaunay triangulation. We eliminate two points and all their adjacent edges in order to create two "holes" in the simplicial complex, which are not covered by a 2-simplex. These two holes are represented by orange shaded areas and can be interpreted as obstacles through which trajectories cannot pass. All (other) triangles are considered as 2-simplices. Accordingly, the Hodge Laplacian has two zero eigenvalues associated to two harmonic functions $\mathbf{u}^{(1)}_{\text{harm}}$ and $\mathbf{u}^{(2)}_{\text{harm}}$.

On the edges of the simplicial complex, we define five trajectories as displayed in Fig. 4A. Fig. 4B shows the corresponding embeddings of the flow vectors of each trajectory and their evolution in the embedding space. More explicitly, for a given trajectory we build the embedding sequentially as follows. The embedding starts at zero. We then iteratively project the next edge in the trajectory (accounting for the chosen reference direction) into the harmonic space. In our case each edge is described by a position (u_1, u_2) in the harmonic space: one component along $\mathbf{u}^{(1)}_{\text{harm}}$ and the other along $\mathbf{u}^{(2)}_{\text{harm}}$. The embedding of the trajectory is then obtained from adding these position vectors of the individual edges. Note that due to the linearity of the projection operation, this leads to the same final embedding (marked by a red dot) as if we had directly projected the full trajectory vector.

Importantly, the embedding differentiates the topological properties of the trajectories. The magenta and olive green trajectories have a similar embedding since they both pass above the top left obstacle. The maroon and green trajectories pass between the two obstacle and have a similar embedding (negative coordinate along $\mathbf{u}^{(1)}_{\text{harm}}$ and zero component along $\mathbf{u}^{(2)}_{\text{harm}}$). The orange trajectory is the only one that goes through the right of the bottom right obstacle. Hence, its embedding stands out from the other four trajectories in the embedding space. For a more extensive discussion of these aspects see [49].

As we have seen in the above example, trajectories that behave similarly with respect to the 1-homology ("holes") of a simplicial complex will have a similar embedding [49]. One may thus, for instance, also identify topologically similar trajectories on the simplicial complex by clustering the resulting points in the harmonic embedding. Such an approach is of interest for a number of applications: One can construct simplicial complexes and appropriate trajectory embedding from a variety of flow data, including physical flows such as buoys drifting in the ocean [49], or "virtual" flows such as click streams or flows of goods and money. Related ideas for analyzing trajectories have also been considered in the context of traffic prediction [59].

While we have considered here only harmonic embeddings corresponding to signals with zero frequency, other type of embeddings may be of interest as well. We may, for instance, be interested in gradient-flow-based embeddings, which can be used to define a form of ranking of the nodes in terms of the associated potentials [60], or be interested in other forms of flows, which are only approximately harmonic [55].

4.2. Flow smoothing and denoising

We now revisit the question of smoothing and denoising from the perspective of signals defined in the edge space of a simplicial complex $\mathcal{X}$. In parallel, we provide a more in-depth discussion on the basis vectors and notion of a smooth signal encapsulated in the Hodge 1-Laplacian $\mathbf{L}_1$ and how it differs from the graph Laplacian [9,48,61].

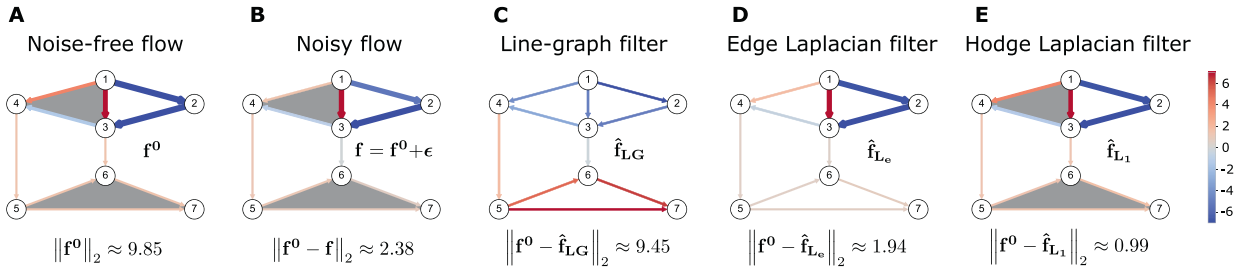

Fig. 5. Flow smoothing on a graph. **A** An undirected graph with a pre-defined and oriented flow $\mathbf{f}^0$. **B** The observed flow is a noisy version of the flow $\mathbf{f}^0$, i.e., $\mathbf{f}^0$ is distorted by a Gaussian white noise vector $\boldsymbol{\epsilon}$. **C** We denoise the flow by applying a Laplacian filter based on the line-graph. This filter performs worse compared to the edge space filters in **D** and **E** that account for flow conservation. **D** Denoised flow obtained after applying the filter based on the edge Laplacian. **E** Denoised flow obtained after applying the filter based on the Hodge Laplacian. The estimation error is lower than in the edge Laplacian case as the filter accounts for filled faces in the graph.

Let us assume that the simplicial complex $\mathcal{X}$ is associated with oriented flows $\mathbf{f}^0 \in \mathbb{R}^E$ defined on edges. Like in the node-based setup discussed in Section 2.4.2, we assume that we can only observe a noisy version $\mathbf{f} = \mathbf{f}^0 + \boldsymbol{\epsilon}$ of the true underlying signal, where $\boldsymbol{\epsilon}$ is again a zero-mean white Gaussian noise vector of appropriate dimension. By analogy with the graph case, in order to get a smooth estimate $\hat{\mathbf{f}}$ of the true signal $\mathbf{f}^0$ from the noisy signal $\mathbf{f}$, it is tempting to adopt the successful procedures from GSP (cf. Eq. (2)) and solve the following optimization program for the edge-flows $\mathbf{f}$

$$\min_{\hat{\mathbf{f}}} \left\{ \left\| \hat{\mathbf{f}} - \mathbf{f} \right\|_2^2 + \alpha \hat{\mathbf{f}}^\top \mathbf{Q} \hat{\mathbf{f}} \right\}, \tag{13}$$

with optimal solution $\hat{\mathbf{f}} = \mathbf{H}_Q \mathbf{f} := (\mathbf{I} + \alpha \mathbf{Q})^{-1} \mathbf{f}$, where the matrix $\mathbf{Q}$ is a regularizer that needs to be chosen to ensure a smooth estimate. Following our discussion above, since the filter $\mathbf{H}_Q$ will inherit the eigenvectors of the regularizer $\mathbf{Q}$, a natural choice for a regularizer is an appropriate (simplicial) shift operator.

We discuss three possible choices for the regularizer (shift operator) $\mathbf{Q}$: (i) the graph Laplacian $\mathbf{L}_{\mathrm{LG}}$ of the line-graph of the underlying graph skeleton of the complex $\mathcal{X}$, i.e., the line-graph of the graph induced by the 0-simplices (nodes) and 1-simplices (edges) of $\mathcal{X}$; (ii) the edge Laplacian $\mathbf{L}_e = \mathbf{B}_1^\top \mathbf{B}_1$, i.e., a form of the Hodge Laplacian that ignores all 2-simplices in the complex $\mathcal{X}$ such that $\mathbf{B}_2 = 0$; (iii) the Hodge Laplacian $\mathbf{L}_1 = \mathbf{B}_1^\top \mathbf{B}_1 + \mathbf{B}_2 \mathbf{B}_2^\top$ that takes into account all the triangles of $\mathcal{X}$ as well. Before embarking on this discussion, however, let us illustrate the effects of these choices by means of the following concrete example.

Example 6. Fig. 5A displays a conservative (cyclic) flow on a simplicial complex, i.e., all of the flow entering a particular node exits the node again. This flow is then distorted by a Gaussian noise vector $\boldsymbol{\epsilon}$ in Fig. 5B. The estimation error produced by the filter based on the line-graph (Fig. 5C) is comparatively worse (9.45 vs. 1.94 and 0.99 respectively) than the estimation performance of the edge Laplacian (Fig. 5D) and the Hodge Laplacian (Fig. 5E) filters.

Let us explain the results obtained from the individual filters in the above example in more detail, starting with the line-graph approach. As can be seen from Fig. 5C, in this case the filtering operation leads to an increased error compared to the noisy input signal. This ineffective filtering result by means of the line-graph Laplacian has been observed in [54]. The reason for this unintended behavior is that the line-graph Laplacian is not well-suited as a shift operator for flow signals. The basis functions given by the eigenvectors of the line-graph Laplacians induce a notion of smooth, low frequency signals that supposes that signals on adjacent edges in the simplicial complex have a small difference. This is equivalent to the fact that low-frequency modes in the node space do not vary a lot on tightly connected nodes on a graph. However, for flow signals this type of smoothness induced by eigenvectors of the line-graph Laplacian as shift operator is often not appropriate. Specifically, real-world flow signals typically display a large degree of flow conservation: most of the flow signal entering a node exits the node again, but the relative allocation of the flow to the edges does not have to be similar. Moreover, the line-graph Laplacian does not reflect the arbitrary orientation of the edges, so that performance is dependent on the chosen sign of the flow. Notice, however, that the line-graph *can* be a valid representation to process signals on edges that are not encoding flows and, as such, do not have a natural orientation. For example, one might expect the level of congestion on different roads to vary smoothly across edges, thus, justifying the use of a line-graph in such a case.

Unlike the line-graph Laplacian, the Edge Laplacian captures a notion of flow conservation, which implies that smooth flows should by cyclic [54]. To see this, it is insightful to inspect the quadratic regularizer induced by $\mathbf{L}_e = \mathbf{B}_1^\top \mathbf{B}_1$. Note that this quadratic form can be written as $\mathbf{f}^\top \mathbf{L}_e \mathbf{f} = \|\mathbf{B}_1 \mathbf{f}\|_2^2$. This is precisely the (summed) squared divergence of the flow signal $\mathbf{f}$, as each entry $(\mathbf{B}_1 \mathbf{f})_i$ corresponds to the difference of the inflow and outflow to node i

$$(\mathbf{B}_1 \mathbf{f})_i = \sum_{r \in \{(j,i) | \{i,j\} \in \mathcal{E}, j < i\}} f_r - \sum_{r \in \{(i,j) | \{i,j\} \in \mathcal{E}, i < j\}} f_r, \tag{14}$$

where f_r is the flow on edge $r = (i, j)$, and we have used a reference orientation induced by the lexicographic order. As a consequence, all cyclic flows will induce zero cost for the regularizer $\mathbf{f}^\top \mathbf{L}_e \mathbf{f}$, which may also be seen from the fact that $\ker(\mathbf{B}_1)$ is precisely the cycle space of a graph with incidence matrix $\mathbf{B}_1$. Stated differently, any flow that is *not* divergence free, i.e., not cyclic, will be penalized by the quadratic form. Since by the fundamental theorem of linear algebra $\ker(\mathbf{B}_1) \perp \mathrm{im}(\mathbf{B}_1^\top)$, any such non-cyclic flows can be written as a gradient flow $\mathbf{f}_{\mathrm{grad}} = \mathbf{B}_1^\top \mathbf{v}$ for some vector $\mathbf{v}$ of scalar node potentials — in line with the Hodge decomposition discussed in (9).

In contrast to the Edge Laplacian, the full Hodge Laplacian $\mathbf{L}_1$ includes the additional regularization term $\mathbf{f}^\top \mathbf{B}_2 \mathbf{B}_2^\top \mathbf{f} = \|\mathbf{B}_2^\top \mathbf{f}\|_2^2$, which may induce a non-zero cost even for certain cyclic flows. More precisely, any cyclic flow that can be written as a curl flow $\mathbf{f}_{\mathrm{curl}} = \mathbf{B}_2 \mathbf{t}$, for some vector $\mathbf{t}$ will have a non-zero penalty. This penalty is incurred despite the fact that $\mathbf{f}_{\mathrm{curl}}$ is a cyclic flow by construction (since $\mathbf{B}_1 \mathbf{f}_{\mathrm{curl}} = \mathbf{B}_1 \mathbf{B}_2 \mathbf{c} = 0$, the vector $\mathbf{f}_{\mathrm{curl}}$ is clearly in the cycle space; see also discussion in Section 3.2). The additional regularization term $\|\mathbf{B}_2^\top \mathbf{f}\|_2^2$ may thus be interpreted as squared curl penalty.

