Big Data Analysis with Signal Processing on Graphs

Aliaksei Sandryhaila, *Member, IEEE* and José M. F. Moura, *Fellow, IEEE*

Analysis and processing of very large datasets, or Big Data, poses a significant challenge. Massive datasets are collected and studied in numerous domains, from engineering sciences to social networks, biomolecular research, commerce and security. Extracting valuable information from Big Data requires innovative approaches that efficiently process large amounts of data as well as handle and, moreover, utilize their structure. This article discusses a paradigm for large-scale data analysis based on the *discrete signal processing on graphs* (DSP_G). DSP_G extends signal processing concepts and methodologies from the classical signal processing theory to data indexed by general graphs. Big Data analysis presents several challenges to DSP_G, in particular, in filtering and frequency analysis of very large datasets. We review fundamental concepts of DSP_G, including graph signals and graph filters, graph Fourier transform, graph frequency and spectrum ordering, and compare them with their counterparts from the classical signal processing theory. We then consider product graphs as a graph model that helps extend the application of DSP_G methods to large datasets through efficient implementation based on parallelization and vectorization. We relate the presented framework to existing methods for large-scale data processing and illustrate it with an application to data compression.

I. Introduction

Data analysts in scientific, government, industrial, and commercial domains face the challenge of coping with rapidly growing volumes of data that are collected in numerous applications. Examples include biochemical and genetics research, fundamental physical experiments and astronomical observations, social networks, consumer behavior studies, and many others. In these applications, large amounts of raw data can be used for decision making and action planning, but their volume and increasingly complex structure limit the applicability of many well-known approaches widely used with small datasets, such as principal component analysis, singular value decomposition, spectral analysis, and others. This problem—the *Big Data* problem [1]—requires new paradigms, techniques, and algorithms.

Several approaches have been proposed for representation and processing of large datasets with complex structure. Multi-dimensional data, described by multiple parameters, can be expressed and analyzed using multi-way arrays [2], [3], [4]. Multi-way arrays have been used in biomedical signal processing [5], [6], telecommunications and sensor array processing [7], [8], [9], and other domains.

Low-dimensional representations of high-dimensional data have been extensively studied in [10], [11], [12], [13]. In these approaches, datasets are viewed as graphs in high-dimensional spaces and data are projected on low-dimensional subspaces generated by small subsets of the graph Laplacian eigenbasis.

Signal processing on graphs extends classical signal processing theory to general graphs. Some techniques, such as in [14], [15], [16], are motivated in part by the works on graph Laplacian-based low-dimensional data representations. *Discrete signal processing on graphs* (DSP_G) [17], [18] builds upon the algebraic signal processing theory [19], [20].

This article considers the use of DSP_G as a methodology for Big Data analysis. We discuss how, for appropriate graph models, fundamental signal processing techniques, such as filtering and frequency analysis, can be implemented efficiently for large data sizes. The discussed framework addresses some of the key challenges of Big Data through arithmetic cost reduction of associated algorithms and use of parallel and distributed computations. The presented methodology introduces elements of high-performance computing to DSP_G and offers a structured approach to the development of data analysis tools for large data volumes.

II. Signal Processing on Graphs

We begin by reviewing notation and main concepts of DSP_G. For a detailed introduction to the theory we refer the readers to [17], [18]. Definitions and constructs presented here apply to general graphs. In the special case of undirected graphs with non-negative real edge weights, similar definitions can be formulated using the graph Laplacian matrix, as discussed in [14], [15], [16] and references therein.

Graph Signals

DSP_G studies the analysis and processing of datasets in which data elements are related by dependency, similarity, physical proximity, or other properties. This relation is expressed though a graph $G = (\mathcal{V}, \mathbf{A})$, where $\mathcal{V} = \{v_0, \ldots, v_{N-1}\}$ is the set of N nodes and $\mathbf{A}$ is the weighted adjacency matrix of the graph. Each data element corresponds to a node v_n (we also say the data element is *indexed* by v_n). A non-zero weight $\mathbf{A}_{n,m} \in \mathbb{C}$ indicates the presence of a directed edge from v_m to v_n that reflects the appropriate dependency or similarity relation between the nth and mth data elements. The set of neighbors of v_n forms its *neighborhood* denoted as $\mathcal{N}_n = \{m \mid \mathbf{A}_{n,m} \neq 0\}$.

This work was supported in part by AFOSR grant FA95501210087. A. Sandryhaila and J. M. F. Moura are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890. Ph: (412)268-6341; fax: (412)268-3890. Email: asandryh@andrew.cmu.edu, moura@ece.cmu.edu.

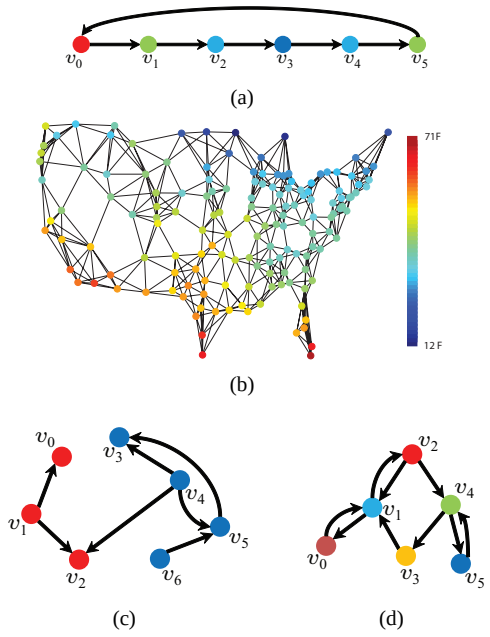


Fig. 1. Examples of graph signals. Signal values are represented with different colors. a) The periodic time series $\cos\frac{2\pi n}{6}$ resides on a directed line graph with six nodes; the edge from the last node to the first captures the periodicity of the series. b) Temperature measurements across the United States resides on the graph that represents the network of weather sensors. c) Website topics are encoded as a signal that resides on the graph formed by hyperlinks between the websites. d) Average numbers of tweets for Tweeter users are encoded as a signal that resides on the graph representing who follows whom.

Given the graph, the dataset forms a *graph signal*, defined as a map

$$\begin{aligned} \mathbf{s} \quad &: \quad \mathcal{V} \to \mathbb{C}, \\ &\quad\; v_n \mapsto s_n, \end{aligned}$$

where $\mathbb{C}$ is the set of complex numbers. It is convenient to write graph signals as vectors

$$\mathbf{s} = \begin{bmatrix} s_0 & s_1 & \dots & s_{N-1} \end{bmatrix}^T \in \mathbb{C}^N. \tag{1}$$

One should view the vector (1) not just as a list, but as a graph with each value s_n residing at node v_n.

Fig. 1 shows examples of graph signals. Finite periodic time series, studied by finite-time discrete signal processing (DSP) [21], [19], are indexed by directed cyclic graphs, such as the graph in Fig. 1(a). Each node corresponds to a time sample; all edges are directed and have the same weight 1, reflecting the causality of time series; and the edge from the last to the first node reflects the periodicity assumption. Data collected by sensor networks is another example of graph signals: sensor measurements form a graph signal indexed by the sensor network graph, such as the graph in Fig. 1(b). Each graph node is a sensor, and edges connect closely located sensors. Graph signals also arise in the World Wide Web: for instance, website features (topic, view count, relevance) are graph signals indexed by graphs formed by hyperlink references, such as the graph in Fig. 1(c). Each node represents a website, and directed edges correspond to hyperlinks. Finally, graph signals are collected in social networks, where characteristics of individuals (opinions, preferences, demographics) form graph signals on social graphs, such as the graph in Fig. 1(d). Nodes of the social graph represent individuals, and edges connect people based on their friendship, collaboration, or other relations. Edges can be directed (such as follower relations on Twitter) or undirected (such as friendship on Facebook or collaboration ties in publication databases).

