USC-SIPI Report #448

EFFICIENT TRANSFORMS FOR GRAPH SIGNALS WITH APPLICATIONS TO VIDEO CODING

By
Keng-Shih Lu

December 2020

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
USC Viterbi School of Engineering
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Suite 400
Los Angeles, CA 90089-2564 U.S.A.

EFFICIENT TRANSFORMS FOR GRAPH SIGNALS WITH APPLICATIONS TO VIDEO CODING

by

Keng-Shih Lu

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

December 2020

Dedication

To my family and my friends.

Acknowledgments

First, I would like to express my deepest gratitude and thanks to my advisor, Professor Antonio Ortega, for his continuous guidance and patience through the years I have been pursuing my Ph.D. degree. His expertise and inspiration has enlightened me for refining research ideas and approaches. It has been a pleasure and privilege to work with him. I owe thanks to Professor C.-C. Jay Kuo and Professor Ulrich Neumann for serving on my dissertation committee and giving me valuable feedbacks. I am very grateful to Professor Shrikanth Narayanan, Professor Richard Leahy, and Professor Yingying Fan for being members of my qualifying exam committee. I also thank all my teachers at USC for their instruction in a wide range of courses.

Many thanks to Dr. Debargha Mukherjee for his guidance and valuable discussions we had during my two summer internships at Google. I am also thankful to Dr. Yue Chen and Dr. Elliott Karpolovsky for their generous support and feedbacks on several collaborative projects with Google.

I would like to thank my labmates in the Signal Transformation, Analysis and Compression Group, especially Jiun-Yu (Joanne), Hilmi, Eduardo, Ajinkya, Pratyusha, and Sarath, for their collaboration and brainstorming, and for being great colleagues and friends. I also thank my housemates and all my friends who have been along with me during my Ph.D. study in LA, including Ting-Ru Lin, Po-Wen Chen, I-Hsiu (Albert) Kao, Po-Han Huang, Kuan-Wen Huang, Ming-Chun Lee, Chin-Chuan (Bill) Chang, and Kai-Che (Kevin) Lin.

Finally, special thanks to my parents and my two little sisters for their continuous supports, and for letting me pursue my dream. And most importantly, to my wife, Ashley, for her infinite love, companionship, and support. My Ph.D. journey would not have been so memorable and fruitful without her.

Contents

List of Tables

List of Figures

Abstract

Graph signal processing (GSP) extends tools of conventional signal processing to graph signals, i.e., structured data residing on an irregular domain. The graph Fourier transform (GFT), as an extension of the discrete Fourier transform, defines the notion of frequency on graph, and can be used as a tool for frequency analysis of graph signals. However, challenges are posed by the computational complexity of graph signal processing techniques. First, a high dimensional graph Fourier transform can be computationally expensive and there is no fast algorithms for general graphs. Second, in graph filtering, which is a crucial tool in many GSP applications, an efficient implementation without GFT computation is a challenging problem that receives a great amount of attention. Third, while graph-based transforms have been shown to enhance the coding efficiency in MSE-based video compression framework, this application in coding schemes based on perceptually-driven metrics remains an open problem.

With the aim of answering these questions, we propose several techniques in this work. First, we study algebraic properties of the graph Laplacian matrix that lead to fast implementations of the graph Fourier transform, based on which we can design fast algorithms when the graph satisfies certain symmetry or bipartition properties. Second, we propose data-driven approaches to obtain fast GFT solutions for efficient separable and non-separable transforms. Third, we extend the notion of lapped transform to a graph-based setting, and propose a generalized version of the well-known lapped orthogonal transform. Fourth, we explore sparse graph operators for the well-known discrete cosine transform (DCT) and discrete sine transform (DST), which lead us to an efficient DCT/DST graph filtering approach and a fast transform type selection scheme for video encoder. Finally, we apply irregularity-aware graph Fourier transform (IAGFT) to perceptual image coding, which gives promising experimental results in terms of a popular perceptual quality metric, structural similarity (SSIM).