From a signal processing perspective, the $\mathbf{L}_1$ based filter thus allows for a more refined notion of a smooth signal. Unlike in the Edge Laplacian filter, we do not declare all cyclic flows to be maximally smooth and consist only of frequency (eigenvalue) 0 basis signals. Instead a signal can have a high-frequency even if it is cyclic, if it has a high curl component. Hence, by constructing simplicial complexes with appropriate (triangular) 2-simplices, we

have additional modeling flexibility for shaping the frequency response of an edge-flow filter [62].

In our example above, this more refined notion of a smooth signal is precisely what leads to an improvement in the filtering performance, since the ground truth signal is a harmonic function with respect to the simplicial complex and thus does not contain any curl components. We remark that the eigenvector basis of $\mathbf{L}_e$ can always be chosen to be identical to the eigenvectors of $\mathbf{L}_1$; thus, we may represent any signal in exactly the same way in a basis of $\mathbf{L}_e$ or $\mathbf{L}_1$; however, the frequencies associated with all cyclic vectors will be 0 for the Edge Laplacian, while there will be cyclic flows with nonzero frequencies for $\mathbf{L}_1$, in general. This emphasizes that the construction of faces is an important modeling choice for the selection of an appropriate notion of a smooth signal.

4.3. Interpolation and semi-supervised learning

Let us now focus on the interpolation problem for edge-data on a simplicial complex [55]. Analogously to node signals, we are given a simplicial complex (or its graph skeleton) and a set of "labeled" oriented edges $\mathcal{E}^L \subset \mathcal{E}$, i.e., we assume that we have measured the edge-signals on some edges but not on all. Our goal is now to predict the signals on the unlabeled or unmeasured edges in the set $\mathcal{E}^U \equiv \mathcal{E}\backslash\mathcal{E}^L$, whose cardinality we will denote by E_U. Following [55], we will again start by considering the problem setup with no 2-simplices first ($\mathbf{B}_2 = 0$), before we consider the general case in which 2-simplices are present.

To arrive at a well-defined problem for imputing the remaining edge-flows, we need to make an assumption about the structure of the true signal. Following our above discussions, we will again assume that the true signal has a low-pass characteristic in the sense of the Hodge 1-Laplacian, i.e., that the edge flows are mostly conserved. Let $\hat{\mathbf{f}}$ denote the vector of the true (partly measured) edge-flow. As discussed in the context of flow smoothing, a convenient loss function to promote flow conservation is the sum-of-squares vertex divergence

$$\left\|\mathbf{B}_1\hat{\mathbf{f}}\right\|_2^2 = \hat{\mathbf{f}}^\top\mathbf{B}_1^\top\mathbf{B}_1\hat{\mathbf{f}} = \hat{\mathbf{f}}^\top\mathbf{L}_e\hat{\mathbf{f}}. \tag{15}$$

We can then formalize the flow interpolation problem via the following optimization program

$$\begin{aligned}&\min_{\hat{\mathbf{f}}} \left\|\mathbf{B}_1\hat{\mathbf{f}}\right\|_2^2 + \alpha^2\cdot\left\|\hat{\mathbf{f}}\right\|_2^2\\ &\text{s.t. } \hat{f}_r = f_r, \text{ for all measured edges } r\in\mathcal{E}^L,\end{aligned} \tag{16}$$

Note that, in contrast to the node signal interpolation problem, we have to add an additional regularization term $\|\hat{\mathbf{f}}\|_2^2$ to guarantee the uniqueness of the optimal solution. The reason is that, if there is more than one independent cycle in the network for which we have no measurement available, we may add any cyclic flow on such a cycle while not changing the cost function. To remedy this aspect, we simply add a 2-norm regularization which promotes small edge-flow magnitudes by default. Other regularization terms are possible as well, however this formulation enables us to rewrite the above problem in a least squares form as described below.

To arrive at a least-squares formulation, we consider a trivial feasible solution $\hat{\mathbf{f}}^0$ for (16) that satisfies $\hat{f}_r^0 = f_r$ if $r\in\mathcal{E}^L$ and $\hat{f}_r^0 = 0$ otherwise. Let us now define the expansion operator $\mathbf{\Phi}$ as the linear map from $\mathbb{R}^{E_U}$ to $\mathbb{R}^E$ such that the true flow $\mathbf{f}$ can be written as $\mathbf{f} = \hat{\mathbf{f}}^0 + \mathbf{\Phi}\mathbf{f}^U$, where $\mathbf{f}^U\in\mathbb{R}^{E_U}$ is the vector of the unmeasured true edge-flows. Reducing the number of variables considered in this way, we can convert the constrained optimization problem (16) into the following equivalent unconstrained least-squares estimation problem for the *unmeasured* edges $\hat{\mathbf{f}}^U$:

$$\hat{\mathbf{f}}^{U*} = arg\,min_{\hat{\mathbf{f}}}^U \left\|\begin{bmatrix}\mathbf{B}_1\mathbf{\Phi}\\ \alpha\mathbf{I}\end{bmatrix}\hat{\mathbf{f}}^U - \begin{bmatrix}-\mathbf{B}_1\mathbf{f}^0\\ 0\end{bmatrix}\right\|_2^2. \tag{17}$$

We illustrate the above procedure by the following example.

Example 7. We consider the network structure in Fig. 2A. The ground truth signal is $\mathbf{f} = (-2,-2,4,-2,3,-7,7,3,4,-4)^\top$. We pick five labeled edges at random (colored in Fig. 6A). The goal is to predict the labels of the unlabeled edges (in grey with a question mark in Fig. 6A). The set of labeled edges is $\mathcal{E}^L = \{(1,3),(1,4),(3,6),(4,5),(5,6)\}$. The set of unlabeled edges is $\mathcal{E}^U = \{(1,2),(2,3),(3,4),(5,7),(6,7)\}$. Solving the optimization program (17), we obtain the predicted signal $\mathbf{f}_{SSL}^*$ in Fig. 6B. Numerical values are given in Fig. 6C. The Pearson correlation coefficient between $\mathbf{f}$ and $\mathbf{f}_{SSL}^*$ is 0.99. The 2-norm of the error is 0.064.

Analogously to our discussion above, it may be relevant to include 2-simplices for the signal interpolation problem. We interpret such an inclusion of 2-simplices in two ways. From the point of view of the cost function, it implies that instead of penalizing primarily gradient flows (which have nonzero divergence), we in addition penalize certain cyclic flows, namely those that have a nonzero curl component. From a signal processing point of view, it means that we are changing what we consider a smooth (low-pass) signal, by adjusting the frequency representation of certain flows. Accordingly, one possible formulation of the signal interpolation problem, including information about 2 simplices is

$$\hat{\mathbf{f}}^\star = arg\,min_{\hat{\mathbf{f}}}\left\|\mathbf{B}_1\hat{\mathbf{f}}\right\|_2^2 + \left\|\mathbf{B}_2^\top\hat{\mathbf{f}}\right\|_2^2 + \alpha^2\left\|\hat{\mathbf{f}}\right\|_2^2, \tag{18}$$

subject to the constraint that the components of $\hat{\mathbf{f}}$ corresponding to measured flows are identical to those measurements. As in (17), we can convert this program into the following least-squares problem

$$\hat{\mathbf{f}}^{U\star} = arg\,min_{\hat{\mathbf{f}}^U}\left\|\begin{bmatrix}\mathbf{B}_1\mathbf{\Phi}\\ \alpha\mathbf{I}\\ \mathbf{B}_2^\top\mathbf{\Phi}\end{bmatrix}\hat{\mathbf{f}}^U - \begin{bmatrix}-\mathbf{B}_1\mathbf{f}^0\\ 0\\ -\mathbf{B}_2^\top\mathbf{f}^0\end{bmatrix}\right\|_2^2. \tag{19}$$

Remark 8. Note that the problem of flow interpolation is tightly coupled to the issue of signal reconstruction from sampled measurements. Indeed, if we knew that the edge signal to be recovered was exactly bandlimited [56], then we could reconstruct the edge-signal if we had chosen the edges to be sampled appropriately. Just like the interpolation problem considered here may be seen as a semi-supervised learning problem for edge labels, finding and choosing such optimal edges to be sampled may be seen as an active learning problem in the context of machine learning. While we do not expand further in this tutorial on the choice of edges to be sampled, we point the interested reader to two heuristic active learning algorithms for edge flows presented in [55]. We also refer the reader to [56,61] for a theory of sampling and reconstruction of bandlimited signals on simplicial complexes, and to [63] for a similar overview that includes an approach for *topology inference* based on signals supported on simplicial complexes.

4.4. Beyond linear filters: simplicial neural networks and Hodge theory

As discussed in Section 2.4.4, graph neural networks incorporate nonlinear activation functions in the graph signal processing pipeline in order to learn rich representations for graphs. In order to generalize these architectures to operate on simplicial complexes, we discuss central concepts underpinning graph neural network architectures in order to understand desirable properties of neural networks for higher-order data. Graph neural networks in the nodal domain typically have two important features in common:

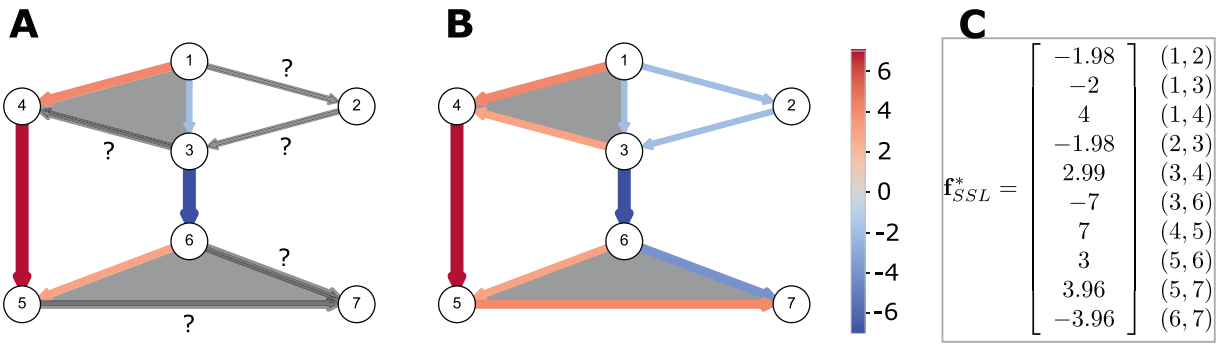

Fig. 6. Semi-supervised learning for edge flow. A Synthetic flow. 50% of the edges are labeled. Labeled edges are colored based on the value of their flow. The remaining edges in grey are inferred from the procedure explained in the text. **B** Edge flow obtained after applying the semi-supervised algorithm in (17). **C** Numerical value of the inferred signal.

Permutation equivariance. Although the nodes are given labels and an ordering for notational convenience, graph neural networks are not dependent on the chosen labeling of the nodes. That is, if the node and corresponding input labels were permuted in some way, the output of the graph neural network, modulo said permutation, will not change.

Locality. Graph neural networks in their most basic form operate *locally* in the graph structure. Typically, at each layer a node's representation is affected only by its own state and the state of its immediate neighbors. Forcing operations to occur locally is how the underlying graph structure is used to regularize the functional form of the graph neural network.

Based on these two principles, many architectures have been proposed, such as the popular *graph convolutional network* [45], which mixes one-step graph filters and nodewise nonlinearities for semi-supervised learning on the nodes of a graph. Indeed, there has been significant study in understanding the nature of graph convolutional architectures in terms of the spectral properties of the chosen shift or filter operation [64].

4.4.1. Simplicial graph neural networks

Motivated by work on graph neural networks in the node space, and the effectiveness of the Hodge Laplacian for representing certain types of data supported on simplicial complexes as in Section 3, we now discuss considerations and limitations for building graph neural network architectures based on representations grounded in combinatorial Hodge theory. This approach to processing data on simplicial complexes generated a flurry of interest recently, with convolutional architectures based on the Hodge Laplacians and boundary maps being proposed in [65–68]. As before, let $\mathcal{X}$ be a simplicial complex over a finite set of vertices $\mathcal{V}$, with boundary operators $\{\mathbf{B}_k\}_{k=1}^K$, where K is the order of $\mathcal{X}$.

We consider architectures built on the composition of matrix multiplication with boundary operators and/or Hodge Laplacians of varying order, aggregation functions, and nonlinear activation functions that obey *permutation invariance, locality*, and the additional properties of *orientation invariance* and *simplicial locality*.

We begin by defining orientation equivariance, which describes a similar property to permutation invariance for graph neural networks [69].