Graph Shift

In DSP, a signal shift, implemented as a time delay, is a basic non-trivial operation performed on a signal. A delayed finite periodic time series of length N is $\tilde{s}_n = s_{n-1 \mod N}$. Using the vector notation (1), the shifted signal is written as

$$\tilde{\mathbf{s}} = \begin{bmatrix} \tilde{s}_0 & \dots & \tilde{s}_{N-1} \end{bmatrix}^T = \mathbf{C}\,\mathbf{s}, \tag{2}$$

where $\mathbf{C}$ is the $N \times N$ cyclic shift matrix (only non-zero entries are shown)

$$\mathbf{C} = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}. \tag{3}$$

Note that (3) is precisely the adjacency matrix of the periodic time series graph in Fig. 1(a).

DSP_G extends the concept of shift to general graphs by defining the *graph shift* as a local operation that replaces a signal value s_n at node v_n by a linear combination of the values at the neighbors of v_n weighted by their edge weights:

$$\tilde{s}_n = \sum_{m \in \mathcal{N}_n} \mathbf{A}_{n,m} s_m. \tag{4}$$

It can be interpreted as a first-order interpolation, weighted averaging, or regression on graphs, which is a widely-used operation in graph regression, distributed consensus, telecommunications, Markov processes and other approaches. Using the vector notation (1), the graph shift (4) is written as

$$\tilde{\mathbf{s}} = \begin{bmatrix} \tilde{s}_0 & \dots & \tilde{s}_{N-1} \end{bmatrix}^T = \mathbf{A}\mathbf{s}. \tag{5}$$

The graph shift (5) naturally generalizes the time shift (2).

Since in DSP_G the graph shift is defined axiomatically, other choices for the operation of a graph shift are possible. The advantage of the definition (4) is that it leads to a signal processing framework for linear and commutative graph filters. Other choices, such as selective averaging over a subset of neighbors for each graph vertex, do not lead to linear commutative filters and hence to well-defined concepts of frequency, Fourier transform, and others.

Graph Filters and z-transform

In signal processing, a *filter* is a system $\mathbf{H}(\cdot)$ that takes a signal (1) as an input and outputs a signal

$$\tilde{\mathbf{s}} = \begin{bmatrix} \tilde{s}_0 & \dots & \tilde{s}_{N-1} \end{bmatrix}^T = \mathbf{H}(\mathbf{s}). \tag{6}$$

Among the most widely used filters are linear shift-invariant (LSI) ones. A filter is *linear*, if for a linear combination of inputs it produces the same combination of outputs: $\mathbf{H}(\alpha\mathbf{s}_1 +$

$\beta\mathbf{s}_2) = \alpha\mathbf{H}(\mathbf{s}_1) + \beta\mathbf{H}(\mathbf{s}_2)$. Filters $\mathbf{H}_1(\cdot)$ and $\mathbf{H}_2(\cdot)$ are *commutative*, or *shift-invariant*, if the order of their application to a signal does not change the output: $\mathbf{H}_1(\mathbf{H}_2(\mathbf{s})) = \mathbf{H}_2(\mathbf{H}_1(\mathbf{s}))$.

The *z-transform* provides a convenient representation for signals and filters in DSP. By denoting the time delay (2) as z^{-1}, all LSI filters in finite-time DSP are written as polynomials in z^{-1}

$$h(z^{-1}) = \sum_{n=0}^{N-1} h_n z^{-n}, \tag{7}$$

where the coefficients $h_0, h_1, \ldots, h_{N-1}$ are called *filter taps*. Similarly, finite time signals are written as

$$s(z^{-1}) = \sum_{n=0}^{N-1} s_n z^{-n}. \tag{8}$$

The filter output is calculated by multiplying its z-transform (7) with the z-transform of the input signal (8) modulo the polynomial $z^{-N} - 1$, [19]:

$$\begin{aligned} \tilde{s}(z^{-1}) &= \sum_{n=0}^{N-1} \tilde{s}_n z^{-n} \\ &= h(z^{-1})s(z^{-1}) \mod (z^{-N} - 1). \end{aligned} \tag{9}$$

Equivalently, the output signal is given by the product [21]

$$\tilde{\mathbf{s}} = h(\mathbf{C})\mathbf{s}, \tag{10}$$

of the input signal (1) and the matrix

$$\begin{aligned} h(\mathbf{C}) &= \sum_{n=0}^{N-1} h_n \, \mathbf{C}^n \\ &= \begin{bmatrix} h_0 & h_{N-1} & \ldots & h_1 \\ h_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & h_{N-1} \\ h_{N-1} & \ldots & h_1 & h_0 \end{bmatrix}. \end{aligned} \tag{11}$$

Observe that the circulant matrix $h(\mathbf{C})$ in (11) is obtained by substituting the time shift matrix (3) for z^{-1} in the filter z-transform (7). In finite-time DSP, this substitution establishes a surjective (onto) mapping from the space of LSI filters and the space of $N \times N$ circulant matrices.

DSP_G extends the concept of filters to general graphs. Similarly to the extension of the time shift (2) to the graph shift (5), filters (11) are generalized to *graph filters* as polynomials in the graph shift [17], and all LSI graph filters have the form

$$h(\mathbf{A}) = \sum_{\ell=0}^{L-1} h_\ell \mathbf{A}^\ell. \tag{12}$$

In analogy with (10), the graph filter output is given by

$$\widetilde{\mathbf{s}} = h(\mathbf{A})\mathbf{s}. \tag{13}$$

The output can also be computed using the *graph z-transform* that represents graph filters (12) as

$$h(z^{-1}) = \sum_{\ell=0}^{L-1} h_\ell z^{-\ell}, \tag{14}$$

and graph signals (1) as polynomials $s(z^{-1}) = \sum_{n=0}^{N-1} s_n b_n(z^{-1})$, where $b_n(z^{-1})$, $0 \leq n < N$, are appropriately constructed, linearly independent polynomials of degree smaller than N (see [17] for details). Analogously to (9), the output of the graph filter (14) is obtained as the product of z-transforms modulo the minimal polynomial $m_\mathbf{A}(z^{-1})$ of the shift matrix $\mathbf{A}$:

$$\begin{aligned} \tilde{s}(z^{-1}) &= \sum_{n=0}^{N-1} \tilde{s}_n b_n(z^{-1}) \\ &= h(z^{-1})s(z^{-1}) \mod m_\mathbf{A}(z^{-1}). \end{aligned} \tag{15}$$

Recall that the minimal polynomial of $\mathbf{A}$ is the unique monic polynomial of the smallest degree that annihilates $\mathbf{A}$, i.e., $m_\mathbf{A}(\mathbf{A}) = 0$ [22].

Graph filters have a number of important properties. An inverse of a graph filter, if it exists, is also a graph filter that can be found by solving a system of at most N linear equations. Also, the number of taps in a graph filter is not larger than the degree of the minimal polynomial of $\mathbf{A}$, which provides an upper bound on the complexity of their computation. In particular, since the graph filter (12) can be factored as

$$h(\mathbf{A}) = h_{L-1} \prod_{\ell=0}^{L-1} (\mathbf{A} - g_\ell \, \mathbf{I}), \tag{16}$$

the computation of the output (13) requires, in general, $L \leq \deg m_\mathbf{A}(x)$ multiplications by $\mathbf{A}$.

Graph Fourier Transform

Mathematically, a Fourier transform with respect to a set of operators is the expansion of a signal into a basis of the operators' eigenfunctions. Since in signal processing the operators of interest are filters, DSP_G defines the Fourier transform with respect to the graph filters.