Chapter 1

Introduction

In recent years, the advances in data storage, communication, and computing power have led to an explosion of data. The demand for data science tasks, such as clustering, classification, and data-driven decision making, has been rapidly increasing in various application fields, and for a wide variety of high dimensional data. In practice, for some applications such as sensor, social, and transportation networks, data of interest is usually irregularly structured. However, conventional signal processing tools, such as the discrete Fourier transform, are targeted at regularly spaced data, such as time series. Those signal processing tools cannot be easily applied to high dimensional data lying on an irregular domain.

Graphs offer the capability of representing relations among data samples. For example, in a sensor network, each sensor can be represented as a graph node, and the weight of the graph edge connecting two nodes can model the similarity (based on e.g., distance) between the associated sensors. In view of this convenience, graph signal processing (GSP) [90, 104, 114] has become an emerging research field, where signal processing techniques such as frequency analysis and filtering are extended to signals defined on graphs. In GSP, a data sample, represented as a vector, is called a graph signal. A graph signal is associated to a graph that models the underlying relations between elements of the signal, where each vertex corresponds to a sample and each edge represents the correlation between a pair of samples. Traditional signal processing techniques including the Fourier transform, filtering, convolution, and downsampling, can be extended for graph signals. In particular, the graph Fourier transform (GFT) is a generalization of the discrete Fourier transform. The GFT provides the notion of frequency for graph signals, based on which we can be perform transformations [20, 36, 106], frequency analysis [105], filtering [7], and sampling [1, 88] for graph signals, which lead to applications in sensor networks [21], image and video processing [11], machine learning [6], and so forth.

One drawback of graph signal processing techniques is this increased complexity compared to conventional signal processing methods. In particular, the graph Fourier transform in general does not have a fast algorithm. In addition, implementation of graph filters, which usually require GFT as a key building block, can be challenging when the graph size is large. To address these issues, the main focus of this thesis is on

efficient graph Fourier transforms and graph filtering methods. The scope of this thesis, as well as a summary of contributions, is presented as follows.

- *Fast GFTs* (Chapter 2): Unlike the discrete Fourier transform, which is fixed and has a well-known divide-and-conquer algorithm [15] known as the fast Fourier transform (FFT), the GFT is defined based on the graph structure, and in general does not have a fast algorithm. In this chapter, we investigate conditions on graph structure for a butterfly stage to be available in the associated GFT implementation. We show that such fast GFTs arise when the graph has certain symmetric or bipartite properties, and propose a graph decomposition approach to derive fast GFT algorithms. We provide several examples for such fast GFTs with their applications. We also compare the runtimes of these fast GFTs with respect to the matrix GFT as well as other approximate fast GFT techniques.

- *Data-driven fast GFTs for video coding* (Chapter 3): We consider various data-driven fast GFTs for video compression. In Section 3.1 we introduce a graph learning approach for a fast GFT within the framework of Chapter 2: given a particular butterfly stage, we learn a graph whose GFT has a fast algorithm with the specified butterfly stage. Several types of butterfly stages can be used for image and video coding applications. For example, symmetric line graph transforms (SLGT) can be applied to 1D pixel blocks (Section 3.2). In Section 3.3, we propose an approximation method to determine the butterfly stage for the problem formulated in Section 3.1. By learning the optimal parameters of graphs from data, we obtain a compression gain for both inter- and intra-predicted residual blocks in Section 3.4.

- *Lapped GFTs* (Chapter 4): When the number of nodes is large, applying a GFT on the entire graph may not be practical in terms of computational complexity. A more practical solution is to divide the signal into blocks, and to apply the GFT block-wise. However, this does not take into account the correlations across the block boundaries, leading to the so-called blocking artifact. In conventional signal processing, the lapped transforms [81] was proposed to eliminate blocking artifacts of block-based discrete cosine transforms (DCT). In this thesis, we extend the notion of lapped transforms to graph signals. We show that lapped graph Fourier transforms (LGFT) can be obtained and implemented under a generalized framework of the lapped orthogonal transform design [81]. Using a piecewise smooth image as an example, we show that the lapped transform leads to a compression gain compared to DCT and the lapped orthogonal transform, and with blocking artifacts reduced.