Orientation equivariance. If the chosen arbitrary reference orientation of the simplices in $\mathcal{X}$ is changed, the output of the neural network architecture remains the same, modulo said change in orientation.

Due to the arbitrary nature of the simplex orientations, orientation invariance is clearly a desirable property for a neural network architecture to have. For a simple class of convolutional neural networks for flows, we must choose the nonlinear activation function carefully in order to satisfy this property. If one were to construct a simple architecture with weight matrices $\mathbf{W}_1, \mathbf{W}_2$ for flows on a simplicial complex based on $\mathbf{L}_1$ of the form

$$g_{\mathbf{L}_1,\mathbf{W}}(\mathbf{f}) = \sigma(\mathbf{L}_1\sigma(\mathbf{L}_1\mathbf{f}\mathbf{W}_1)\mathbf{W}_2), \tag{20}$$

we want g to not change when a different orientation is chosen. Let $\mathbf{\Theta} \in \mathbb{R}^{E\times E}$ be a matrix taking values ± 1 on the diagonal and zeros elsewhere, representing a change in orientation for each edge. Then, for a flow $\mathbf{f}$ and Hodge Laplacian $\mathbf{L}_1$, this change in orientation is realized by $\mathbf{\Theta}\mathbf{f}$ and $\mathbf{\Theta}\mathbf{L}_1\mathbf{\Theta}$. Therefore, for orientation equivariance to hold, we need

$$g_{\mathbf{\Theta}\mathbf{L}_1\mathbf{\Theta},\mathbf{W}}(\mathbf{\Theta}\mathbf{f}) = \mathbf{\Theta}g_{\mathbf{L}_1,\mathbf{W}}(\mathbf{f}) \tag{21}$$

to hold for all flows $\mathbf{f}$. For this to be true, σ must be an odd function so that it commutes with $\mathbf{\Theta}$. A natural extension to the notion of orientation equivariance is *orientation invariance*, which rewrites (21) as

$$g_{\mathbf{\Theta}\mathbf{L}_1\mathbf{\Theta},\mathbf{W}}(\mathbf{\Theta}\mathbf{f}) = g_{\mathbf{L}_1,\mathbf{W}}(\mathbf{f}). \tag{22}$$

This property has greater utility for tasks such as graph classification, where a global descriptor is desired, rather than output on each simplex.

Another consideration that does not typically arise in the design of graph neural networks is data supported on different levels of the graph. Data on a simplicial complex can lie on, e.g., nodes, edges, and faces *simultaneously*, motivating the need for architectures that pass data along the many levels of a simplicial complex. Analogous to the property of locality for graph neural networks, we consider a notion of locality for different levels of a simplicial complex.

Simplicial locality. At each layer of an architecture with simplicial locality, information exchange only occurs between adjacent levels of the underlying simplicial complex, i.e., the output of a layer restricted to $k-$simplices is dependent only on the input of that layer restricted to $k-1, k, k+1-$simplices.

As an illustrative example, loosely based on the architecture proposed in [66], consider a small two-layer neural network simultaneously operating over a simplicial complex of nodes, edges, and triangles. That is, the input to the neural network is a tuple of signals $(\mathbf{v}_0, \mathbf{f}_0, \mathbf{t}_0)$ on the vertices (graph signals), edges (flows), and triangles, respectively, and each layer performs the following computation:

$$\mathbf{v}_k = \sigma(\mathbf{L}_0\mathbf{v}_{k-1} + \mathbf{B}_1\mathbf{f}_{k-1}) \tag{23}$$

$$\mathbf{f}_k = \sigma(\mathbf{L}_1\mathbf{f}_{k-1} + \mathbf{B}_2\mathbf{t}_{k-1} + \mathbf{B}_1^\top\mathbf{v}_{k-1}) \tag{24}$$

$$\mathbf{t}_k = \sigma\left(\mathbf{L}_2\mathbf{t}_{k-1} + \mathbf{B}_2^\top \mathbf{f}_{k-1}\right), \tag{25}$$

for some odd elementwise activation function σ. That is, at each layer, signals on each level of the simplicial complex are either lifted to the next highest level via the coboundary operator, projected to its boundary using the boundary operator, or diffused via the Hodge Laplacian. This "lifting" and "projecting" can only occur between adjacent levels of the simplicial complex, due to the fact that the composition of boundary operators is null, *thereby satisfying simplicial locality.*

We now examine the tuple of signals $(\mathbf{v}_2, \mathbf{f}_2, \mathbf{t}_2)$. First, suppose σ is the identity mapping, so that each signal in $(\mathbf{v}_2, \mathbf{f}_2, \mathbf{t}_2)$ is a linear function of $(\mathbf{v}_0, \mathbf{f}_0, \mathbf{t}_0)$. Then, one can check that

$$\mathbf{v}_2 = 2\mathbf{L}_0\mathbf{B}_1\mathbf{f}_0 + \mathbf{L}_0(\mathbf{L}_0 + \mathbf{I})\mathbf{v}_0 \tag{26}$$

$$\mathbf{f}_2 = (\mathbf{L}_1^2\mathbf{B}_2 + \mathbf{B}_2\mathbf{L}_2)\mathbf{t}_0 + \mathbf{L}_1(\mathbf{L}_1 + \mathbf{I})\mathbf{f}_0 + (\mathbf{L}_1^2 + \mathbf{B}_1^\top\mathbf{B}_1)\mathbf{B}_1^\top\mathbf{v}_0 \tag{27}$$

$$\mathbf{t}_2 = \mathbf{L}_2(\mathbf{L}_2 + \mathbf{I})\mathbf{t}_0 + (\mathbf{L}_2\mathbf{B}_2^\top + \mathbf{B}_2^\top\mathbf{L}_1)\mathbf{f}_0. \tag{28}$$

Notice that each signal is strictly a function of the signals above and below it, even after multiple layers of the architecture are evaluated. This indicates that our architecture is incapable of incorporating information from nonadjacent levels of the simplicial complex, due to the composition of boundary operators being null: note that similar properties hold for linear variants of this example making use of boundary operators in this way.

This is not the case, though, when σ is nonlinear. While $\mathbf{B}_1\mathbf{B}_2\mathbf{t} = 0$ may hold for all signals $\mathbf{t}$ on the faces, $\mathbf{B}_1\sigma(\mathbf{B}_2\mathbf{t}) \neq 0$, in general. By incorporating nonlinear activation functions, we facilitate full incorporation of signals from all levels of the simplicial complex in the output at each level. We call this property *extended simplicial locality*.

> *Extended simplicial locality.* For an architecture with extended simplicial locality, the output restricted to $k-$simplices is dependent on the input restricted to simplices at all levels, not just those of order $k-1, k, k+1$.

Notice that while simplicial locality is defined for each layer of an architecture, extended simplicial locality is a global property, so that both are simultaneously attainable. There is a trade-off in achieving extended simplicial locality by interleaving nonlinearities: although there is full influence of the entire simplicial structure on all levels of the output, the structure endowed by the boundary operators (namely, the composition of boundary operators being null) is no longer in effect. Although the Hodge decomposition (9) can still be applied to the output signals of such an architecture, the expression of the space of k-simplex signals strictly in terms of upper and lower incidence through $k-1$ and $k+1$ simplices ceases to hold when considering the input and output jointly, as opposed to linear filters of the Hodge Laplacian. This motivates further considerations of how nonlinearities may be necessary in modeling higher-order data, such as in the work of [70,71], where it is shown that higher-order opinion dynamics *must* be nonlinear, lest they be equivalently modeled by a purely pairwise system. That is, we must relax the structure of simplicial complexes in order to represent more general high-order interactions. In doing this, we exchange the connection to algebraic topology for greater flexibility in modeling. This naturally leads to the consideration of hypergraphs and associated signal processing ideas, as discussed in the next section.

5. Modeling higher-order interactions via hypergraphs

In this section, we discuss *hypergraphs* as an alternative to simplicial complexes to model higher-order analogs of graphs, and then discuss how we can construct appropriate matrix-based and tensor-based shift operators for such hypergraphs to enable the development of signal processing tools.

An important feature of simplicial complexes is that for every simplex present all of its faces are also included in the complex (and recursively the corresponding faces, and so on). This inclusion property gives rise to the hierarchy of boundary operators, which anchor simplicial complexes to algebraic topology. However, this subset inclusion property may be an undesirable restriction, if we want to represent interactions that are exclusive to multiple nodes and do not imply the interaction between all the subsets of nodes. A related problem is the issue of (extended) simplicial locality as discussed in the previous section, which arises from the restrictions imposed on the boundary operators of simplicial complexes. Finally, while simplices are endowed with a reference orientation and may be weighted, we might be interested in encoding other types of directionality or heterogeneous weighting schemes of group interactions, which are not easily compatible with the mathematical structure of simplicial complexes.

To illustrate the utility of hypergraphs as modelling tools, let us consider a number of concrete examples in which a hypergraph model may be preferred over a simplicial complex, before providing a more mathematical definition.

Example 9. In a co-authorship network [72], having a paper with three or more authors does not imply that these people have written papers in pairs. Hypergraphs can distinguish these two cases while graphs and simplicial complexes cannot, in general. Moreover, the relative contribution of the authors to a paper may be different and we thus may want to have a representation that enables us to assign heterogeneous weights within group interactions. This again can be done using hypergraphs [73]. An email network may be described using a directed hypergraph [74], whenever there exist emails containing multiple senders or multiple receivers. This kind of directional information will be difficult to encode in a simplicial complex (while graphs can encode the directionality here, they lose the higher-order information). Further examples in which hypergraphs appear naturally include word-document networks in text mining [75,76], gene-disease networks in bioinformatics [77,78], and consumer-product networks in e-commerce [79].

Mathematically, a typical hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \omega)$ consists of a set of vertices $\mathcal{V}$, a set of hyperedges $\mathcal{E}$, and a function $\omega : \mathcal{E} \to \mathbb{R}_+$ that assigns positive weights to hyperedges. Hyperedges generalize edges in the sense that each hyperedge can connect more than two vertices. In the most common case, where there is one type of node and one type of hyperedge (namely all hyperedges representing the same type of relationship such as co-authorship), a hypergraph is called homogeneous. A hypergraph is called k-uniform if all of its hyperedges have the same cardinality k. Notice, in particular, that a hypergraph is a bona fide generalization of a graph, since a 2-uniform hypergraph reduces to a graph. More interestingly, a simplicial complex may be seen as a hypergraph satisfying the property that every subset of a hyperedge is also a hyperedge. Similar to a standard graph, a hypergraph can also be directed in which case each (directed) hyperedge e is an ordered pair $(T(e), H(e))$ where $T(e)$ and $H(e)$ are two disjoint subsets of vertices respectively called the tail and the head of e [80]. This flexibility is of interest, e.g., when modelling multiway communication patterns as illustrated in the example of email networks above.

While the standard framework of hypergraphs is already very flexible, in recent years several more elaborate hypergraph models have been proposed to better represent real-world datasets:

(1) Heterogeneous hypergraphs refer to hypergraphs containing different types of vertices and/or different types of hyper-

edges [81–84] and may thus be seen as a generalization of multilayer and multiplex networks. For example, in a GPS network [85], a hyperedge can have three types of vertices (user, location, activity). Another example is online social networks such as Twitter, in which we can have different types of vertices including users, tweets, usertags, hashtags and groups as well as multiple types of hyperedges such as 'users release tweets containing hashtags or not', 'users join groups', and 'users assign usertags to themselves' [86].

(2) Edge-dependent vertex weights are introduced into hypergraphs in [73,75,76] to reflect the different contribution (e.g., importance or influence) of vertices in the same hyperedge. More precisely, for each hyperedge $e \in \mathcal{E}$, a function $\gamma_e : e \to \mathbb{R}_+$ is defined to assign positive weights to vertices in this hyperedge. For instance, in the co-authorship network in Example 9, the different levels of contribution of the authors of a paper can be encoded as edge-dependent vertex weights. If $\gamma_e(v) = \gamma_{e'}(v)$ for every vertex v and every pair of hyperedges e and e' containing v, then we say that the vertex weights are edge-independent. Such hypergraphs are also called vertex-weighted hypergraphs [87]. Moreover, if $\gamma_e(v) = 1$ for all vertices v and incident hyperedges e, the vertex weights are trivial and we recover the homogeneous hypergraph model.