For simplicity of the discussion, assume that $\mathbf{A}$ is diagonalizable and its eigendecomposition is

$$\mathbf{A} = \mathbf{V} \, \mathbf{\Lambda} \, \mathbf{V}^{-1}, \tag{17}$$

where the columns $\mathbf{v}_n$ of the matrix $\mathbf{V} = \begin{bmatrix} \mathbf{v}_0 & \ldots & \mathbf{v}_{N-1} \end{bmatrix} \in \mathbb{C}^{N \times N}$ are the eigenvectors of $\mathbf{A}$, and $\mathbf{\Lambda} \in \mathbb{C}^{N \times N}$ is the diagonal matrix of corresponding eigenvalues $\lambda_0, \ldots, \lambda_{N-1}$ of $\mathbf{A}$. If $\mathbf{A}$ is not diagonalizable, Jordan decomposition into generalized eignevectors is used [17].

The eigenfunctions of graph filters $h(\mathbf{A})$ are given by the eigenvectors of the graph shift matrix $\mathbf{A}$ [17]. Since the expansion into the eigenbasis is given by the multiplication with the inverse eigenvector matrix [22], which always exists, the *graph Fourier transform* of a graph signal (1) is well-defined and computed as

$$\begin{aligned} \widehat{\mathbf{s}} = \begin{bmatrix} \widehat{s}_0 & \ldots & \widehat{s}_{N-1} \end{bmatrix}^T &= \mathbf{V}^{-1} \, \mathbf{s} \\ &= \mathbf{F} \, \mathbf{s} \end{aligned} \tag{18}$$

where $\mathbf{F} = \mathbf{V}^{-1}$ is the graph Fourier transform matrix.

The values $\widehat{s}_n$ in (18) are the signal's expansion in the eigenvector basis and represent the *graph frequency content* of the signal $\mathbf{s}$. The eigenvalues λ_n of the shift matrix $\mathbf{A}$ represent *graph frequencies*, and the eigenvectors $\mathbf{v}_n$ represent

the corresponding *graph frequency components*. Observe that each frequency component $\mathbf{v}_n$ is a graph signal, too, with its mth entry indexed by the node v_m.

The *inverse graph Fourier transform* reconstructs the graph signal from its frequency content by combining graph frequency components weighted by the coefficients of the signal's graph Fourier transform:

$$\begin{aligned} \mathbf{s} &= \widehat{s}_0\mathbf{v}_0 + \widehat{s}_1\mathbf{v}_1 + \ldots + \widehat{s}_{N-1}\mathbf{v}_{N-1} \\ &= \mathbf{F}^{-1}\,\widehat{\mathbf{s}} = \mathbf{V}\,\widehat{\mathbf{s}}. \end{aligned} \tag{19}$$

Analogously to other DSP_G concepts, the graph Fourier transform is a generalization of the discrete Fourier transform from DSP. Recall that the mth Fourier coefficient of a finite time series of length N is

$$\widehat{s}_m = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} s_n e^{-j\frac{2\pi}{N}mn},$$

and the time signal's discrete Fourier transform is written in vector form as $\widehat{\mathbf{s}} = \mathbf{DFT}_N\,\mathbf{s}$, where $\mathbf{DFT}_N$ is the $N \times N$ discrete Fourier transform matrix with the (n,m)th entry $1/\sqrt{N}\exp(-j2\pi nm/N)$. It is well-known that the eigendecomposition of the time shift matrix (3) is

$$\mathbf{C} = \mathbf{DFT}_N^{-1} \begin{bmatrix} e^{-j\frac{2\pi\cdot 0}{N}} & & \\ & \ddots & \\ & & e^{-j\frac{2\pi\cdot(N-1)}{N}} \end{bmatrix} \mathbf{DFT}_N\,.$$

Hence, the discrete Fourier transform is the graph Fourier transform for cyclic line graphs, such as the graph in Fig. 1(a), and $\lambda_n = \exp(-j2\pi n/N)$, $0 \le n < N$, are the corresponding frequencies. In DSP, the ratio $2\pi n/N$ in the exponent $\lambda_n = \exp(-j2\pi n/N)$ is also sometimes called (angular) frequency.

Alternative Choices of Graph Fourier Basis: In some cases, for example, when eigenvector computation is not stable, it may be advantageous to use other vectors as the graph Fourier basis, such as singular vectors or eigenvectors of the Laplacian matrix. These choices are consistent with DSP_G, since singular vectors form the graph Fourier basis when the graph shift matrix is defined as $\mathbf{A}\mathbf{A}^*$, and Laplacian eigenvectors form the graph Fourier basis when the shift matrix is defined by the Laplacian. However, the former implicitly turns the original graph into an undirected graph, and the latter explicitly requires that the original graph is undirected. As a result, in both cases the framework does not use the information about the direction of graph edges that is useful in various applications [17], [23], [18]. Examples, where relations are directed and not always reciprocal, are Twitter (if user A follows user B, user B does not necessarily follows user A), and world wide web (if document A links to document B, document B does not necessarily link to document A).

Frequency Response of Graph Filters

In addition to expressing the frequency content of graph signals, the graph Fourier transform also characterizes the effect of filters on the frequency content of signals. The filtering operation (13) can be written using (12) and (18) as

$$\widetilde{\mathbf{s}} = h(\mathbf{A})\mathbf{s} = h(\mathbf{F}^{-1}\,\mathbf{\Lambda}\,\mathbf{F})\mathbf{s} = \mathbf{F}^{-1}\,h(\mathbf{\Lambda})\,\mathbf{F}\,\mathbf{s}, \tag{20}$$

where $h(\mathbf{\Lambda})$ is a diagonal matrix with values $h(\lambda_n) = \sum_{\ell=0}^{L-1} h_\ell \lambda_n^\ell$ on the diagonal. As follows from (20),

$$\widetilde{\mathbf{s}} = h(\mathbf{A})\mathbf{s} \;\;\Leftrightarrow\;\; \mathbf{F}\,\widetilde{\mathbf{s}} = h(\mathbf{\Lambda})\widehat{\mathbf{s}}. \tag{21}$$

That is, the frequency content of a filtered signal is modified by multiplying its frequency content elementwise by $h(\lambda_n)$. These values represent the *graph frequency response* of the graph filter (12).

The relation (21) is a generalization of the classical *convolution theorem* [21] to graphs: filtering a graph signal in the graph domain is equivalent in the frequency domain to multiplying the signal's spectrum by the frequency response of the graph filter.

Low and High Frequencies on Graphs

In DSP, frequency contents of time series and digital images are described by complex or real sinusoids that oscillate at different rates [24]. These rates provide an intuitive, physical interpretation of "low" and "high" frequencies: low-frequency components oscillate less and high-frequency ones oscillate more.

In analogy to DSP, frequency components on graphs can also be characterized as "low" and "high" frequencies. In particular, this is achieved by ordering the graph frequency components according to how much they change across the graph; that is, how much the signal coefficients of a frequency component differ at connected nodes. The amount of "change" is calculated using the *graph total variation* [18]. For graphs with real spectra, the ordering from lowest to highest frequencies is $\lambda_0 \ge \lambda_1 \ge \ldots \ge \lambda_{N-1}$. For graphs with complex spectra, frequencies are ordered by their distance from the point $|\lambda_{\text{max}}|$ on the complex plane, where λ_{max} is the eigenvalue with the largest magnitude. The graph frequency order naturally leads to the definition of low-, high-, and band-pass graph filters, analogously to their counterparts in DSP (see [18] for details).

In the special case of undirected graphs with real non-negative edge weights, the graph Fourier transform (18) can also be expressed using the eigenvectors of the graph Laplacian matrix [16]. In general, the eigenvectors of the adjacency and Laplacian matrices do not coincide, which can lead to a different Fourier transform matrix. However, when graphs are regular, both definitions yield the same graph Fourier transform matrix, and the same frequency ordering [18].

Applications

DSP_G is particularly motivated by the need to extend traditional signal processing methods to datasets with complex and irregular structure. Problems in different domains can be formulated and solved as standard signal processing problems. Applications include data compression through Fourier transform or through wavelet expansions; recovery, denoising, and classification of data by signal regularization, by smoothing, or by adaptive filter design; anomaly detection via high-pass filtering; and many others (see [17], [18], [15], [16] and references therein).