- *Fast DCT and DST graph filtering* (Chapter 5): We study the discrete cosine transform (DCT) and discrete sine transform (DST) and their associated filtering operations from a graph-based perspective. For each transform within the 16 types of DCT and DST, we characterize a family of sparse graph operators, where all of these operators have the DCT/DST as their eigenmatrix. The sparse operators can be viewed as graph filters operating in the DCT/DST (graph frequency) domain, but with sample (graph vertex) domain implementation. With the derived sparse graph filters, we propose a filter design approach that allows us to approximate a filter with an arbitrary frequency response by a linear combination of sparse operators that can be applied efficiently in graph vertex domain. Experimental results demonstrate that DCT/DST filters can be approximated with an efficient one when the graph size is sufficiently large. In addition, we apply our method to rate-distortion optimization process for transform type selection in video codec, which gives a negligible compression loss with a significant encoding time saving.

- *Adaptive GFT for perceptual video coding* (Chapter 6): In image and video coding applications, perceptual quality metrics such as Structural Similarity (SSIM) are typically used after encoding, but not tied to the encoding process. We consider an alternative framework where the goal is to optimize a *weighted MSE* metric, where different weights can be assigned to each pixel so as to reflect their relative importance in terms of perceptual image quality. For this purpose, we propose a novel transform coding scheme based on irregularity-aware graph Fourier transform (IAGFT), where the induced IAGFT is orthogonal, but the orthogonality is defined with respect to an inner product corresponding to the weighted MSE. We propose to use weights derived from local variances of the input image, such that the weighted MSE aligns with SSIM. In this way, the associated IAGFT can achieve a coding efficiency improvement in terms of multi-scale SSIM with respect to conventional transform coding based on DCT.

1.1 Notations and Conventions

We use bold symbols to denote vectors and matrices. Lowercase symbols with double sub-indices are used to denote matrix elements. For example, $m_{i,j}$ denotes the (i, j) entry

of the matrix $\mathbf{M}$. The $n \times n$ identity matrix is denoted by $\mathbf{I}_n$. The $n \times n$ order-reversal permutation matrix is denoted by

$$\mathbf{J}_n = \begin{pmatrix} & & & 1 \\ & & 1 & \\ & \iddots & & \\ 1 & & & \end{pmatrix}. \tag{1.1}$$

When $\mathbf{J}_n$ right (resp. left) multiplies another matrix, it flips this matrix left to right (resp. up to down). It also satisfies $\mathbf{J}_n = \mathbf{J}_n^\top$ and $\mathbf{J}_n\mathbf{J}_n = \mathbf{I}$. In (1.1) and in what follows, the entries not included in the matrix are meant to be zero. The subscripts of $\mathbf{I}$ and $\mathbf{J}$ matrices indicate their sizes, and may be omitted for brevity. For scalars $a_1, \ldots, a_k$ and square matrices $\mathbf{M}_i$ with arbitrary sizes, we denote diagonal and block diagonal matrices in compact notation as

$$\operatorname{diag}(a_1, \ldots, a_k) = \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_k \end{pmatrix}$$

$$\operatorname{diag}(\mathbf{M}_1, \ldots, \mathbf{M}_k) = \begin{pmatrix} \mathbf{M}_1 & & & \\ & \mathbf{M}_2 & & \\ & & \ddots & \\ & & & \mathbf{M}_k \end{pmatrix}.$$

Finally, the set of n-dimensional real-valued vectors is denoted as $\mathbb{R}^n$, and the set of $n \times n$ orthogonal matrices (with columns normalized to have unit norms) is denoted as $\mathbb{O}^n$. Finally, the pseudo inverse matrix of $\mathbf{M}$ is denoted as $\mathbf{M}^\dagger$.