(3) In order to leverage the fact that different subsets of vertices in one hyperedge may have different structural importance, the concept of an inhomogeneous hyperedge is proposed in [88]. Each inhomogeneous hyperedge e is associated with a function $w_e : 2^e \to \mathbb{R}_{\geq} 0$ that assigns non-negative costs to different cuts of the hyperedge, where 2^e denotes the power set of e. The weight $w_e(\mathcal{S})$ indicates the cost of partitioning the hyperedge e into two subsets $\mathcal{S}$ and $e \setminus \mathcal{S}$. This is called a submodular hypergraph when w_e satisfies submodularity constraints [89].

Similar to graphs and simplicial complexes, a key factor for developing signal processing tools for hypergraphs is the definition of an appropriate shift operator. For simplicial complexes, we argued that the Hodge Laplacian is a natural and principled operator for this purpose. For hypergraphs there are two major approaches to their mathematical representation, which induce different kinds of shift operators.

The first option is to use a matrix-based representation and derive a shift operator from it, akin to the approach of GSP. As any matrix may be interpreted as an adjacency matrix of a graph and thus induces a weighted, directed graph, this procedure may be understood as first deriving a graph-based representation of the hypergraph and then using an algebraic representation of this graph (e.g., adjacency or Laplacian matrices) as the algebraic shift operator of the hypergraph.

The second option is to represent the hypergraph using a tensor, i.e., a multi-dimensional array representation instead of the 2-dimensional array representation provided by matrices (we refer to [90–92] for a general introduction to tensors and tensor decompositions). While this provides, in principle, a richer set of possible representations of the shift operator, there are also challenges associated with this procedure as the definition of a hypergraph signal and its processing is less grounded in GSP and related techniques. In the following subsections, we respectively discuss these two choices of representations, starting with matrix-based representations.

5.1. Matrix-based hypergraph representations

The most common approach to deal with hypergraph-structured data is to encode the hypergraph as a matrix. When interpreting the corresponding matrices as graphs, many of these matrix-based approaches can thus, alternatively, be viewed as deriving a graph representation for the hypergraph. Accordingly, these approaches are often described in terms of graph expansions. We prefer the term matrix representation here, as the fact that we encode a particular data structure via a matrix does not imply that the data structure is itself a graph (possibly with weights and signed edges). For instance, we studied matrix-based representations of simplicial complexes in the previous sections, but this would typically not be considered a graph expansion of a simplicial complex.

Let us now discuss some of the most common matrix-based hypergraph representations and transformations (see Fig. 7 for a visual overview of the discussed variants), including the so-called clique and star expansions as the most popular variants [93]. To this end, consider a homogeneous hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E}, \omega)$ and define the vertex-to-hyperedge incidence matrix as $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}$ with entries $Z_{ve} = 1$ if vertex v belongs to hyperedge e. In addition, we will represent the weights of the hyperedges by the diagonal matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$, whose diagonal corresponds to the hyperedge weights.

Let us first consider the so called star-graph expansion (Fig. 7D) [94,95]. Using the above defined matrices, the star-graph expansion can be explained by constructing the following adjacency matrix $\mathbf{A}_*$ of a bipartite graph

$$\mathbf{A}_* = \begin{bmatrix} \mathbf{0} & \mathbf{ZW} \\ \mathbf{WZ}^\top & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{(|\mathcal{V}|+|\mathcal{E}|)\times(|\mathcal{V}|+|\mathcal{E}|)}. \tag{29}$$

When interpreted in terms of a graph, this construction may be explained as follows: We introduce a new vertex for each hyperedge and each of these vertices is then connected with a weight corresponding to the weight of the hyperedge to all the (original) vertices in this hyperedge. The constructed weighted graph $\mathcal{G}_* = (\mathcal{V}_*, \mathcal{E}_*, \omega_*)$, thus has a vertex set $\mathcal{V}_* = \mathcal{V} \cup \mathcal{E}$, an edge set $\mathcal{E}_* = \{(v, e) : v \in e, e \in \mathcal{E}\}$, and an edge weight function $\omega_*(v, e) = \omega(e)$. Many other weight functions are possible here as well, e.g., we may normalize by the cardinality of the hyperedges. By constructing appropriate Laplacian operators (combinatorial or normalized) of such a star expansion matrix, we can thus obtain a shift-operator for the hypergraph in a straightforward fashion.

An alternative matrix-based representation that can be derived from the same matrices defined above is the clique expansion (Fig. 7C) [96–99]. In matrix terms, this corresponds to projecting out the hyperedge dimension of the incidence matrix $\mathbf{Z}$. Specifically, if we assume unit hyperedge weights for simplicity, the clique expansion may be computed by forming the product $\mathbf{ZZ}^\top$. As this matrix has a nonzero diagonal, we can simply set the diagonal of this matrix to zero to obtain a basic clique expansion matrix $\mathbf{A}_c = \mathbf{ZZ}^\top - \mathrm{Diag}(\mathrm{diag}(\mathbf{ZZ}^\top))$. By including various weighting factors, alternative variants of this matrix can be derived. The name clique expansion becomes intuitive if we again interpret $\mathbf{A}_c$ as the adjacency matrix of a graph: The above construction corresponds to replacing every hyperedge with a clique subgraph. More precisely, the clique expansion leads to the adjacency matrix of a graph $\mathcal{G}_c = (\mathcal{V}_c, \mathcal{E}_c, \omega_c)$ in which $\mathcal{V}_c = \mathcal{V}$, $\mathcal{E}_c = \{(u, v) : u, v \in e, e \in \mathcal{E}, u \neq v\}$. One of the most common definitions for the edge weighting function in this context is $\omega_c(u, v) = \sum_{e\in\mathcal{E}:u,v\in e} \omega(e)$, i.e., the edge weight in the graph is simply given by the sum of the weights of hyperedges that contain the two endpoints. However, many other weighting schemes are conceivable.

As has been shown in [93], many hypergraph learning algorithms [94–99] correspond to either the clique or star expansions with an appropriate weighting function. However, apart from these common expansions, there also exist other methods for projecting hypergraphs to graphs such as constructing a line graph [100]. This line-graph expansion for hypergraphs (see Fig. 7E for an illus-

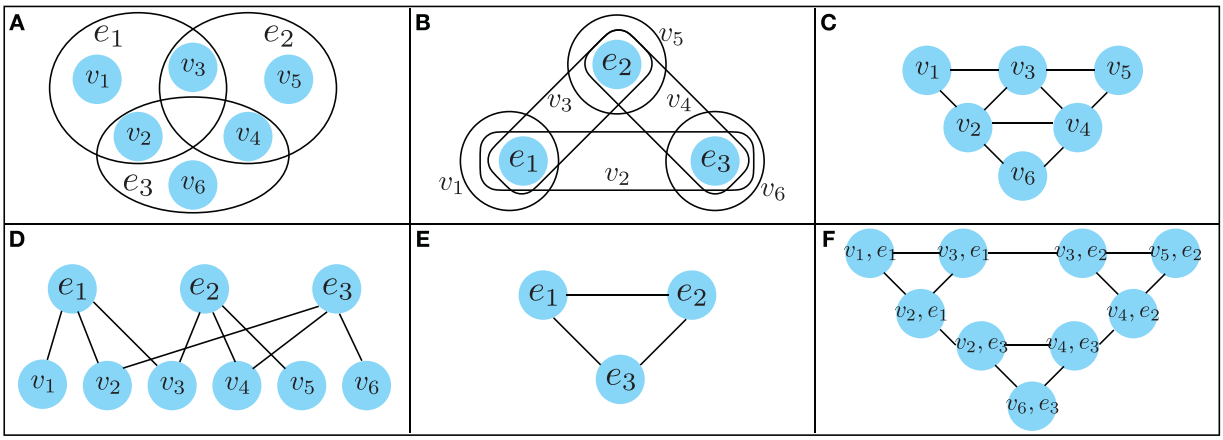

Fig. 7. Different transformations on an example hypergraph. A The original hypergraph. **B** The dual hypergraph. **C** The clique expansion. **D** The star expansion. **E** The line graph. **F** The line expansion.

tration) may be computed in terms of (weighted variants of) the second possible projection of the incidence matrix $\mathbf{Z}$, namely $\mathbf{Z}^\top\mathbf{Z}$. Apart from these three canonical types of graph representations (star, clique, and line graph) that can be derived from the incidence matrix $\mathbf{Z}$ and additional (weighting) transformations, a few other matrix-based schemes have been proposed for representing hypergraphs. For instance, the recent paper [101] proposes the so-called line expansion of a hypergraph (different from the line graph; see Fig. 7F), which is isomorphic to the line graph of its star expansion and aims to unify the clique and star expansions. In the line expansion, each incident vertex-hyperedge pair is considered as a "line node" and two "line nodes" are connected if they share either the vertex or the hyperedge. We would like to remark that in some cases we might be more interested in the dual of one hypergraph in which the roles of vertices and hyperedges are interchanged and the incidence matrix is $\mathbf{Z}^\top$ [78]; see Fig. 7B.

While we have so far considered only homogeneous hypergraphs, Laplacian matrices have also been proposed for more general hypergraph models. For instance, [73,75,88] use variants of the clique expansion to derive matrix representations of hypergraphs with edge-dependent vertex weights or inhomogeneous hyperedges. Specifically, in [73,75] hypergraphs with edge-dependent vertex weights are projected onto asymmetric matrices, corresponding to induced directed graphs with self-loops. The authors then use established combinatorial and normalized Laplacians for digraphs [102] applied to these matrices to derive a Laplacian matrix for hypergraphs. Finally, in [88], a novel algorithm for assigning edge weights to the graph representation is proposed, allowing for non-uniform expansions of hyperedges.

As the above discussion shows, there is an enormous variety of matrix-based representations for hypergraphs, and the relative advantages and disadvantages of these constructions are still sparsely understood. Ultimately, the choice of a particular matrix representation corresponds to a specific model for what constitutes a smooth signal on a hypergraph. We believe that a better understanding of the spectral properties of the individual constructions will thus be an important step for choosing good matrix representations for different application scenarios.

5.2. Tensor-based hypergraph representations

Instead of working with matrix-based representations, hypergraphs can alternatively be represented by tensors. A tensor is simply a multi-dimensional array, whose order is the number of indices needed to label an element in the tensor [90]. For instance, a vector and a matrix are a first-order and a second-order tensor, respectively. Several different versions of a hypergraph adjacency tensor have been proposed in existing work [103–112]. In this section, we focus on unweighted hypergraphs to keep our exposition accessible and to remain consistent with the majority of the existing work in this domain.

Due to their relative simplicity, k-uniform hypergraphs have been first studied in the literature. As every hyperedge is of the same order, a k-uniform hypergraph with N nodes can be naturally represented by a kth-order adjacency tensor $\mathcal{A} \in \mathbb{R}^{N\times N\times\cdots\times N}$, where each index ranges from 1 to N, and the entries of $\mathcal{A}$ are defined as follows [103,104]

$$A_{i_1\cdots i_k} = 1, \qquad \text{if } \{v_{i_1},\cdots,v_{i_k}\} \in \mathcal{E}. \tag{30}$$

Every other entry in $\mathcal{A}$ is set to zero. Similarly to how it can be meaningful to normalize the adjacency matrix, normalized versions of this adjacency tensor have been proposed as well. In [105], the tensor in (30) is normalized by $1/(k-1)!$. This normalization guarantees that the degree of a vertex v_i, i.e., the number of hyperedges that it belongs to, can be retrieved by summing the entries in the tensor whose first mode index is i, namely $\deg(v_i) = \sum_{i_2,\cdots,i_k=1}^{N} A_{ii_2\cdots i_k}$; see [108]. This is desirable because it resembles the way of obtaining the degree of a vertex in a graph from its adjacency matrix. Another normalized adjacency is proposed in [106] where

$$A_{i_1\cdots i_k} = \frac{1}{(k-1)!}\prod_{j=1}^{k}\frac{1}{\sqrt[k]{\deg(v_{i_j})}}, \qquad \text{if } \{v_{i_1},\cdots,v_{i_k}\} \in \mathcal{E}, \tag{31}$$

and the rest of the entries are equal to zero. Its associated normalized Laplacian tensor is defined as $\mathcal{L} = \mathcal{J} - \mathcal{A}$ where $\mathcal{J}$ is a tensor of the same size as $\mathcal{A}$, and its entry $J_{ii\cdots i} = 1$ if $\deg(v_i) > 0$ and 0 otherwise. This normalization ensures that $\mathcal{L}$ has certain desirable spectral properties that mimic those of the normalized graph Laplacian [106]. For example, the eigenvalues of $\mathcal{L}$ as defined in [113] are guaranteed to be contained in [0,2]. Having a bounded spectrum has shown to be useful in GSP for the stability analysis of graph filters [114].