For instance, a graph signal can be compressed by computing its graph Fourier transform and storing only a small fraction of

its spectral coefficients, the ones with largest magnitudes. The compressed signal is reconstructed by computing the inverse graph Fourier transform with the preserved coefficients. When the signal is sparse in the Fourier domain, that is, when most energy is concentrated in a few frequencies, the compressed signal is reconstructed with a small error [17], [25].

Another example application is detection of corrupted data. In traditional DSP, a corrupted value in a slowly changing time signal introduces additional high-frequency components that can be detected by high-pass filtering of the corrupted signal. Similarly, a corrupted value in a graph signal can be detected through a high-pass graph filter, which can be used, for instance, to detect malfunctioning sensors in sensor networks [18].

III. Challenges of Big Data

While there is no single, universally agreed upon set of properties that define Big Data, some of the commonly mentioned ones are *volume*, *velocity*, and *variety* of data [1]. Each of these characteristics poses a separate challenge to the design and implementation of analysis systems and algorithms for Big Data. First of all, the sheer volume of data to be processed requires efficient distributed and scalable storage, access, and processing. Next, in many applications new data is obtained continuously. High velocity of new data arrival demands fast algorithms to prevent bottlenecks and explosion of the data volume and to extract valuable information from the data and incorporate it into the decision-making process in real time. Finally, collected datasets contain information in all varieties and forms, including numerical, textual, and visual data. To generalize data analysis techniques to diverse datasets, we need a common representation framework for datasets and their structure.

The latter challenge of data diversity is addressed in DSP_G by representing dataset structure with graphs and quantifying data into graph signals. Graphs provide a versatile data abstraction for multiple types of data, including sensor network measurements, text documents, image and video databases, social networks, and others. Using this abstraction, data analysis methods and tools can be developed and applied to datasets of different nature.

For efficient Big Data analysis, the challenges of data volume and velocity must be addressed as well. In particular, the fundamental signal processing operations of filtering and spectral decomposition may be prohibitively expensive for large datasets both in the amount of required computations and memory demands.

Recall that processing a graph signal (1) with a graph filter (16) requires L multiplications by a $N \times N$ graph shift matrix $\mathbf{A}$. For a general matrix, this computation requires $O(LN^2)$ arithmetic operations (additions and multiplications) [26]. When $\mathbf{A}$ is sparse and has on average K nonzero entries in every row, graph filtering requires $O(LNK)$ operations. In addition, graph filtering also requires access to the entire graph signal in memory. Similarly, computation of the graph Fourier transform (18) requires $O(N^2)$ operations and access to the entire signal in memory. Moreover, the eigendecomposition of the matrix $\mathbf{A}$ requires additional $O(N^3)$ operations and memory access to the entire $N \times N$ matrix $\mathbf{A}$. Note that graph filtering can also be performed in the spectral domain with $O(N^2)$ operations using the graph convolution theorem (21), but it also requires the initial eigendecomposition of $\mathbf{A}$.

Degree heterogeneity in graphs with heavily skewed degree distributions, such as scale-free graphs, presents an additional challenge. Graph filtering (16) requires iterative weighted averaging over each vertex's neighbors, and for vertices with large degrees this process takes significantly longer than for vertices with small degrees. In this case, load balancing through smart distribution of vertices between computational nodes is required to avoid a computation bottleneck.

For very large datasets, algorithms with quadratic and cubic arithmetic cost are not acceptable. Moreover, computations that require access to the entire datasets are ill-suited for large data sizes and lead to performance bottlenecks, since memory access is orders of magnitude slower than arithmetic computations. This problem is exacerbated by the fact that large datasets often do not fit into main memory or even local disk storage of a single machine, and must be stored and accessed remotely and processed with distributed systems.

Fifty years ago, the invention of the famous fast Fourier transform algorithm by Cooley and Tukey [27], as well as many other algorithms that followed (see [28], [29] and references therein), dramatically reduced the computational cost of the discrete Fourier transform by using suitable properties of the structure of time signals, and made frequency analysis and filtering of very large signals practical. Similarly, in this article, we identify and discuss properties of certain data representation graphs that lead to more efficient implementations of DSP_G operations for Big Data. A suitable graph model is provided by product graphs discussed in the next Section.

IV. Product Graphs

Consider two graphs $G_1 = (\mathcal{V}_1, \mathbf{A}_1)$ and $G_2 = (\mathcal{V}_2, \mathbf{A}_2)$ with $|\mathcal{V}_1| = N_1$ and $|\mathcal{V}_2| = N_2$ nodes, respectively. The *product graph*, denoted by $\diamond$, of G_1 and G_2 is the graph

$$G = G_1 \diamond G_2 = (\mathcal{V}, \mathbf{A}_\diamond), \tag{22}$$

with $|\mathcal{V}| = N_1 N_2$ nodes and an appropriately defined $N_1 N_2 \times N_1 N_2$ adjacency matrix $\mathbf{A}_\diamond$ [30], [31]. In particular, three commonly studied graph products are the Kronecker, Cartesian, and strong products.

For the *Kronecker* graph product, denoted as $G = G_1 \otimes G_2$, the adjacency matrix is obtained by the matrix Kronecker product of adjacency matrices $\mathbf{A}_1$ and $\mathbf{A}_2$:

$$\mathbf{A}_\otimes = \mathbf{A}_1 \otimes \mathbf{A}_2. \tag{23}$$

Recall that the *Kronecker* product of matrices $\mathbf{B} = [b_{mn}] \in \mathbb{C}^{M \times N}$ and $\mathbf{C} \in \mathbb{C}^{K \times L}$ is a $KM \times LN$ matrix with block structure

$$\mathbf{B} \otimes \mathbf{C} = \begin{bmatrix} b_{0,0}\,\mathbf{C} & \dots & b_{0,N-1}\,\mathbf{C} \\ \vdots & \vdots & \vdots \\ b_{M-1,0}\,\mathbf{C} & \dots & b_{M-1,N-1}\,\mathbf{C} \end{bmatrix}. \tag{24}$$

For the *Cartesian* graph product, denoted as $G = G_1 \times G_2$, the adjacency matrix is

$$\mathbf{A}_{\times} = \mathbf{A}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{A}_2. \tag{25}$$

Finally, for the *strong* product, denoted as $G = G_1 \boxtimes G_2$, the adjacency matrix is

$$\mathbf{A}_{\boxtimes} = \mathbf{A}_1 \otimes \mathbf{A}_2 + \mathbf{A}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{A}_2. \tag{26}$$

The strong product can be seen as a combination of the Kronecker and Cartesian products. Since the products (23), (25), and (26) are associative, Kronecker, Cartesian, and strong graph products can be defined for an arbitrary number of graphs.

Product graphs arise in different applications, including signal and image processing [32], computational sciences and data mining [33], and computational biology [34]. Their probabilistic counterparts are used in network modeling and generation [35], [36], [37]. Multiple approaches have been proposed for the decomposition and approximation of graphs with product graphs [38], [30], [31], [39].

Product graphs offer a versatile graph model for the representation of complex datasets in multi-level and multi-parameter ways. In traditional DSP, multi-dimensional signals, such as digital images and video, reside on rectangular lattices that are Cartesian products of line graphs. Fig. 2(a) shows a two-dimensional lattice formed by the Cartesian product of two one-dimensional lattices.

