$$= [s_0 \;\; s_1 \;\; \ldots \;\; s_{N-1}]^T \in \mathbb{C}^N. \tag{1}$$

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.

Figure 1 shows examples of graph signals. Finite periodic time series, studied by finite-time DSP [19], [21], are indexed by directed cyclic graphs, such as the graph in Figure 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 Figure 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, Web site features (topic, view count, relevance) are graph signals indexed by graphs formed by hyperlink references, such as the graph in Figure 1(c). Each node represents a Web site, 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 Figure 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 nontrivial operation performed on a signal. A delayed finite periodic time series of length N is $\tilde{s}_n = s_{n-1 \bmod N}$. Using the vector notation (1), the shifted signal is written as

$$= [s_0 \;\; \ldots \;\; s_{N-1}]^T = \quad , \tag{2}$$

where $\mathbf{C}$ is the $N \times N$ cyclic shift matrix (only nonzero 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 Figure 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:

.

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

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

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 $s(z^{-1}) = \sum_{n=0}^{N-1} s_n b_n(z^{-1})$,

graph Fourier basis, such as singular vectors or eigenvectors of the Laplacian matrix. These choices are consistent with DSP_G,

generalize data analysis techniques to diverse data sets, we need a common representation framework for data sets and their structure.

The latter challenge of data diversity is addressed in DSP_G by representing data set 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 data sets of a 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 data sets 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 data sets, algorithms with quadratic and cubic arithmetic cost are not acceptable. Moreover, computations that require access to the entire data sets 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 data sets 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] and [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.

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 $N_1N_2 \times N_1N_2$ 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

$$\otimes \quad \begin{matrix} b_{0,0}\mathbf{C} & \cdots & b_{0,N-1}\mathbf{C} \\ \vdots & & \vdots \\ b_{M-1,0}\mathbf{C} & \cdots & b_{M-1,N-1}\mathbf{C} \end{matrix}.$$

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

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

Multiplication by matrices of the form $\ _{N_1} \otimes \mathbf{A}_2$ and $\mathbf{A}_1 \otimes \ _{N_2}$ 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 $(\ _{N_1} \otimes \mathbf{A}_2)_{\ }$

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:

$$\mathcal{F}_{\times} = (\mathcal{F}_1 \otimes \mathcal{F}_2)^{-1} = \mathcal{F}_1^{-1} \otimes \mathcal{F}_2^{-1} = \mathcal{F}_1 \otimes \mathcal{F}_2, \tag{31}$$

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 $\mathcal{F}_{\times} = \mathcal{F}_1 \otimes \mathcal{F}_2 = (\mathcal{F}_1 \otimes I_{N_2})(I_{N_1} \otimes \mathcal{F}_2)$ 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

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 multidimensional arrays [2]–[4]. Given a K-dimensional data set $\mathcal{L} \in \mathbb{C}^{N_1 \times N_2 \times \ldots \times N_K}$, the family of methods called *canonical decomposition* or *parallel factor analysis* searches for K matrices $\mathcal{F}_k \in \mathbb{C}^{N_k \times R}$, $1 \leq k \leq K$, that provide an optimal approximation of the data set

$$= \sum_{r=1}^{R} \mathbf{m}_{1,r} \circ \mathbf{m}_{2,r} \circ \ldots \circ \mathbf{m}_{K,r} + \quad , \tag{32}$$

that minimizes the error

$$= \sqrt{\sum_{n_1=1}^{N_1} \cdots \sum_{n_K=1}^{N_K} \quad _{n_1,n_2,\ldots,n_K}{}^2}.$$

Here, $_{k,r}$ denotes the r column of matrix $\mathcal{F}_k$, and $\circ$ denotes the outer product of vectors.

A more general approach, called *Tucker decomposition*, searches for K matrices $\mathcal{F}_k \in \mathbb{C}^{N_k \times R_k}$, $1 \leq k \leq K$, 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 data set as

$$= \sum_{r_1=1}^{R_1} \cdots \sum_{r_K=1}^{R_K} \quad _{r_1,\ldots,r_K} \mathbf{m}_{1,r_1} \circ \ldots \circ \mathbf{m}_{K,r_K} + \quad . \tag{33}$$

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 $_{} \in \mathbb{C}^{N_1 N_2}$ that lies on a product graph (22) and corresponds to a 2-D signal $\mathcal{L} \in \mathbb{C}^{N_1 \times N_2}$, so that $_{n_1,n_2} = \quad _{n_1 N_2 + n_2}$, where $0 \leq n_i < N_i$ for $i = 1,2$. If matrices $\mathcal{F}_1$ and $\mathcal{F}_2$ 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].

EXAMPLE APPLICATION

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

The compression is performed in the frequency domain. We compute the Fourier transform (31) of the data set, 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 data set and the reconstructed one normalized by the norm of the original data set. Note that, while the