For hypergraphs with non-uniform hyperedges, i.e., hyperedges of different sizes, the above construction does not extend easily. Since some edges will have smaller cardinality than others, some indices in the adjacency tensor would simply be undefined. A naive approach would be to keep an adjacency tensor for each observed cardinality of hyperedges, but this approach is computationally impractical. An alternative is to augment the above construction of an adjacency tensor for general homogeneous hypergraphs as follows. Denote by m the cardinality of the largest hyperedge across

all hyperedges $e \in \mathcal{E}$. Then, we construct an adjacency tensor of order m according to the following rules [107]. For every hyperedge $e = \{v_{i_1}, \cdots, v_{i_s}\} \in \mathcal{E}$ of cardinality $s \leq m$, we assign the following nonzero entries to $\mathcal{A}$

$$A_{p_1 p_2 \cdots p_m} = s \cdot \left(\sum_{l_1, \cdots, l_s \geq 1, \sum_{j=1}^{s} l_j = m} \frac{m!}{l_1! l_2! \cdots l_s!} \right)^{-1}, \tag{32}$$

where the indices $p_1, p_2, \cdots, p_m$ are chosen in all possible ways from $\{i_1, i_2, \cdots, i_s\}$ such that every element of this latter set is represented at least once. The rest of the entries of $\mathcal{A}$ are set to zero. The Laplacian tensor is then defined as $\mathcal{L} = \mathcal{D} - \mathcal{A}$ where $\mathcal{D}$ is a super-diagonal tensor of the same size as $\mathcal{A}$ and with entries $D_{ii\cdots i}$ equal to the degree of vertex v_i. To illustrate definition (32), consider the following example.

Example 10. Consider a hypergraph composed of four nodes v_1, v_2, v_3, v_4 and two hyperedges $e_1 = \{v_1, v_2, v_3\}$ and $e_2 = \{v_3, v_4\}$. We have that $m = \max\{|e_1|, |e_2|\} = 3$ and the adjacency tensor is of size $4 \times 4 \times 4$. For e_1, the corresponding $s = |e_1| = 3$ and $l_1 = l_2 = l_3 = 1$, thus the corresponding entries in the tensor are defined as $A_{123} = A_{132} = A_{213} = A_{231} = A_{312} = A_{321} = 3/3! = 1/2$. For e_2, the corresponding $s = |e_2| = 2$ and there are two choices for l_1 and l_2, i.e., $l_1 = 1, l_2 = 2$ or $l_1 = 2, l_2 = 1$. Thus we have $A_{344} = A_{434} = A_{443} = A_{334} = A_{343} = A_{433} = 2 \cdot (3!/2! + 3!/2!)^{-1} = 1/3$. The remaining entries are set to zero.

Having defined adjacency and Laplacian tensors, we can now construct appropriate shift operators based on these tensors. In the context in which we are interested in processing signals $\mathbf{y} = [y_1, y_2, \cdots, y_N]^\top$ defined on the nodes, the following approach has been proposed [112]. First, given the signal vector $\mathbf{y}$ construct the following $(m-1)$th-order outer product tensor $\mathcal{Y} \in \mathbb{R}^{N \times \cdots \times N}$ as

$$\mathcal{Y} = \underbrace{\mathbf{y} \circ \cdots \circ \mathbf{y}}_{m-1 \text{ times}}, \text{ with entries } Y_{i_1 i_2 \cdots i_{m-1}} = y_{i_1} y_{i_2} \cdots y_{i_{m-1}}, \tag{33}$$

where $\circ$ denotes the tensor outer product and m is the order of the adjacency or Laplacian (shift) tensor of interest. Then, the hypergraph shift operation leading to the output signal $\mathbf{y}^{\text{out}} \in \mathbb{R}^N$ is defined elementwise as

$$y_i^{\text{out}} = \sum_{j_1, \cdots, j_{m-1}=1}^{N} S_{i j_1 \cdots j_{m-1}} y_{j_1} y_{j_2} \cdots y_{j_{m-1}}, \tag{34}$$

where $S_{i j_1 \cdots j_{m-1}}$ correspond to the entries of the chosen shift tensor. Equivalently, we may express the above in terms of tensor products as

$$\mathbf{y}^{\text{out}} = \mathcal{S}\mathcal{Y}. \tag{35}$$

Note that, due to the symmetry of the tensor $\mathcal{S}$, it does not matter which mode we leave out in the tensor multiplication, i.e., which of the indices is kept fixed to i in (34). Furthermore, for the specific case where $m = 2$, we have that $\mathcal{Y} = \mathbf{y}$ in (33) and the shift operation in (34) boils down to a standard matrix-vector multiplication as in GSP.

Example 11. Consider the hypergraph in Fig. 8: vertex v_3 is contained in two hyperedges $e_1 = \{v_1, v_2, v_3\}$ and $e_2 = \{v_3, v_4, v_5\}$. We define the adjacency tensor as the shift tensor $\mathcal{S}$. According to (34), the output signal at vertex v_3 after one hypergraph shift is computed as

$$y_3^{\text{out}} = S_{321} \times y_2 y_1 + S_{312} \times y_1 y_2 + S_{354} \times y_5 y_4 + S_{345} \times y_4 y_5. \tag{36}$$

As in the graph case where the entry S_{ij} of the shift operator indicates the shift from vertex v_j to vertex v_i, the entry $S_{i j_1 \cdots j_{m-1}}$ of the hypergraph shift operator indicates the shift in one hyperedge following the order $v_{j_{m-1}} \to v_{j_{m-2}} \to \cdots \to v_{j_1} \to v_i$. Fig. 8 illustrates the process defined by (36).

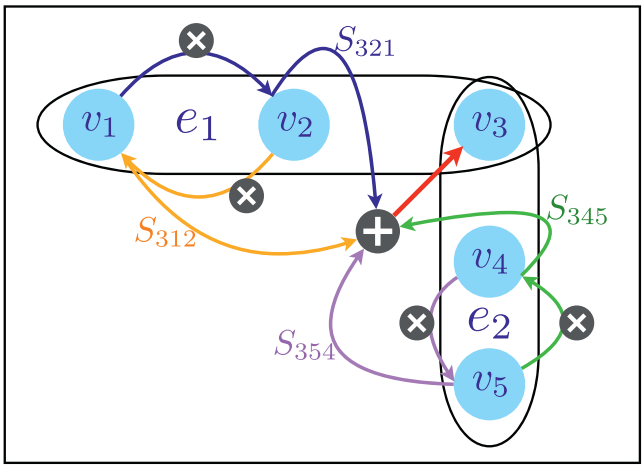

Fig. 8. Tensor based shift operator on a hypergraph. The output of y_3^{out} at vertex v_3 is determined by a weighted sum over the hyperedges incident to v_3, where the summands correspond to the products of the vertex signals within the respective hyperedges excluding v_3. [Figure adapted from Fig. 10(a) in [112]].

5.3. Comparison between matrix-based and tensor-based hypergraph representations

The major advantage of matrix-based methods is that a lot of well-developed graph-related algorithms can be directly utilized. However, if the resulting matrix representation is akin to a graph in that it only encodes pairwise relations between vertices (clique expansion), or hyperedges (line graphs), there will be some information loss, in general, compared to the original hypergraph structure. In contrast, for the star-expansion, all the incidence information is kept in the matrix representation. However, the resulting graph is bipartite. The bipartite graph structure might be undesirable for some applications since there are no explicit links between the same types of vertices and there are much fewer algorithms tailored for bipartite graphs than those for simple graphs [101].

Compared with matrix representations, tensors can better retain the set-level information contained in hypergraphs. However, tensor computations are more complicated and lack algorithmic guarantees [110]. For example, determining the rank of a specific tensor is NP-hard [115]. Most existing papers have focused on super-symmetric tensors [113], while more general tensors are less explored. Indeed, how to best leverage tensor-based representations to study hypergraphs that are not homogeneous is an open problem.

Remark 12. There is a rich and complementary line of research on *nonlinear* Laplacian operators. In [116,117], a continuous diffusion process on the hypergraph is considered to define a Laplacian operator that enables a Cheeger-type inequality for hypergraphs. To understand this diffusion process, suppose that, at some instant, there is some signal $\mathbf{y} \in \mathbb{R}^{|\mathcal{V}|}$ defined on the vertices of a hypergraph. Each hyperedge $e \in \mathcal{E}$ directs flow from vertices $S_e(\mathbf{y}) = arg\,max_{v_i \in e} y_i$ having the maximum signal value to vertices $I_e(\mathbf{y}) = arg\,min_{v_i \in e} y_i$ having the minimum signal value, at a total rate of $c_e = \omega(e) \cdot \max_{v_i, v_j \in e} |y_i - y_j|$. As the diffusion progresses, the cardinality of $S_e(\mathbf{y})$ and $I_e(\mathbf{y})$ increases, conferring a nonlinear nature to the diffusion process, which can be modeled through a nonlinear Laplacian. A generalization of this process was proposed in [118], where hyperedges can act as mediators to receive flow from vertices in $S_e(\mathbf{y})$ and deliver flow to those in $I_e(\mathbf{y})$. Moreover, a unifying framework was recently presented in [119] by proposing a Cheeger inequality for submodular transformations. In particular, the Laplacian operators as well as the Cheeger inequalities for

undirected graphs, directed graphs and hypergraphs can be recovered by defining proper submodular transformations; see [119] for more details. In [89], similar results have been independently obtained for symmetric submodular transformations.

6. Signal processing and learning on hypergraphs

Mimicking the respective developments in Section 2.4 for graphs and Section 4 for simplicial complexes, in this section we consider the four signal processing setups for hypergraphs equipped with the algebraic representations developed in Section 5.

6.1. Fourier analysis, node and hyperedge embeddings

As stated in Section 5.1, shift operators for hypergraphs can be represented via matrices. The corresponding eigenvectors may then be used as Fourier modes and, thus, most GSP tools discussed in Section 2 can be directly translated to hypergraphs for matrix-based hypergraph shift operators. However, unlike for graphs, even an undirected hypergraph may result in an *asymmetric* matrix, e.g., if hyperedge weightings are considered. Hence, one may have to adopt tools from GSP for directed graphs in this case; see [19] for a more detailed exposition of these issues.

In contrast to matrix-based shift operators, the notion of Fourier analysis for hypergraphs represented via tensors is far less developed. Nonetheless, we may proceed analogously to the matrix case and define Fourier modes via a tensor decomposition, in lieu of the eigenvector decomposition. Specifically, we can consider the orthogonal canonical polyadic (CP) decomposition [120] of the adjacency tensor $\mathcal{A}$ (other representative tensors can also be considered) given by

$$\mathcal{A} = \sum_{r=1}^{R} \lambda_r \cdot \underbrace{\mathbf{v}_r \circ \cdots \circ \mathbf{v}_r}_{m \text{ times}}, \tag{37}$$

where λ_r are scalars, and R is the so-called rank of the tensor (i.e., R is the smallest number such that $\mathcal{A}$ can be represented as a weighted sum of rank-1 outer-product tensors). Using this decomposition, the hypergraph Fourier basis can then be defined as $\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_N]$. When $R < N$, the first R vectors are determined via the CP decomposition and $N - R$ additional vectors satisfying specific conditions are selected to complete the basis (see Section III-F in [112] for details). Similar to the matrix case, the hypergraph Fourier frequencies are defined as the coefficients λ_r associated with the rank-1 terms in the decomposition.

Extending the arguments from the matrix case to the tensor case, the hypergraph Fourier transform (HGFT) and inverse HGFT (iHGFT) [112] are then defined as

$$\tilde{\mathbf{y}} = (\mathbf{V}^\top \mathbf{y})^{m-1}, \quad \mathbf{y} = \mathbf{V}\tilde{\mathbf{y}}^{\frac{1}{m-1}}, \tag{38}$$

where $\mathbf{y}^m = [y_1^m, \cdots, y_N^m]^\top$ denotes the m-th power of each entry of $\mathbf{y}$. By introducing such definitions, an application of the tensor shift can be equivalently interpreted as a HGFT followed by a combined operation consisting of filtering in the Fourier domain plus iHGFT (cf. Equation (25) in [112]). Observe that, when $m = 2$, the hypergraph shift defined in (34), and the HGFT and iHGFT defined in (38) have the same form as the corresponding concepts in GSP (cf. Section 2.3).