Another example of graph signals residing on product graphs is data collected by a sensor network over a period of time. In this case, the graph signal formed by measurements of all sensors at all time steps resides on the product of the sensor network graph with the time series graph. As the example in Fig. 2(b) illustrates, the kth measurement of the nth sensor is indexed by the nth node of the kth copy of the sensor graph (or, equivalently, the kth node of the nth copy of the time series graph). Depending on the choice of product, a measurement of a sensor is related to the measurements collected by this sensor and its neighbors at the same time and previous and following time steps. For instance, the strong product in Fig. 2(b) relates the measurement of the nth sensor at time step k to its measurements at time steps $k-1$ and $k+1$, as well as to measurements of its neighbors at times $k-1$, k, and $k+1$.

A social network with multiple communities also may be representable by a graph product. Fig. 2(c) shows an example of a social network that has three communities with similar structures, where individuals from different communities also interact with each other. This social graph may be seen as an approximation of the Cartesian product of the graph that captures the community structure and the graph that captures the interaction between communities.

Other examples where product graphs are potentially useful for data representation include multi-way data arrays that contain elements described by multiple features, parameters, or characteristics, such as publications in citation databases described by their topics, authors, and venues; or internet connections described by their time, location, IP address, port accesses, and other parameters. In this case, the graph factors in (22) represent similarities or dependencies between subsets of characteristics.

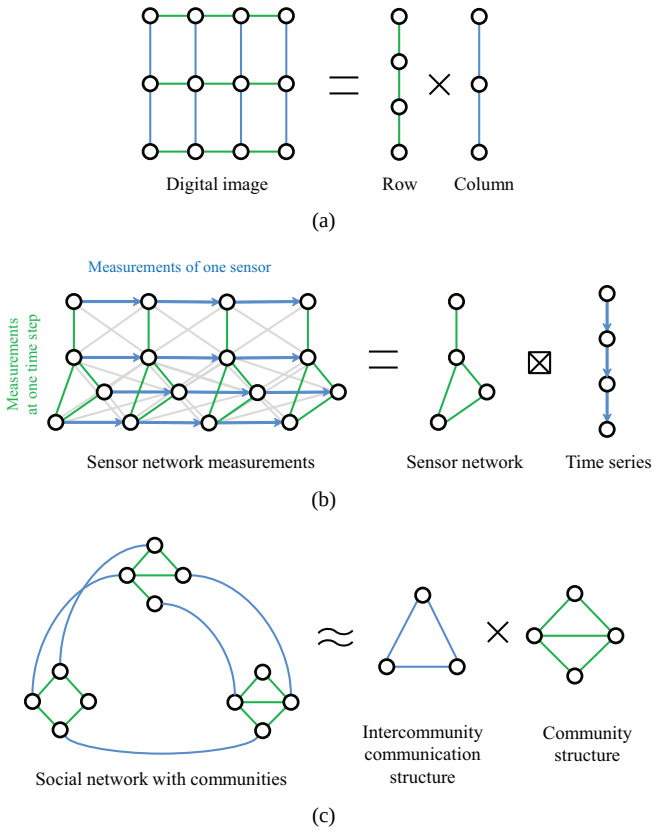


Fig. 2. Examples of product graphs indexing various data: a) Digital images reside on rectangular lattices that are Cartesian products of line graphs for rows and columns; b) Measurements of a sensor network are indexed by the strong product of the sensor network graph with the time series graph (edges of the Cartesian product are shown in blue and green, and edges of the Kronecker product are shown in grey; the strong product contains all edges); c) A social network with three similar communities is approximated by a Cartesian product of the community structure graph with the intercommunity communication graph.

Graph products are also used for modeling entire graph families. Kronecker products of scale-free graphs with the same degree distribution are also scale-free and have the same distribution [40], [35]. K- and ϵ-nearest neighbor graphs, which are used in signal processing, communications and machine learning to represent spatial and temporal location of data, such as sensor networks and image pixels, or data similarity structure, can be approximated with graph products, as the examples in Figs. 2(a) and 2(b) suggest. Other graph families, such as trees, are constructed using rooted graph products [41], which are not discussed in this article.

V. Signal Processing on Product Graphs

In this Section, we discuss how product graphs help "modularize" the computation of filtering and Fourier transform on graphs and improve algorithms, data storage and memory access for large datasets. They lead to graph filtering and Fourier transform implementations suitable for multi-core and clustered platforms with distributed storage by taking advantage of such performance optimization techniques as parallelization and vectorization. The presented results illustrate how product graphs offer a suitable and practical model for constructing

and implementing signal processing methodologies for large datasets. In this, product graphs are similar to other graph families, such as scale-free and small-world graphs, that are used to model properties of real-world graphs and datasets: while models do not fit exactly to all real-world graphs, they capture and abstract relevant representations of graphs and facilitate their analysis and processing.

Filtering

Recall that graph filtering is computed as the multiplication of a graph signal (1) by a filter (16). As we discussed in Section III, computation of a filtered signal requires repeated multiplications by the shift matrix, which is in general a computation- and memory-expensive operation for very large datasets.

Now, consider, for instance, a Cartesian product graph with the shift matrix (25). A graph filter of the form (16) for this graph is written as

$$h(\mathbf{A}_\times) = h_L \prod_{\ell=0}^{L-1} \left(\mathbf{A}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{A}_2 - g_\ell \, \mathbf{I}_{N_1 N_2}\right). \quad (27)$$

Hence, multiplication by the shift matrix $\mathbf{A}_\times$ is replaced with multiplications by matrices $\mathbf{A}_1 \otimes \mathbf{I}_{N_2}$ and $\mathbf{I}_{N_1} \otimes \mathbf{A}_2$.

Multiplication by matrices of the form $\mathbf{I}_{N_1} \otimes \mathbf{A}_2$ and $\mathbf{A}_1 \otimes \mathbf{I}_{N_2}$ have multiple efficient implementations that take advantage of modern optimization and high-performance techniques, such as parallelization and vectorization [26], [42], [43]. In particular, the product $(\mathbf{I}_{N_1} \otimes \mathbf{A}_2)\mathbf{s}$ is calculated by multiplying N_1 signal segments $\mathbf{s}_{n,\ldots,n+N_2}$, $0 \le n < N_1$, of length N_2 by the matrix $\mathbf{A}_2$. These products are computed with independent parts of the input signal, which eliminates data dependency and makes these operations highly suitable for a *parallel* implementation on a multicore or cluster platform [42]. As an illustration, for $N_1 = 3$, $N_2 = 2$, matrix

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad (28)$$

and a signal $\mathbf{s} \in \mathbb{C}^6$, we obtain

$$\Big(\mathbf{I}_3 \otimes \mathbf{A}\Big)\mathbf{s} = \begin{bmatrix} \mathbf{A} & & \\ & \mathbf{A} & \\ & & \mathbf{A} \end{bmatrix} \mathbf{s} = \begin{bmatrix} \mathbf{A} \begin{bmatrix} s_0 \\ s_1 \end{bmatrix} \\ \mathbf{A} \begin{bmatrix} s_2 \\ s_3 \end{bmatrix} \\ \mathbf{A} \begin{bmatrix} s_4 \\ s_5 \end{bmatrix} \end{bmatrix}.$$

Here, all multiplications by $\mathbf{A}$ are independent from each other both in data access and computations.

Similarly, the product $(\mathbf{A}_1 \otimes \mathbf{I}_{N_2})\mathbf{s}$ is calculated by multiplying N_2 segments $\mathbf{s}_{n,n+N_1,\ldots,n+(N_2-1)N_1}$, $0 \le n < N_2$, of the input signal by the matrix $\mathbf{A}_1$. These products are highly suitable for a *vectorized* implementation, available on modern computational platforms, that performs an operation on several input values simultaneously [42]. For instance, for $\mathbf{A}$ in (28), we obtain

$$\Big(\mathbf{A} \otimes \mathbf{I}_3\Big)\mathbf{s} = \begin{bmatrix} a_{00} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} + a_{01} \begin{bmatrix} s_3 \\ s_4 \\ s_5 \end{bmatrix} \\ a_{10} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} + a_{11} \begin{bmatrix} s_3 \\ s_4 \\ s_5 \end{bmatrix} \end{bmatrix}.$$

Here, three sequential signal values are multiplied by one element of matrix $\mathbf{A}$ at the same time. These operations are performed simultaneously by a processor with vectorization capabilities, which respectively decreases the computation time by a factor of three.