1.2 Review of Graph Signal Processing

In this thesis, all graphs we consider are assumed to be *undirected* and *weighted*. Let $\mathbf{x}$ be a length-n graph signal associated to a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathbf{W})$. There are n nodes in the vertex set $\mathcal{V}$, each corresponding to one of the entries of $\mathbf{x}$. Each edge $e_{i,j} \in \mathcal{E}$ describes the inter-sample relation between nodes i and j. The (i, j) entry of the weight matrix, $w_{i,j}$, is the weight of the edge between nodes i and j. The *combinatorial graph Laplacian* (CGL) matrix of $\mathcal{G}$ is:

$$\mathbf{L}_{\text{CGL}} = \mathbf{D} - \mathbf{W}, \tag{1.2}$$

where the degree matrix $\mathbf{D} = \text{diag}(d_1, \ldots, d_n)$ is a diagonal matrix with

$$d_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{i,j}. \tag{1.3}$$

The *symmetric normalized Laplacian* (GGL) is defined as $\boldsymbol{\mathcal{L}} = \mathbf{D}^{-1/2}\mathbf{L}_{\text{CGL}}\mathbf{D}^{-1/2}$.

In many cases, we consider graph with self-loops, i.e., edges going from a node to itself. We allow these self-loops to be weighted, with $s_i := w_{ii}$ being the weight of the self-loop on node i. This leads to the definition of *generalized graph Laplacian*:

$$\mathbf{L} = \mathbf{D} - \mathbf{W} + \mathbf{S}, \tag{1.4}$$

where $\mathbf{S} = \text{diag}(s_1, \ldots, s_n)$ is the diagonal self-loop matrix. In this thesis, if not stated otherwise, the terms "Laplacian" or "unnormalized Laplacian" refer to GGL, where the graph is allowed to have self-loops.

A graph is *simple* if it does not contain any self-loops, and k-regular if all node degrees are k. The Laplacian matrix for such graph is called a *combinatorial graph Laplacian*. We call a graph *bipartite* if its vertices can be divided into two disjoint sets (or, two *parts*) $\mathcal{S}_1$ and $\mathcal{S}_2$ such that every edge connects a vertex in $\mathcal{S}_1$ and one in $\mathcal{S}_2$. An *acyclic* graph is a graph with no cycles. If an acyclic graph is connected, it is called a *tree*. Otherwise, it is called a *forest*.

1.2.1 Graph Fourier Transform

There are several definitions of the GFT [36, 90, 104, 114], depending on whether the graph is directed, which fundamental graph operator is used (e.g., adjacency or Laplacian matrix), and how the graph signal energy is defined. Here, we adopt the definition in [114], where the GFT is obtained from the eigen-decomposition of the graph Laplacian matrix, $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$. Then, the i-th coefficient of GFT of a graph signal $\mathbf{x}$ is defined as the projection of $\mathbf{x}$ onto $\mathbf{u}_i$, the i-th column of $\mathbf{U}$. In some applications, such as spectral clustering [132], it may be beneficial to use a GFT defined on the eigenvectors of the symmetric normalized Laplacian $\boldsymbol{\mathcal{L}}$. However, in what follows, we use GFTs associated to the GGL unless stated otherwise.

GFT coefficients provide a frequency representation of the given signal, since GFT basis functions associated to lower (resp. higher) eigenvalues represent lower (resp. higher) variation on the graph. To see this, we note that the Laplacian quadratic form

$$\mathbf{f}^\top \mathbf{L}\mathbf{f} = \sum_{(i,j)\in\mathcal{E}} w_{i,j}(f_i - f_j)^2 + \sum_{k=1}^{n} s_k f_k^2 \tag{1.5}$$

measures the variation of signal $\mathbf{f}$ on the graph. Since $w_{i,j}$ and s_k are non-negative, $\mathbf{L}$ is positive semi-definite and thus $\mathbf{f}^\top \mathbf{L}\mathbf{f}$ is always non-negative. We know that the eigenvectors of $\mathbf{L}$ are the solution to

$$\begin{aligned}
\mathbf{u}_1 &= \operatorname*{argmin}_{\|\mathbf{f}\|=1} \quad \mathbf{f}^\top \mathbf{L}\mathbf{f}, \\
\mathbf{u}_k &= \operatorname*{argmin}_{\mathbf{f}\perp\mathbf{u}_1,\ldots,\mathbf{u}_{k-1},\|\mathbf{f}\|=1} \quad \mathbf{f}^\top \mathbf{L}\mathbf{f}.
\end{aligned}$$