Similar to how graph Fourier modes can be used to derive node embeddings, the same can be done for hypergraphs using either matrix or tensor representations. For matrix representations, the procedure is entirely analogous. For tensor representations, we have to use a tensor decomposition but can proceed in a similar fashion once the (tensor-based) Fourier modes are derived. While tensor-based embeddings have only been scarcely considered in the literature so far, e.g., [121] represents the dual hypergraph (see Fig. 7B) using tensors and learns hyperedge embeddings by performing a symmetric tensor decomposition [122,123]. Finally, the embedding of *heterogeneous* hypergraphs entails additional challenges that can be (partially) addressed through nonlinear neural network based approaches; see [82] for more details.

6.2. Signal smoothing and denoising

As was the case for graphs and simplicial complexes, one can leverage the structure of hypergraphs to solve inverse problems associated with signals defined on them. For the specific problem of denoising, the assumption is that the signal to be recovered is smooth in the hypergraph, where smoothness typically encodes the fact that tightly connected nodes should have similar signal values. For instance, in the co-authorship network in Example 9, if two authors share many papers (hyperedges) either written solely by them or in collaboration with others, one would expect a signal that represents research interests to take similar values for the two mentioned authors. This notion of homophily is well established for graphs and naturally extends to hypergraphs.

Mathematically, in line with the graph case in Section 2.4.2, we assume that we observe a noisy version $\mathbf{y} = \mathbf{y}^0 + \boldsymbol{\epsilon}$ of the true underlying signal defined on the node set of our hypergraph $\mathcal{H}$. Then, we can try to estimate $\mathbf{y}^0$ by solving the optimization problem

$$\min_{\hat{\mathbf{y}}} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 + \alpha \Omega_{\mathcal{H}}(\hat{\mathbf{y}}), \tag{39}$$

where the first term is to constrain the denoised signal $\hat{\mathbf{y}}$ to be close to the observation $\mathbf{y}$ and the second term is a regularizer shaped by the structure of $\mathcal{H}$.

A possible choice for $\Omega_{\mathcal{H}}(\hat{\mathbf{y}})$ is to select a Laplacian matrix representation of $\mathcal{H}$ (cf. Section 5.1) and set the regularizer to the quadratic form as in the graph case [93,94]. From the discussion after (2) it follows that the optimal solution $\hat{\mathbf{y}}$ will then be a low-pass version of $\mathbf{y}$ where the bases for low and high frequencies depend on the specific graph expansion selected. The most common one is to consider the clique expansion, in which we have

$$\Omega_{\mathcal{H}}(\hat{\mathbf{y}}) = \sum_{(u,v)\in\mathcal{E}_c} \omega_c(u,v)(\hat{y}_u - \hat{y}_v)^2 = \hat{\mathbf{y}}^\top \mathbf{L}_c \hat{\mathbf{y}}, \tag{40}$$

where $\mathbf{L}_c$ corresponds to the graph Laplacian obtained via clique expansion of the hypergraph. Alternatively, one can rely on tensor-based representations for hypergraphs in the definition of $\Omega_{\mathcal{H}}(\hat{\mathbf{y}})$. In particular, we can set the regularizer to be equal to the tensor-based total variation in [112]. In this case, smooth signals would also be promoted but the meaning of a smooth signal will correspond to one that suffers little change under a tensor shift as defined in (34).

An alternative regularizer based on the Lovász extension of the hypergraph cut has also been proposed [124]. More specifically, a parametric family of regularizers was considered

$$\Omega_{\mathcal{H},p}(\hat{\mathbf{y}}) = \sum_{e\in\mathcal{E}} \omega(e) \left(\max_{u\in e} \hat{y}_u - \min_{v\in e} \hat{y}_v \right)^p, \tag{41}$$

which can be shown to be convex for $p \geq 1$. Consequently, the optimization problem (39) remains convex and, in particular, tailored efficient algorithms have been proposed for $p = 1$ and $p = 2$; see [124]. In interpreting (41) we can see that $\Omega_{\mathcal{H},p}(\hat{\mathbf{y}})$ induces yet another related notion of smoothness. For every hyperedge $e \in \mathcal{E}$ we look at the difference between the extreme values of the signal attained at the nodes contained in e, we scale this penalization by the weight of the hyperedge, and we sum over all hyperedges. Intuitively, this regularizer promotes signals that are constant within the hyperedges. Moreover, the power p controls the form of the deviations from these piecewise constant signals. For

example, the sparsity promoting $p = 1$ would encourage the signal variation to be zero within some hyperedges and possibly high in others, whereas $p = 2$ would promote a low (possibly non-zero) variation across all hyperedges.

If we consider a general submodular function F_e instead of the hypergraph cut, then (41) can be generalized as

$$\Omega_{\mathcal{H},p}(\hat{\mathbf{y}}) = \sum_{e\in\mathcal{E}} [f_e(\hat{\mathbf{y}})]^p, \tag{42}$$

where f_e is the Lovász extension of F_e (cf. Remark 12). The optimization problem (39) equipped with (42) are respectively referred to as decomposable submodular function minimization (DSFM) for $p = 1$ [125–130] and quadratic DSFM (QDSFM) for $p = 2$ [131]. Similar to (40) which can also be written as $\langle \hat{\mathbf{y}}, \mathbf{L}_c \hat{\mathbf{y}} \rangle$, (42) can be viewed as $\langle \hat{\mathbf{y}}, \mathcal{L}(\hat{\mathbf{y}}) \rangle$ for some Laplacian operator $\mathcal{L}$ depending on F_e.

6.3. Signal interpolation on hypergraphs

As discussed in the previous sections, signal interpolation and smoothing are closely related problems. Successful signal interpolation from an observed subset $\mathcal{V}^L$ hinges to a large extent on the selection of a sensible model for a (smooth) ground truth signal that is compatible with the observed (desired) signal characteristics. For a chosen signal model, we may then again set up an optimization problem for interpolating hypergraph signals as

$$\min_{\hat{\mathbf{y}}} \Omega_{\mathcal{H}}(\hat{\mathbf{y}}), \qquad \text{s.t. } \hat{y}_v = y_v \text{ for all } v \in \mathcal{V}^L, \tag{43}$$

where $\Omega_{\mathcal{H}}$ is a regularizer chosen to promote the desired signal characteristics, e.g., a low-pass signal. Like for graphs and simplicial complexes, many choices for the regularization term are possible here and the optimal choice of a regularizer will generally be dependent on the considered application scenario. For instance, we may choose to use a regularizer based on the clique expansion or some of the other strategies discussed in Section 6.2. Unlike in the graph and simplicial complex setting, however, for hypergraphs we may also consider tensor-based regularizers, which can offer smoothing and interpolation strategies that are not accessible via matrix-based approaches. Developing and analyzing such approaches for hypergraphs appears to be an interesting avenue for future research. Problem (43) can also be converted to another class of optimization problem called submodular Laplacian system [132] which is a generalization of the Laplacian system on graphs [39].

6.4. Hypergraph neural networks

The design of neural network architectures to process and learn from data on hypergraphs is a nascent area of research. Given the developments in graph neural networks mentioned in Section 2.4.4 and the graph expansions for hypergraphs introduced in Section 5.1, an avenue to derive hypergraph neural networks is to compute the graph shifts based on the (clique, star, or line graph) expansions of the hypergraph and then apply a (classical) graph neural network as the one in (6) or any of the variants surveyed in [41].

In this direction, one of the earliest hypergraph neural networks [133] adopts the hypergraph Laplacian matrix associated with a weighted clique expansion in [94] as a graph shift and then implements a graph convolutional network [45,46] where shift-invariant filters are intertwined with pointwise nonlinearities. One drawback of the clique expansion is that the resulting graph tends to be dense since a hyperedge is replaced by a number of edges that is quadratic in the size of the hyperedge. A similar idea is proposed in [134], but this convolutional neural network is based on a different hypergraph Laplacian shift (proposed in [118]), which only requires a linear number of edges for each hyperedge. This provides a more efficient training when compared with that of [133]. Under this same methodological umbrella, a line hypergraph convolution network is proposed in [100], which expands the hypergraph into a weighted and attributed line graph and then implements a graph convolutional network using the corresponding shift operator.

Architectures grounded on the message-passing variants of graph neural networks (cf. Section 2.4.4) have also been proposed for hypergraphs. For instance, in [101] the line expansion of the hypergraph is used to define a message passing process where potentially different aggregation functions can be used when passing messages between nodes in the expansion that have either one vertex or one hyperedge in common; see Fig. 7-F. Also, [135] proposes a generalization of GraphSAGE [136] to hypergraphs, a well-established message passing architecture for graphs. Recent developments that further extend the state of the art include architectures that tackle the issue that the initially constructed hypergraphs may not be a suitable representation for data [137] as well as the formulation of attention [138] and self-attention [83] mechanisms for hypergraphs.

As a closing note, a different perspective is put forth in [139], where a convolutional neural network architecture for powerset data is introduced. These architectures are designed to learn from set functions, which are signals on the powerset of a given set. By noticing that cuts in hypergraphs can be interpreted as set functions, these convolutional architectures can be used to solve problems in hypergraphs; see [139] for more details.

7. Discussion

Graph signal processing tools have been highly successful in a wide range of applications, ranging from biological to social domains. This success hinges to a large extent on providing sensible notions for filtering graph signals, such that the relevant dependencies in the signal are kept intact, while undesirable noise components are filtered out. However, as graphs are only concerned with pairwise relationships, their capabilities for modeling higher-order dependencies are too limited for certain application scenarios in which polyadic relationships are essential. In such scenarios, simplicial complexes and hypergraphs have recently emerged as two promising conceptual frameworks to address the specific shortcomings of graph-based representations.

Unlike for GSP that can benefit from a rich set of results in spectral graph theory, e.g., to derive appropriate notions of shift operators and signal smoothness, the theory of signal processing on higher-order networks is far less developed. In this tutorial paper, we provided an introduction to this emerging area, focusing on the choice of appropriate shift operators and associated frequency domain representations, as well as a set of important application scenarios comprising signal smoothing and denoising, signal interpolation, and the construction of nonlinear neural network architectures that can leverage the structure of such higher-order networks.

We believe that this area holds an enormous potential for future developments. A few relevant future direction include the following.

In the context of simplicial complexes, the investigation of how these should be constructed from data to capture desirable features is certainly one aspect that deserves further research. As discussed in Sections 3 and 4, the choice of appropriate faces has direct consequences on the frequency representation of any signal and is thus highly relevant for applications [63]. Similarly, while we discussed only unweighted simplicial complexes for simplicity, the appropriate introduction of weights to emphasize certain features in the data to be investigated is a pertinent issue that should

be addressed in future research. Finally, while we concentrated on simplicial complexes as the most common complexes considered, the restriction to simplicial instead of other type of cell complexes such as cubical complexes is essentially artificial. From a modeling perspective, simplices may not always capture the appropriate notion of a "cell" in a higher-order interaction network. For instance, in traffic and street networks it may be beneficial to consider cubical complexes or other types of models that can better represent the grid-like structure of many of these networks [59].

In the context of hypergraphs, we provide several potential directions for future work. As the first step, constructing a suitable hypergraph is key to the final performance. Hence, it is important to develop effective and efficient methods for the construction of hypergraphs from real-world datasets that are usually large-scale. To better characterize a wider range of datasets, it is necessary to develop more general hypergraph models, such as those considering different types of vertices or having different levels of relations (cf. Section 5). A variety of problems that have been well studied in graphs or homogeneous hypergraphs are valuable to be reconsidered and extended to those less explored but more expressive models. These problems include, but are not limited to, developing spectral hypergraph theory, node clustering, classification and ranking, link prediction, hypergraph representation learning (especially for heterogeneous hypergraphs in which hyperedges are generally indecomposable [82]), the modeling and analysis of diffusion processes on hypergraphs, tensor-based representations and operations (especially for hypergraphs with edge-dependent vertex weights which are hard to be modeled using super-symmetric tensors), hypergraph kernels, hypergraph classification, and hypergraph alignment. Although one framework for hypergraph signal processing has already been proposed in [112], there are still many open questions. In GSP, graph shift and filters can be understood as some network diffusion processes, while it is not clear if and how the hypergraph shift can be connected with a physical process. Other problems such as hypergraph filter design, active sampling for reconstruction, and fast hypergraph Fourier transforms are also worth investigating. Finally, most existing hypergraph neural networks are matrix-based like those introduced in Section 6.4. A natural extension in this context would be to derive the theory of tensor-based neural networks for hypergraphs.