In addition to its suitability for parallelized and vectorized implementations, computing the output of the filter (27) on a Cartesian graph also requires significantly fewer operations, since the multiplication by the shift matrix (25) requires N_1 multiplications by an $N_2 \times N_2$ matrix and N_2 multiplications by an $N_1 \times N_1$ matrix, which results in $O(N_1 N_2^2) + O(N_1^2 N_2) = O(N(N_1 + N_2))$ operations rather than $O(N^2)$. For example, when $N_1, N_2 \approx \sqrt{N}$, this represents a reduction of the computational cost of graph filtering by a factor $\sqrt{N}$. To put this into the Big Data perspective, for a graph with a million vertices, the cost of filtering is reduced by a factor of 1000, and for a graph with a billion vertices, the cost reduction factor is more than 30000.

We discuss here operation counts for general graphs with full matrices. In practice, adjacency matrices are often sparse, and their multiplication requires fewer operations. Computational savings provided by product graphs are, likewise, significant for sparse adjacency matrices.

Furthermore, the multiplication by a matrix of the form $\mathbf{I} \otimes \mathbf{A}$ can be replaced by the multiplication with a matrix $\mathbf{A} \otimes \mathbf{I}$ with no additional arithmetic operations by suitable permutation of signal values [22], [42], [43]. This interchangeability leads to a selection between parallelized and vectorized implementations and provides means to efficiently compute graph filtered signals on platforms with arbitrary number of cores and vectorization capabilities.

The advantages of filtering on Cartesian product graphs also apply to Kronecker and strong product graphs. In particular, using the property [22]

$$\mathbf{A}_1 \otimes \mathbf{A}_2 = (\mathbf{A}_1 \otimes \mathbf{I}_{N_2})(\mathbf{I}_{N_1} \otimes \mathbf{A}_2), \quad (29)$$

we write the graph filter (16) for the Kronecker product as

$$h(\mathbf{A}_\otimes) = h_L \prod_{\ell=0}^{L-1} \Big((\mathbf{A}_1 \otimes \mathbf{I}_{N_2})(\mathbf{I}_{N_1} \otimes \mathbf{A}_2) - g_\ell \, \mathbf{I}_{N_1 N_2}\Big),$$

and for the strong product as

$$\begin{aligned} h(\mathbf{A}_\boxtimes) \;=\; & h_L \prod_{\ell=0}^{L-1} \Big((\mathbf{A}_1 \otimes \mathbf{I}_{N_2})(\mathbf{I}_{N_1} \otimes \mathbf{A}_2) \\ & + \mathbf{A}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{A}_2 - g_\ell \, \mathbf{I}_{N_1 N_2}\Big). \end{aligned}$$

Similarly to (27), these filters multiply input signals by matrices $\mathbf{I}_{N_1} \otimes \mathbf{A}_2$ and $\mathbf{A}_1 \otimes \mathbf{I}_{N_2}$ and are implementable using parallelization and vectorization techniques. They also lead to substantial reductions of the number of required computations.

Fourier Transform

The frequency content of a graph signal is computed through the graph Fourier transform (18). In general, this procedure has the computational cost of $O(N^2)$ operations and requires access to the entire signal in memory. Moreover, it also requires a preliminary calculation of the eigendecomposition of the graph shift matrix $\mathbf{A}$, which, in general, takes $O(N^3)$ operation.

Let us consider a Cartesian product graph with the shift matrix (25). Assume that the eigendecomposition (17) of the matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ is respectively $\mathbf{A}_i = \mathbf{V}_i \mathbf{\Lambda}_i \mathbf{V}_i^{-1}$, $i \in \{1, 2\}$, where $\mathbf{\Lambda}_i$ has eigenvalues $\lambda_{i,0}, \ldots, \lambda_{i,N-1}$ on the main diagonal. Similar results can be obtained for non-diagonalizable matrices using Jordan decomposition. The derivation is more involved, and we omit it for simplicity of discussion.

If we denote $\mathbf{V} = \mathbf{V}_1 \otimes \mathbf{V}_2$, then the eigendecomposition of the shift matrix (25) is [22]

$$\mathbf{A}_\times = \mathbf{V}(\mathbf{\Lambda}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{\Lambda}_2)\, \mathbf{V}^{-1}. \qquad (30)$$

Hence, the graph Fourier transform associated with a Cartesian product graph is given by the matrix Kronecker product of the graph Fourier transforms for its factor graphs:

$$\mathbf{F}_\times = (\mathbf{V}_1 \otimes \mathbf{V}_2)^{-1} = \mathbf{V}_1^{-1} \otimes \mathbf{V}_2^{-1} = \mathbf{F}_1 \otimes \mathbf{F}_2, \qquad (31)$$

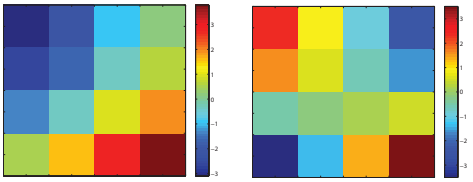
and the spectrum is given by the element-wise summation of the spectra of the smaller graphs: $\lambda_{1,n} + \lambda_{2,m}$, $0 \le n < N_1$ and $0 \le m < N_2$.

Reusing the property (29), (31) can be written as $\mathbf{F}_\times = \mathbf{F}_1 \otimes \mathbf{F}_2 = (\mathbf{F}_1 \otimes \mathbf{I}_{N_2})(\mathbf{I}_{N_1} \otimes \mathbf{F}_2)$ and efficiently implemented using parallelization and vectorization techniques. Moreover, the computation of the eigendecomposition (30) is replaced with finding the eigendecomposition of the shift matrices $\mathbf{A}_1$ and $\mathbf{A}_2$, which reduces the computation cost from $O(N^3)$ to $O(N_1^3 + N_2^3)$. For instance, when $N_1, N_2 \approx \sqrt{N}$, the computational cost of the eigendecomposition is reduced by a factor $N\sqrt{N}$. Hence, for a graph with a million vertices, the cost of computing the eigendecomposition is reduced by a factor of more than 3×10^4, and for a graph with a billion vertices, the cost reduction factor is over 3×10^{13}.

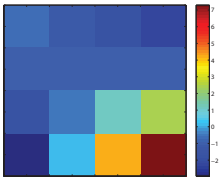
The same improvements apply to the Kronecker and strong matrix products, since the eigendecomposition of the corresponding shift matrices is

$$\begin{aligned} \mathbf{A}_\otimes &= \mathbf{V}(\mathbf{\Lambda}_1 \otimes \mathbf{\Lambda}_2)\, \mathbf{V}^{-1}, \\ \mathbf{A}_\boxtimes &= \mathbf{V}(\mathbf{\Lambda}_1 \otimes \mathbf{I}_{N_2} + \mathbf{I}_{N_1} \otimes \mathbf{\Lambda}_2 + \mathbf{\Lambda}_1 \otimes \mathbf{\Lambda}_2)\, \mathbf{V}^{-1}. \end{aligned}$$

Observe that all three graph products have the same graph Fourier transform. However, the corresponding spectra are different: for Cartesian and strong products, they are, respectively, $\lambda_{1,n}\lambda_{2,m}$ and $\lambda_{1,n}\lambda_{2,m} + \lambda_{1,n} + \lambda_{2,m}$, where $0 \le n < N_1$ and $0 \le m < N_2$. Thus, while all three graph products have the same frequency components, the ordering of these components from lowest to highest frequencies, as defined by DSP_G and discussed in Section II, can be different. As an illustration, consider the example in Fig. 3. It shows the frequencies (eigenvalues) of the three graph products in Fig. 2(b). All product graphs have the same 16 frequency components (eigenvectors), but the frequencies (eigenvalues) corresponding to these components are different and on each graph have a different interpretation as low or high frequency. For example, the values in the upper left corner of Figs. 3(a), 3(b), and 3(c) correspond to the same frequency component. By comparing these values, we observe that this component represents the highest frequency in the Cartesian product graph, the lowest frequency in the Kronecker product graph, and a mid-spectrum component in the strong product graph.