Thus, eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$ form an orthogonal basis with functions having lower to higher variations on the graph. The quantities of their corresponding variations are given by the associated eigenvalues $\lambda_1, \ldots, \lambda_n$, which are also called *graph frequencies*. In addition, from the graph variation quantity (1.5), we can see the connection between GFT basis functions and the graph weights. Note that, when $w_{i,j}$, the weight of edge (i, j), is large, the first GFT basis function $\mathbf{u}_1$ tends to have similar values on entries i and j. Similarly, when there is a self-loop with a large weight on node k, then $\mathbf{u}_1$ tends to have its k-th entry close to zero.

The GFT has a wide range of applications. First, based on the GFT and its frequency interpretation, graph spectral filters [51] can be defined by multiplying the GFT coefficients by the frequency response in the graph spectral domain, leading to applications such as image denoising and edge-preserving smoothing [7, 89]. Second, when the graph signal is modeled by a Gaussian Markov random field (GMRF) [101], the corresponding GFT can be regarded as the orthogonal transform that optimally decorrelates the graph signal and hence reduces spatial redundancies. Based on this fact, the GFT has been applied to image and video compression [26, 32, 50]. Third, in machine learning, when relations between data points are modeled by a graph, the GFT can be used for data clustering [132] and dimensionality reduction for classification [85, 121].

1.2.2 Graph Filtering

For a given graph with Laplacian $\mathbf{L}$, we consider a 1-hop graph operator $\mathbf{Z}$, which could be adjacency or Laplacian matrix. When it is applied to a signal $\mathbf{x}$ to compute $\mathbf{y} = \mathbf{Z}\mathbf{x}$, it defines an operation for each node with its 1-hop neighbors (e.g., when $\mathbf{Z} = \mathbf{A}$,

$y(i) = \sum_{j \in \mathcal{N}(i)} x(j)$ is a sum over the neighbors of graph node i). Furthermore, it can be shown that a $\mathbf{y} = \mathbf{Z}^K \mathbf{x}$ is a K-hop operation, and thus a degree-K polynomial of $\mathbf{Z}$,

$$\mathbf{H} = \sum_{k=0}^{K} g_k \mathbf{Z}^k, \quad \text{with } \mathbf{Z}^0 = \mathbf{I}, \tag{1.6}$$

induces an operation for each node with its neighbors within K hops. The operation defined in (1.6) is thus called a *graph filter*, an FIR graph filter, or a polynomial graph filter (PGF). In what follows, we refer to $\mathbf{Z}$ as *graph operator* for short[1]. For the rest of this thesis, we choose $\mathbf{Z} = \mathbf{L}$ or define $\mathbf{Z}$ as a matrix with the same eigenbasis as $\mathbf{L}$, e.g., $\mathbf{Z}$ could be a polynomial of $\mathbf{L}$ such as $\mathbf{Z} = 2\mathbf{I} - \mathbf{L}$.

Let $\mathbf{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N)$ be the matrix of eigenvectors of $\mathbf{Z} = \mathbf{L}$, and λ_j the associated eigenvalue of $\boldsymbol{\phi}_j$, then it follows that $\mathbf{\Phi}$ is also the eigenvector matrix of the polynomial $\mathbf{H}$. That is,

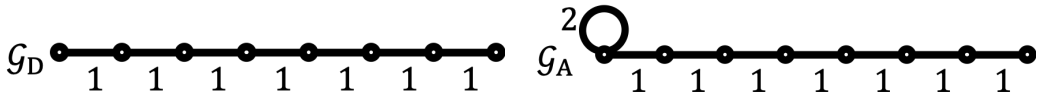
$$\mathbf{H} = \mathbf{\Phi} \cdot h(\mathbf{\Lambda}) \cdot \mathbf{\Phi}^\top, \quad h(\mathbf{\Lambda}) := \mathrm{diag}(h(\lambda_1), \cdots, h(\lambda_N)). \tag{1.7}$$