Declaration of Competing Interest

We declare the following conflicts of interest, regarding our submission "Signal Processing on Higher-Order Networks: Livin' on the Edge... and Beyond" to Signal Processing. The last co-author, Santiago Segarra, is a guest editor for the special issue we are submitting to. There are no COI for any of the other authors.

Acknowledgements

This work was partially supported by USA NSF under award CCF-2008555. MTS acknowledges partial funding from the Ministry of Culture and Science (MKW) of the German State of North Rhine-Westphalia ("NRW Rückkehrprogramm"), and the Excellence Strategy of the Federal Government and the Länder.

References

[1] M. Newman, Networks, Oxford University Press, 2018.
[2] D. Easley, J. Kleinberg, Networks, Crowds, and Markets, volume 8, Cambridge University Press, 2010.
[3] O. Sporns, Graph theory methods: applications in brain networks, Dialog. Clin. Neurosci. 20 (2) (2018) 111–121.
[4] J.D. Medaglia, W. Huang, S. Segarra, C. Olm, J. Gee, M. Grossman, A. Ribeiro, C.T. McMillan, D.S. Bassett, Brain network efficiency is influenced by the pathologic source of corticobasal syndrome, Neurology 89 (13) (2017) 1373–1381.
[5] S. Derible, C. Kennedy, Applications of graph theory and network science to transit network design, Transp. Rev. 31 (4) (2011) 495–519.
[6] S. Borgatti, A. Mehra, D. Brass, G. Labianca, Network analysis in the social sciences, Science 323 (892) (2009).
[7] A. Sandryhaila, J. Moura, Discrete signal processing on graphs, IEEE Trans. Signal Process. 61 (7) (2013) 1644–1656.
[8] D. Shuman, S. Narang, P. Frossard, A. Ortega, 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. 30 (7) (2013) 83–98.
[9] A. Ortega, P. Frossard, J. Kovacevic, P. Vandergheynst, Graph signal processing: overview, challenges and applications, Proc. IEEE 106 (5) (2018) 808–828.
[10] S. Segarra, A.G. Marques, A. Ribeiro, Optimal graph-filter design and applications to distributed linear network operators, IEEE Trans. Signal Process. 65 (15) (2017) 4117–4131.
[11] F.R.K. Chung, Spectral Graph Theory, American Mathematical Society, 1997.
[12] C. Giusti, R. Ghrist, D.S. Bassett, Two's company, three (or more) is a simplex: Algebraic-topological tools for understanding higher-order structure in neural data, J. Comput. Neurosci. 41 (2016) 1–14.
[13] S. Klamt, U.-U. Haus, F. Theis, Hypergraphs and cellular networks, PLoS Comput. Biol. 5 (2009).
[14] K. Kee, L. Sparks, D. Struppa, M. Mannucci, Social groups, social media, and higher-dimensional social structures: a simplicial model of social aggregation for computational communication research, Commun. Q. 61 (2013) 35–68.
[15] A. Hatcher, Algebraic Topology, Cambridge University Press, 2002.
[16] C. Berge, Hypergraphs, Elsevier, 1989.
[17] P. Frankl, Extremal Set Systems, MIT Press, Cambridge, MA, USA, 1996, pp. 1293–1329.
[18] M. Robinson, Topological Signal Processing, 81, Springer, 2014.
[19] A.G. Marques, S. Segarra, G. Mateos, Signal processing on directed graphs: the role of edge directionality when processing and learning from network data, IEEE Signal Process. Mag. 37 (6) (2020) 99–116.
[20] S. Furutani, T. Shibahara, M. Akiyama, K. Hato, M. Aida, Graph signal processing for directed graphs based on the hermitian Laplacian, in: U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, C. Robardet (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer International Publishing, Cham, 2020, pp. 447–463.
[21] A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Processing, third ed., Prentice Hall Press, USA, 2009.
[22] A.G. Marques, S. Segarra, G. Leus, A. Ribeiro, Sampling of graph signals with successive local aggregations, IEEE Trans. Signal Process. 64 (7) (2016) 1832–1843.
[23] S. Chen, R. Varma, A. Sandryhaila, J. Kovacevic, Discrete signal processing on graphs: Sampling theory, IEEE Trans. Signal Process. 63 (24) (2015) 6510–6523.
[24] A. Anis, A. Gadde, A. Ortega, Efficient sampling set selection for bandlimited graph signals using graph spectral proxies, IEEE Trans. Signal Process. 64 (14) (2016) 3775–3789.
[25] S. Segarra, G. Mateos, A.G. Marques, A. Ribeiro, Blind identification of graph filters, IEEE Trans. Signal Process. 65 (5) (2017) 1146–1159.
[26] Y. Zhu, F. Iglesias, A.G. Marques, S. Segarra, Estimating network processes via blind identification of multiple graph filters, IEEE Trans. Signal Process. 68 (2020) 3049–3063.
[27] X. Dong, D. Thanou, P. Frossard, P. Vandergheynst, Learning Laplacian matrix in smooth graph signal representations, IEEE Trans. Signal Process. 64 (23) (2016) 6160–6173.
[28] S. Segarra, A. Marques, G. Mateos, A. Ribeiro, Network topology inference from spectral templates, IEEE Trans. Signal Inf. Process. Netw. 3 (3) (2017) 467–483.
[29] Y. Shen, B. Baingana, G.B. Giannakis, Kernel-based structural equation models for topology identification of directed networks, IEEE Trans. Signal Process. 65 (10) (2017) 2503–2516.
[30] G. Mateos, S. Segarra, A.G. Marques, A. Ribeiro, Connecting the dots: Identifying network structure via graph signal processing, IEEE Signal Process. Mag. 36 (3) (2019) 16–43.
[31] A. Heimowitz, Y.C. Eldar, A unified view of diffusion maps and signal processing on graphs, in: Samp. Theory and Appl. (SampTA), IEEE, 2017, pp. 308–312.
[32] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Comput. 15 (6) (2003) 1373–1396.
[33] R.R. Coifman, S. Lafon, Diffusion maps, Appl. Comput. Harmonic Anal. 21 (1) (2006) 5–30.
[34] W.L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: methods and applications, arXiv Preprint (2017). 1709.05584
[35] S. Chen, A. Sandryhaila, J.M.F. Moura, J. Kovacevic, Signal denoising on graphs via graph filtering, in: IEEE Global Conf. Signal and Info. Process. (GlobalSIP), 2014, pp. 872–876.
[36] D. Zhou, B. Schölkopf, A regularization framework for learning from graph data, in: ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields, 2004, pp. 132–137.
[37] S. Segarra, A.G. Marques, G. Leus, A. Ribeiro, Reconstruction of graph signals through percolation from seeding nodes, IEEE Trans. Signal Process. 64 (16) (2016) 4363–4378.
[38] O. Chapelle, J. Weston, B. Schölkopf, Cluster kernels for semi-supervised learning, in: Adv. Neural Info. Process. Syst. (NeurIPS), in: NIPS'02, MIT Press, Cambridge, MA, USA, 2002, pp. 601–608.