(a) Cartesian product (b) Kronecker product



(c) Strong product

Fig. 3. Frequency values for the product graphs in Fig. 2(b). Frequencies are shown as a color-coded 2D map, with x- and y-axis representing frequencies of two factor graphs. Higher values correspond to lower frequencies, and vice versa.

Fast Graph Fourier Transforms: A major motivation behind the use of product graphs in signal processing and DSP_G is derivation of fast computational algorithms for the graph Fourier transform. A proper overview of this topic requires an additional discussion of graph concepts and an algebraic approach to fast algorithms [44], [29], [45] that are beyond the scope of this article.

As an intuitive example, consider a well-known and widely used decimation-in-time fast Fourier transform for power-of-two sizes [27]. It is derived using graph products as follows. We view the $\mathbf{DFT}_N$ as the graph Fourier transform of a graph with adjacency matrix $\mathbf{C}^2$, where $\mathbf{C}$ is the cyclic shift matrix (3). This is a valid algebraic assumption, since the $\mathbf{DFT}_N$ is a graph Fourier transform not only for the graph in Fig. 1(a), but for any graph with adjacency matrix given by a polynomial $h(\mathbf{C})$. This graph, after a permutation of its vertices at stride two (which represents the decimation-in-time step), becomes a product of a cyclic graph with $N/2$ vertices with a graph of two disconnected vertices. As a result, its graph Fourier transform $\mathbf{DFT}_N$ becomes a product $\mathbf{I}_2 \otimes \mathbf{DFT}_{N/2}$ and additional, sparse matrices that capture the operations of graph restructuring. By continuing this process recursively for $\mathbf{DFT}_{N/2}$, $\mathbf{DFT}_{N/4}$, and so forth, we decompose $\mathbf{DFT}_N$ into a product of sparse matrices with cumulative arithmetic cost of $O(N \log N)$, thus obtaining a fast algorithm for the computation of $\mathbf{DFT}_N$.

VI. Relation to Existing Approaches

The instantiation of DSP_G for product graphs relates to existing approaches to complex data analysis that are not based

on graphs but rather view data as multi-dimensional arrays [2], [3], [4]. Given a K-dimensional dataset $\mathbf{S} \in \mathbb{C}^{N_1 \times N_2 \times \ldots \times N_K}$, the family of methods called *canonical decomposition* or *parallel factor analysis* searches for K matrices $\mathbf{M}_k \in \mathbb{C}^{N_k \times R}$, $1 \leq k \leq K$, that provide an optimal approximation of the dataset

$$\mathbf{S} = \sum_{r=1}^{R} \mathbf{m}_{1,r} \circ \mathbf{m}_{2,r} \circ \ldots \circ \mathbf{m}_{K,r} + \mathbf{E}, \qquad (32)$$

that minimizes the error

$$|| \, \mathbf{E} \, || = \sqrt{\sum_{n_1=1}^{N_1} \ldots \sum_{n_k=1}^{N_K} | \, \mathbf{E}_{n_1,n_2,\ldots,n_K} \, |^2}.$$

Here, $\mathbf{m}_{k,r}$ denotes the rth column of matrix $\mathbf{M}_k$, and $\circ$ denotes the outer product of vectors.

A more general approach, called *Tucker decomposition*, searches for K matrices $\mathbf{M}_k \in \mathbb{C}^{N_k \times R_k}$, $1 \leq k \leq K$, and a matrix $\mathbf{C} \in \mathbb{C}^{R_1 \times R_2 \times \ldots \times R_K}$ that provide an optimal approximation of the dataset as

$$\mathbf{S} = \sum_{r_1=1}^{R_1} \ldots \sum_{r_K=1}^{R_K} \mathbf{C}_{r_1,\ldots,r_K} \, \mathbf{m}_{1,r_1} \circ \ldots \circ \mathbf{m}_{K,r_K} + \mathbf{E} \, . \qquad (33)$$

Tucker decomposition is also called a higher-order PCA or SVD, since it effectively extends these techniques from matrices to higher-order arrays.

Decompositions (32) and (33) can be interpreted as signal compression on product graphs. For simplicity of discussion, assume that $K = 2$ and consider a signal $\mathbf{s} \in \mathbb{C}^{N_1 N_2}$ that lies on a product graph (22) and corresponds to a 2-dimensional signal $\mathbf{S} \in \mathbb{C}^{N_1 \times N_2}$, so that $\mathbf{S}_{n_1,n_2} = \mathbf{s}_{n_1 N_2 + n_2}$, where $0 \leq n_i < N_i$ for $i = 1, 2$. If matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ contain as columns, respectively, R_1 and R_2 eigenvectors of $\mathbf{A}_1$ and $\mathbf{A}_2$, then the decomposition (33) represents a lossy compression of the graph signal in the frequency domain, a widely used compression technique in signal processing [21], [24].

VII. Example Application

As a motivational application example of DSP_G on product graphs, we consider data compression. For the testing dataset, we use the set of daily temperature measurements collected by 150 weather stations across the United States [17] during the year 2002. Fig. 1(b) shows the measurements from one day (December 01, 2002), as well as the sensor network graph. The graph is constructed by connecting each sensor to 8 of its nearest neighbors with undirected edges with weights given by Eq. (29) in [17]. As illustrated by the example in Fig. 2(b), such dataset can be described by a product of the sensor network graph and the time series graphs. We use the sensor network graph in Fig. 1(b) with $N_1 = 150$ nodes and the time series graph in Fig. 1(a) with $N_2 = 365$ nodes.

The compression is performed in the frequency domain. We compute the Fourier transform (31) of the dataset, keep only C spectrum coefficients with largest magnitudes and replace others with zeros, and perform the inverse graph Fourier transform on the resulting coefficients. This is a lossy compression scheme, with the compression error given by the norm of the difference between the original dataset and the reconstructed one normalized by the norm of the original dataset. Note that, while the approach is tested here on a relatively small dataset, it is applicable in the same form to arbitrarily large datasets.

	Fraction of coefficients used (C/N)						
	$1/50$	$1/20$	$1/15$	$1/10$	$1/7$	$1/5$	$1/3$
Error (%)	4.9	3.5	3.1	2.6	2.1	1.6	0.7
PSNR (dB)	71.2	74.1	75.1	76.7	78.5	80.9	87.1

TABLE I
ERRORS INTRODUCED BY COMPRESSION OF THE TEMPERATURE DATA.

The compression errors for the considered temperature dataset are shown in Table I. The results demonstrate that even for high compression ratios, that is, when the number C of stored coefficients is much smaller than the dataset size $N = N_1 N_2$, the compression introduces only a small error and leads to insignificant loss of information. A comparison of this approach with schemes that compress the data only in one dimension (they separately compress either time series from each sensor or daily measurements from all sensors) [17], [25] also reveals that compression based on the product graph is significantly more efficient.

VIII. Conclusions

In this article, we presented an approach to Big Data analysis based on the discrete signal processing on graphs. We reviewed fundamental concepts of the framework and illustrated how it extends traditional signal processing theory to datasets represented with general graphs. To address important challenges in Big Data analysis and make implementations of fundamental DSP_G techniques suitable for very large datasets, we considered a generalized graph model given by several kinds of product graphs, including the Cartesian, Kronecker, and strong product graphs. We showed that these product graph structures significantly reduce arithmetic cost of associated DSP_G algorithms and make them suitable for parallel and distributed implementation, as well as improve memory storage and access of data. The discussed methodology bridges a gap between signal processing, Big Data analysis, and high-performance computing, as well as presents a framework for the development of new methods and tools for analysis of massive datasets.