The eigenvalue $h(\lambda_j)$ of $\mathbf{H}$ associated to $\boldsymbol{\phi}_j$ is called the *frequency response* of λ_j. Note that with $\mathbf{y} = \mathbf{H}\mathbf{x}$, in the GFT domain we have $\hat{\mathbf{y}} = h(\mathbf{\Lambda})\hat{\mathbf{x}}$, meaning that the filter operator amplifies or reduces the signal component with λ_j frequency by $h(\lambda_j)$ in the GFT domain. We also note that the definition via (1.7) generalizes the notion of digital filter, since when $\mathbf{\Phi}$ is the discrete Fourier transform (DFT) matrix, $\mathbf{H}$ reduces to the classical Fourier filter [105]. Given a desired graph frequency response $\mathbf{h} = (h_1, \ldots, h_N)^\top$, its associated polynomial coefficients in (1.6) by solving a least squares minimization problem [105]:

$$\mathbf{g} = \underset{\mathbf{g}}{\mathrm{argmin}} \quad ||\mathbf{h} - \mathbf{\Psi}\mathbf{g}||^2, \quad \text{where } \mathbf{\Psi} = \begin{pmatrix} 1 & \lambda_1 & \ldots & \lambda_1^K \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \lambda_N & \ldots & \lambda_N^K \end{pmatrix}, \tag{1.8}$$

with λ_j being the j-th eigenvalue of $\mathbf{Z}$. The PGF operation $\mathbf{y} = \mathbf{H}\mathbf{x}$ can be done via an efficient iterative algorithm: 1) $\mathbf{t}^{(0)} = g_K \mathbf{x}$, 2) $\mathbf{t}^{(i)} = \mathbf{Z}\mathbf{t}^{(i-1)} + g_{K-i}\mathbf{x}$, and 3) $\mathbf{y} = \mathbf{t}^{(K)}$. Its complexity is given by $\mathcal{O}(KE)$ with E the number of graph edges. Note that this algorithm does not require GFT computation, and its complexity depends on how sparse $\mathbf{Z}$ is (lower complexity for sparser $\mathbf{Z}$).

[1] In the literature, $\mathbf{Z}$ is often called *graph shift operator* [34, 51, 104, 109]. Here, we simply call it graph operator, since its properties are different from shift in conventional signal processing, which is always reversible, while the graph operator $\mathbf{Z}$, in most cases, is not.



(a) (b)

Figure 1.1: Graphs associated to (a) DCT-II, (b) DST-IV (ADST).

1.2.3 Gaussian Markov Random Fields (GMRFs)

A Gaussian Markov random field (GMRF) [101] $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a Gaussian random vector, whose inverse covariance matrix (also called a *precision matrix*) $\mathbf{L} = \boldsymbol{\Sigma}^{\dagger}$ is positive semidefinite. Each GMRF $\mathbf{x}$ is associated with an undirected graph $\mathcal{G}$, where the relations among its entries can be characterized by $\mathbf{L}$: the graph vertices correspond to elements in $\mathbf{x}$, and edges of $\mathcal{G}$ describe the *partial correlations* among elements of $\mathbf{x}$:

$$Corr(x_i, x_j | \mathbf{x}_{-ij}) = -\frac{l_{ij}}{\sqrt{l_{ii} l_{jj}}}, \quad i \neq j,$$

which is the correlation between elements i and j given all the other elements. Note that for a general GMRF, the associated graph $\mathcal{G}$ may have negative edge weights.

When the precision matrix $\mathbf{L}$ of a GMRF $\mathbf{x}$ is a graph Laplacian matrix, i.e., its off-diagonal entries of are non-positive, then $\mathbf{x}$ is called an *attractive GMRF*. In this case, the graph $\mathcal{G}$ associated to $\mathbf{x}$ has non-negative edge weights. Among attractive GMRFs, we say $\mathbf{x}$ is a *diagonally dominant GMRF* if $\mathbf{L}$ is diagonally dominant (i.e., $|l_{ii}| > \sum j \neq i |l_{ij}|$); we call $\mathbf{x}$ an *intrinsic GMRF* if $\mathbf{L}$ is singular (i.e., $\mathbf{L}$ is a combinatorial Laplacian) [25].