[39] X. Zhu, Z. Ghahramani, J. Lafferty, Semi-supervised learning using Gaussian fields and harmonic functions, in: Intl. Conf. Mach. Learn. (ICML), AAAI Press, 2003, pp. 912–919.
[40] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond Euclidean data, IEEE Signal Process. Mag. 34 (4) (2017) 18–42.
[41] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[42] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations., in: AAAI Conf. on Artif. Intell. (AAAI), 16, 2016, pp. 1145–1152.
[43] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: ACM Intl. Conf. on Know. Disc. and Data Mining (SIGKDD), 2016, pp. 1225–1234.
[44] T.N. Kipf, M. Welling, Variational graph auto-encoders, NeurIPS Bayesian Deep Learning Workshop, 2016.
[45] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Intl. Conf. Learn. Repres. (ICLR), 2017.
[46] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Adv. Neural Inf. Process. Syst. 29 (2016) 3844–3852.
[47] F. Gama, A.G. Marques, G. Leus, A. Ribeiro, Convolutional neural network architectures for signals supported on graphs, IEEE Trans. Signal Process. 67 (4) (2018) 1034–1049.
[48] L.-H. Lim, Hodge Laplacians on graphs, SIAM Rev. 62 (3) (2020) 685–715.
[49] M. Schaub, A. Benson, P. Horn, G. Lippner, A. Jadbabaie, Random walks on simplicial complexes and the normalized Hodge 1-Laplacian, SIAM Rev. 62 (2020) 353–391.
[50] L. Grady, J. Polimeni, Discrete Calculus: Applied Analysis on Graphs for Computational Science, Springer-Verlag, 2010.
[51] J.R. Munkres, Topology, Prentice Hall, 2000.
[52] B. Eckmann, Harmonische funktionen and randwertaufgaben in einem komplex, Commentarii Math. Helvetici 17 (1944) 240–255.
[53] A.R. Benson, R. Abebe, M.T. Schaub, A. Jadbabaie, J. Kleinberg, Simplicial closure and higher-order link prediction, Proc. Natl. Acad. Sci. 115 (48) (2018) E11221–E11230.
[54] M. Schaub, S. Segarra, Flow smoothing and denoising: graph signal processing in the edge-space, in: IEEE Global Conf. Signal and Info. Process. (GlobalSIP), 2018, pp. 735–739.
[55] J. Jia, M.T. Schaub, S. Segarra, A.R. Benson, Graph-based semi-supervised & active learning for edge flows, in: ACM Intl. Conf. on Know. Disc. and Data Mining (SIGKDD), 2019, pp. 761–771.
[56] S. Barbarossa, S. Sardellitti, Topological signal processing over simplicial complexes, IEEE Trans. Signal Process. PP (2020). 1–1
[57] U. von Luxburg, A tutorial on spectral clustering, Stat. Comp. 17 (2007) 395–416.
[58] S. Ebli, G. Spreemann, A notion of harmonic clustering in simplicial complexes, in: IEEE Intl. Conf. on Mach. Learn. and its Appl. (ICMLA), IEEE, 2019, pp. 1083–1090.
[59] A. Ghosh, B. Rozemberczki, S. Ramamoorthy, R. Sarkar, Topological signatures for fast mobility analysis, in: ACM Intl. Conf. on Adv. in Geo. Inf. Syst. (SIGSPATIAL), 2018, pp. 159–168.
[60] X. Jiang, L.-H. Lim, Y. Yao, Y. Ye, Statistical ranking and combinatorial Hodge theory, Math. Prog. 127 (1) (2011) 203–244.
[61] S. Barbarossa, S. Sardellitti, E. Ceci, Learning from signals defined over simplicial complexes, in: IEEE Data Sci. Wrksp. (DSW), 2018, pp. 51–55.
[62] M. Yang, E. Isufi, M.T. Schaub, G. Leus, Finite impulse response filters for simplicial complexes, arXiv preprint arXiv:2103.12587(2021).
[63] S. Barbarossa, S. Sardellitti, Topological signal processing: making sense of data building on multiway relations, IEEE Signal Process. Mag. 37 (6) (2020) 174–183.
[64] F. Gama, J. Bruna, A. Ribeiro, Stability properties of graph neural networks, IEEE Trans. Signal Process. 68 (2020) 5680–5695.
[65] S. Ebli, M. Defferrard, G. Spreemann, Simplicial neural networks, NeurIPS Workshop on Topological Data Analysis and Beyond, 2020.
[66] E. Bunch, Q. You, G. Fung, V. Singh, Simplicial 2-complex convolutional neural networks, NeurIPS Workshop on Topological Data Analysis and Beyond, 2020.
[67] N. Glaze, T.M. Roddenberry, S. Segarra, Principled simplicial neural networks for trajectory prediction, arXiv preprint arXiv:2102.10058(2021).
[68] C. Bodnar, F. Frasca, Y.G. Wang, N. Otter, G. Montúfar, P. Lió, M. Bronstein, Weisfeiler and Lehman go topological: message passing simplicial networks, arXiv:2103.03212 (2021).
[69] T.M. Roddenberry, S. Segarra, HodgeNet: graph neural networks for edge data, in: Asilomar Conf. Signals, Systems, and Computers, 2019, pp. 220–224.
[70] L. Neuhäuser, M.T. Schaub, A. Mellor, R. Lambiotte, Opinion dynamics with multi-body interactions, arXiv Preprint(2020a). 2004.00901
[71] L. Neuhäuser, A. Mellor, R. Lambiotte, Multibody interactions and nonlinear consensus dynamics on networked systems, Phys. Rev. E 101 (3) (2020) 032310.
[72] Y. Han, B. Zhou, J. Pei, Y. Jia, Understanding importance of collaborations in co-authorship networks: a supportiveness analysis approach, in: SIAM Intl. Conf. on Data Mining, 2009, pp. 1112–1123.
[73] U. Chitra, B.J. Raphael, Random walks on hypergraphs with edge-dependent vertex weights, in: Intl. Conf. Mach. Learn. (ICML), 2019.
[74] Y. Park, C. Priebe, D. Marchette, A. Youssef, Anomaly detection using scan statistics on time series hypergraphs, in: Link Analysis, Counterterrorism and Security (LACTS) Conference, 2009, p. 9.
[75] K. Hayashi, S.G. Aksoy, C.H. Park, H. Park, Hypergraph random walks, Laplacians, and clustering, in: ACM Conf. on Inf. and Knowl. Management (CIKM), 2020, pp. 495–504.
[76] Y. Zhu, B. Li, S. Segarra, Co-clustering vertices and hyperedges via spectral hypergraph partitioning, arXiv preprint arXiv:2102.10169(2021).
[77] K.-I. Goh, M.E. Cusick, D. Valle, B. Childs, M. Vidal, A.-L. Barabási, The human disease network, Proc. Natl. Acad. Sci. 104 (21) (2007) 8685–8690.
[78] S.G. Aksoy, C. Joslyn, C.O. Marrero, B. Praggastis, E. Purvine, Hypernetwork science via high-order hypergraph walks, EPJ Data Sci. 9 (1) (2020) 16.
[79] J. Li, J. He, Y. Zhu, E-tail product return prediction via hypergraph-based local graph cut, in: ACM Intl. Conf. on Know. Disc. and Data Mining (SIGKDD), 2018, pp. 519–527.
[80] G. Gallo, G. Longo, S. Pallottino, S. Nguyen, Directed hypergraphs and applications, Discrete Applied Mathematics 42 (2–3) (1993) 177–201.
[81] I.M. Baytas, C. Xiao, F. Wang, A.K. Jain, J. Zhou, Heterogeneous hyper-network embedding, in: IEEE Intl. Conf. on Data Mining (ICDM), 2018, pp. 875–880.
[82] K. Tu, P. Cui, X. Wang, F. Wang, W. Zhu, Structural deep embedding for hyper-networks, in: AAAI Conf. on Artif. Intell. (AAAI), 2018.
[83] R. Zhang, Y. Zou, J. Ma, Hyper-SAGNN: a self-attention based graph neural network for hypergraphs, in: Intl. Conf. Learn. Repres. (ICLR), 2020.
[84] X. Sun, H. Yin, B. Liu, H. Chen, J. Cao, Y. Shao, N.Q.V. Hung, Heterogeneous hypergraph embedding for graph classification, in: ACM Intl. Conf. on Web Search and Data Mining (WSDM), 2021.
[85] V.W. Zheng, B. Cao, Y. Zheng, X. Xie, Q. Yang, Collaborative filtering meets mobile recommendation: a user-centered approach, in: AAAI Conf. on Artif. Intell. (AAAI), 2010, pp. 236–241.
[86] D. Li, Z. Xu, S. Li, X. Sun, Link prediction in social networks based on hypergraph, in: Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 41–42.
[87] L. Su, Y. Gao, X. Zhao, H. Wan, M. Gu, J. Sun, Vertex-weighted hypergraph learning for multi-view object classification, in: Intl. Joint Conf. on Artif. Intell. (IJCAI), 2017, pp. 2779–2785.
[88] P. Li, O. Milenkovic, Inhomogeneous hypergraph clustering with applications, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2017, pp. 2308–2318.
[89] P. Li, O. Milenkovic, Submodular hypergraphs: p-Laplacians, Cheeger inequalities and spectral clustering, in: Intl. Conf. Mach. Learn. (ICML), 2018.
[90] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.
[91] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, H.A. Phan, Tensor decompositions for signal processing applications: from two-way to multiway component analysis, IEEE Signal Process. Mag. 32 (2) (2015) 145–163.
[92] N.D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E.E. Papalexakis, C. Faloutsos, Tensor decomposition for signal processing and machine learning, IEEE Trans. Signal Process. 65 (13) (2017) 3551–3582.
[93] S. Agarwal, K. Branson, S. Belongie, Higher order learning with graphs, in: Intl. Conf. Mach. Learn. (ICML), 2006, pp. 17–24.
[94] D. Zhou, J. Huang, B. Schölkopf, Learning with hypergraphs: clustering, classification, and embedding, in: Adv. Neural Info. Process. Syst. (NeurIPS), 19, 2006, pp. 1601–1608.
[95] P. Solé, et al., Spectra of regular graphs and hypergraphs and orthogonal polynomials, Eur. J. Combin. 17 (5) (1996) 461–477.
[96] M. Bolla, Spectra, Euclidean representations and clusterings of hypergraphs, Discret. Math. 117 (1–3) (1993) 19–39.
[97] J.A. Rodriguez, On the Laplacian eigenvalues and metric parameters of hypergraphs, Linear and Multilinear Algebra 50 (1) (2002) 1–14.
[98] J.A. Rodriguez, On the Laplacian spectrum and walk-regular hypergraphs, Linear Multilinear Algebra 51 (3) (2003) 285–297.
[99] D. Gibson, J. Kleinberg, P. Raghavan, Clustering categorical data: an approach based on dynamical systems, VLDB J. 8 (3–4) (2000) 222–236.
[100] S. Bandyopadhyay, K. Das, M.N. Murty, Line hypergraph convolution network: applying graph convolution for hypergraphs, arXiv Preprint (2020). 2002.03392
[101] C. Yang, R. Wang, S. Yao, T. Abdelzaher, Hypergraph learning with line expansion, arXiv Preprint (2020). 2005.04843
[102] F. Chung, Laplacians and the Cheeger inequality for directed graphs, Ann. Combin. 9 (1) (2005) 1–19.
[103] T. Michoel, B. Nachtergaele, Alignment and integration of complex networks by hypergraph-based spectral clustering, Phys. Rev. E 86 (5) (2012) 056111.
[104] D. Ghoshdastidar, A. Dukkipati, Uniform hypergraph partitioning: provable tensor methods and sampling techniques, J. Mach. Learn. 18 (1) (2017) 1638–1678.
[105] J. Cooper, A. Dutle, Spectra of uniform hypergraphs, Linear Algebra Appl. 436 (9) (2012) 3268–3292.
[106] S. Hu, L. Qi, The Laplacian of a uniform hypergraph, J. Combin. Optim. 29 (2) (2015) 331–366.
[107] A. Banerjee, A. Char, B. Mondal, Spectra of general hypergraphs, Linear Algebra Appl. 518 (2017) 14–30.
[108] X. Ouvrard, J.-M. L. Goff, S. Marchand-Maillet, Adjacency and tensor representation in general hypergraphs part 1: e-adjacency tensor uniformisation using homogeneous polynomials, arXiv Preprint (2017). 1712.08189
[109] K.J. Pearson, Spectral hypergraph theory of the adjacency hypermatrix and matroids, Linear Algebra Appl. 465 (2015) 176–187.
[110] A.R. Benson, D.F. Gleich, J. Leskovec, Tensor spectral clustering for partitioning higher-order network structures, in: SIAM Intl. Conf. on Data Mining, SIAM, 2015, pp. 118–126.

[111] A.R. Benson, Three hypergraph eigenvector centralities, SIAM J. Math. Data Sci. 1 (2) (2019) 293–312.
[112] S. Zhang, Z. Ding, S. Cui, Introducing hypergraph signal processing: theoretical foundation and practical applications, IEEE Int. Things J. 7 (1) (2019) 639–660.
[113] L. Qi, Eigenvalues of a real supersymmetric tensor, J. Symbol. Comput. 40 (6) (2005) 1302–1324.
[114] E. Isufi, A. Loukas, A. Simonetto, G. Leus, Autoregressive moving average graph filtering, IEEE Trans. Signal Process. 65 (2) (2017) 274–288.
[115] J. Håstad, Tensor rank is NP-complete, in: International Colloquium on Automata, Languages, and Programming, Springer, 1989, pp. 451–460.
[116] A. Louis, Hypergraph Markov operators, eigenvalues and approximation algorithms, in: Proceedings of the forty-seventh annual ACM symposium on Theory of computing, 2015, pp. 713–722.
[117] T.-H.H. Chan, A. Louis, Z.G. Tang, C. Zhang, Spectral properties of hypergraph laplacian and approximation algorithms, J. ACM 65 (3) (2018) 1–48.
[118] T.-H.H. Chan, Z. Liang, Generalizing the hypergraph Laplacian via a diffusion process with mediators, Theor. Comput. Sci. 806 (2020) 416–428.
[119] Y. Yoshida, Cheeger inequalities for submodular transformations, in: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, 2019, pp. 2582–2601.
[120] A. Afshar, J.C. Ho, B. Dilkina, I. Perros, E.B. Khalil, L. Xiong, V. Sunderam, CP-ORTHO: an orthogonal tensor factorization framework for spatio-temporal data, in: ACM Intl. Conf. on Adv. in Geo. Inf. Syst. (SIGSPATIAL), 2017, pp. 1–4.
[121] A. Sharma, S. Joty, H. Kharkwal, J. Srivastava, Hyperedge2vec: distributed representations for hyperedges, 2018, (http://mesh.cs.umn.edu/papers/hyp2vec.pdf).
[122] P. Comon, G. Golub, L.-H. Lim, B. Mourrain, Symmetric tensors and symmetric tensor rank, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 1254–1279.
[123] T.G. Kolda, Numerical optimization for symmetric tensor decomposition, Math. Prog. 151 (1) (2015) 225–248.
[124] M. Hein, S. Setzer, L. Jost, S.S. Rangapuram, The total variation on hypergraphs-learning on hypergraphs revisited, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2013, pp. 2427–2435.
[125] P. Stobbe, A. Krause, Efficient minimization of decomposable submodular functions, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2010, pp. 2208–2216.
[126] S. Jegelka, F. Bach, S. Sra, Reflection methods for user-friendly submodular optimization, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2013, pp. 1313–1321.
[127] R. Nishihara, S. Jegelka, M.I. Jordan, On the convergence rate of decomposable submodular function minimization, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2014, pp. 640–648.
[128] A. Ene, H. Nguyen, Random coordinate descent methods for minimizing decomposable submodular functions, in: Intl. Conf. Mach. Learn. (ICML), 2015, pp. 787–795.
[129] A. Ene, H.L. Nguyên, L.A. Végh, Decomposable submodular function minimization discrete and continuous, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2017, pp. 2874–2884.
[130] P. Li, O. Milenkovic, Revisiting decomposable submodular function minimization with incidence relations, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2018, pp. 2242–2252.
[131] P. Li, N. He, O. Milenkovic, Quadratic decomposable submodular function minimization: theory and practice, J. Mach. Learn. 21 (106) (2020) 1–49.
[132] K. Fujii, T. Soma, Y. Yoshida, Polynomial-time algorithms for submodular laplacian systems, arXiv preprint arXiv:1803.10923(2018).
[133] Y. Feng, H. You, Z. Zhang, R. Ji, Y. Gao, Hypergraph neural networks, in: AAAI Conf. on Artif. Intell. (AAAI), 33, 2019, pp. 3558–3565.
[134] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, P. Talukdar, HyperGCN: a new method for training graph convolutional networks on hypergraphs, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2019, pp. 1511–1522.
[135] D. Arya, D.K. Gupta, S. Rudinac, M. Worring, HyperSAGE: Generalizing inductive representation learning on hypergraphs, arXiv Preprint (2020). 2010.04558
[136] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2017, pp. 1024–1034.
[137] J. Jiang, Y. Wei, Y. Feng, J. Cao, Y. Gao, Dynamic hypergraph neural networks., in: Intl. Joint Conf. on Artif. Intell. (IJCAI), 2019, pp. 2635–2641.
[138] S. Bai, F. Zhang, P.H. Torr, Hypergraph convolution and hypergraph attention, arXiv Preprint (2019). 1901.08150
[139] C. Wendler, M. Püschel, D. Alistarh, Powerset convolutional neural networks, in: Adv. Neural Info. Process. Syst. (NeurIPS), 2019, pp. 929–940.