IX. Authors

Aliaksei Sandryhaila (asandryh@andrew.cmu.edu) received a B.S. degree in Computer Science from Drexel University, Philadelphia, PA, in 2005, and a Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, in 2010. He is currently a Research Scientist in the Department of Electrical and Computer Engineering at CMU. His research interests include big data analysis, signal processing, machine learning, design and optimization of algorithms and software, and high-performance computing.

José M. F. Moura (moura@ece.cmu.edu) is the Philip and Marsha Dowd University Professor at CMU. In 2013-14, he is

a Visiting Professor at NYU with the Center for Urban Science and Progress (CUSP). He holds degrees from IST (Portugal) and MIT where he has been a visiting professor. At CMU, he manages the CMU/Portugal Program (www.cmuportugal.org). His interests are in signal processing and data science. He was an IEEE Board Director, President of the IEEE SP Society (SPS), and Editor-in-Chief for the Transactions on SP. He received the IEEE SPS Technical Achievement Award and the IEEE SPS Society Award , he is a Fellow of the IEEE, of the AAAS, a corresponding member of the Academy of Sciences of Portugal, and a member of the US National Academy of Engineering.

REFERENCES

[1] P. Zikopoulos, D. deRoos, and K. P. Corrigan, *Harness the Power of Big Data*, McGraw-Hill, 2012.

[2] M. W. Mahoney, M. Maggoni, and P. Drineas, "Tensor-CUR decompositions for tensor-based data," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 957–987, 2008.

[3] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[4] E. Acar and B. Yener, "Unsupervised multiway data analysis: A literature survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 21, no. 1, pp. 6–20, 2009.

[5] A. H. Andersen and W. S. Rayens, "Structure-seeking multilinear methods for the analysis of fMRI data," *Neuroimage*, vol. 22, no. 2, pp. 728–739, 2004.

[6] F. Miwakeichi, E. Martinez-Montes, P. A. Valdes-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi, "Decomposing EEG data into space-time-frequency components using parallel factor analysis," *Neuroimage*, vol. 22, no. 3, pp. 1035–1045, 2004.

[7] N. D. Sidiropoulos, G. B. Giannakis, and R. Bro, "Blind PARAFAC receivers for DS-CDMA systems," *IEEE Trans. Signal Proc.*, vol. 48, no. 3, pp. 810–823, 2000.

[8] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, "Parallel factor analysis in sensor array processing," *IEEE Trans. Signal Proc.*, vol. 48, no. 8, pp. 2377–2388, 2000.

[9] L. De Lathauwer and J. Castaing, "Blind identification of underdetermined mixtures by simultaneous matrix diagonalization," *IEEE Trans. Signal Proc.*, vol. 56, no. 3, pp. 1096–1105, 2008.

[10] J. F. Tenenbaum, V. Silva, and J. C. Langford, "A global geometric framework for nonlinear dmensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[11] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[12] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comp.*, vol. 15, no. 6, pp. 1373–1396, 2003.

[13] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *Proc. Nat. Acad. Sci.*, vol. 100, no. 10, pp. 5591–5596, 2003.

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

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

[16] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs," *IEEE Signal Proc. Mag.*, vol. 30, no. 3, pp. 83–98, 2013.

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

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

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

[20] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: 1-D space," *IEEE Trans. Signal Proc.*, vol. 56, no. 8, pp. 3586–3599, 2008.

[21] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, 2nd edition, 1999.

[22] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, Academic Press, 2nd edition, 1985.

[23] A. Sandryhaila and J. M. F. Moura, "Classification via regularization on graphs," in *IEEE Global Conf. Signal and Inf. Proc.*, 2013, pp. 495–498.

[24] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 3rd edition, 2008.

[25] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs: Graph Fourier transform," in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Proc.*, 2013, pp. 6167–6170.

[26] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, 3rd edition, 1996.

[27] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[28] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and a state of the art," *J. Signal Proc.*, vol. 19, no. 4, pp. 259–299, 1990.

[29] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: Cooley-Tukey type algorithms for DCTs and DSTs," *IEEE Trans. Signal Proc.*, vol. 56, no. 4, pp. 1502–1521, 2008.

[30] W. Imrich, S. Klavzar, and D. F. Rall, *Topics in Graph Theory: Graphs and Their Cartesian Product*, CRC Press, 2008.

[31] R. Hammack, W. Imrich, and S. Klavzar, *Handbook of Product Graphs*, CRC Press, 2nd edition, 2011.

[32] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Prentice Hall, 1983.

[33] E. Acar, R. J. Harrison, F. Olken, O. Alter, M. Helal, L. Omberg, B. Bader, A. Kennedy, H. Park, Z. Bai, D. Kim, R. Plemmons, G. Beylkin, T. Kolda, S. Ragnarsson, L. Delathauwer, J. Langou, S. P. Ponnapalli, I. Dhillon, L. Lim, J. R. Ramanujam, C. Ding, M. Mahoney, J. Raynolds, L. Elden, C. Martin, P. Regalia, P. Drineas, M. Mohlenkamp, C. Faloutsos, J. Morton, B. Savas, S. Friedland, L. Mullin, and C. Van Loan, "Future directions in tensor-based computation and modeling," NSF Workshop Report, Arlington, VA, February 2009.

[34] M. Hellmuth, D. Merkle, and M. Middendorf, "Extended shapes for the combinatorial design of RNA sequences," *Int. J. Comp. Biology and Drug Design*, vol. 2, no. 4, pp. 371–384, 2009.

[35] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, pp. 985–1042, 2010.

[36] S. Moreno, S. Kirshner, J. Neville, and S. Vishwanathan, "Tied Kronecker product graph models to capture variance in network populations," in *Proc. Allerton Conf. Comm. Control Comput.*, 2010, pp. 1137–1144.

[37] S. Moreno, J. Neville, and S. Kirshner, "Learning mixed Kronecker product graph models with simulated method of moments," in *Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining*, 2013, pp. 1052–1060.

[38] C. Van Loan and N. Pitsianis, "Approximation with Kronecker products," in *Linear Algebra for Large Scale and Real Time Applications*, 1993, pp. 293–314.

[39] M. Hellmuth, W. Imrich, and T. Kupka, "Partial star products: A local covering approach for the recognition of approximate Cartesian product graphs," *Math. Comp. Sci.*, 2013, in print.

[40] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using Kronecker multiplication," in *Proc. Int. Conf. Mach. Learn.*, 2007, pp. 497–504.

[41] C. D. Godsil and B. D. McKay, "A new graph product and its spectrum," *Bull. Austral. Math. Soc.*, vol. 18, no. 1, pp. 21–28, 1978.

[42] F. Franchetti, M. Püschel, Y. Voronenko, S. Chellappa, and J. M. F. Moura, "Discrete Fourier transform on multicore," *IEEE Signal Proc. Mag.*, vol. 26, no. 6, pp. 90–102, 2009.

[43] Y. Voronenko, F. de Mesmay, and M. Püschel, "Computer generation of general size linear transform libraries," in *IEEE Int. Symp. Code Gen. Opt.*, 2009, pp. 102–113.

[44] M. Püschel and J. M. F. Moura, "The algebraic approach to the discrete cosine and sine transforms and their fast algorithms," *SIAM J. Comp.*, vol. 32, no. 5, pp. 1280–1316, 2003.

[45] A. Sandryhaila, J. Kovacevic, and M. Püschel, "Algebraic signal processing theory: Cooley-Tukey type algorithms for polynomial transforms based on induction," *SIAM J. Matrix Analysis and Appl.*, vol. 32, no. 2, pp. 364–384, 2011.