In image and video compression, pixel data can be modeled by attractive GMRFs, whose probabilistic properties lead to justification of the use of DCT and ADST [99,117]. We summarize the connection between DCT/ADST and graph Laplacian matrices below.

1.2.4 DCT and DST

The discrete cosine transform (DCT) and discrete sine transform (DST) are *discrete trigonometric transforms (DTT)*. These are orthogonal transforms that operate on a finite sequence, where the basis functions are cosine functions and sine functions, respectively. Depending on how samples are taken from a continuous cosine and sine functions, eight types of DCT and eight types of DST can be defined [136]. The definitions are shown in Table 1.2, where scaling factors c_j and d_j are included to make the transforms orthogonal [99, Table A.1]. Table 1.1 shows left and right boundary conditions (b.c.) of all DTTs, which arise from even and odd symmetries of the cosine and sine functions,

		Right b.c.			
		$\phi_j(N+k)$ $= \phi_j(N-k)$	$\phi_j(N+k)$ $= -\phi_j(N-k)$	$\phi_j(N+k)$ $= \phi_j(N-k+1)$	$\phi_j(N+k)$ $= -\phi_j(N-k+1)$
Left b.c.	$\phi_j(k)$ $= \phi_j(-k+2)$	DCT-I	DCT-III	DCT-V	DCT-VII
	$\phi_j(k)$ $= -\phi_j(-k)$	DST-III	DST-I	DST-VII	DST-V
	$\phi_j(k)$ $= \phi_j(-k+1)$	DCT-VI	DCT-VIII	DCT-II	DCT-IV
	$\phi_j(k)$ $= -\phi_j(-k+1)$	DST-VIII	DST-VI	DST-IV	DST-II

Table 1.1: Left and right boundary conditions (b.c.) of 16 DCTs and DSTs.

and can be trivially verified based on DTT definitions in Table 1.2. We will refer to the 8 DCT types DCT-I to DCT-VIII, and to those DST types DST-I to DST-VIII.

Discrete Cosine Transform (DCT)

In what follows, the term DCT refers to DCT-II, the most widely used type, unless stated otherwise. For $j = 1, \ldots, N$ and $k = 1, \ldots, N$, we write the k-th element of the j-th DCT-II basis function with length N as

$$u_j(k) = \sqrt{\frac{2}{N}} c_j \cos \frac{(j-1)(k-\frac{1}{2})\pi}{N}, \tag{1.9}$$

with normalization constant c_j being $1/\sqrt{2}$ for $j = 1$ and 1 otherwise. If those basis functions are written in vector form $\mathbf{u}_j \in \mathbb{R}^N$, it was pointed out in [117] that $\mathbf{u}_j$ are eigenvectors of the $N \times N$ Laplacian matrix

$$\mathbf{L}_{\text{DCT-II}} = \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}. \tag{1.10}$$

This means that the DCT is the GFT of a graph $\mathcal{G}_{\text{D}}$ associated to Laplacian matrix $\mathbf{L}_{\text{DCT-II}}$. In fact, $\mathcal{G}_{\text{D}}$ is a line graph with uniform weights, where the case with $N = 8$ is shown in Fig. 1.1(a). The eigenvalue corresponding to $\mathbf{u}_j$ is $\omega_j = 2 - 2\cos((j-1)\pi/N)$.

Table 1.2: Definitions of all types of DCTs and DSTs. The indices j and k range from 1 to N. Scaling factors for rows and columns are given by $c_j = 1/\sqrt{2}$ for $j = 1$ and 1 otherwise, and $d_j = 1/\sqrt{2}$ for $j = N$ and 1 otherwise.

DTT	Transform functions
DCT-I	$\phi_j(k) = \sqrt{\frac{2}{N-1}} c_j c_k d_j d_k \cos \frac{(j-1)(k-1)\pi}{N-1}$
DCT-II	$\phi_j(k) = u_j(k) = \sqrt{\frac{2}{N}} c_j \cos \frac{(j-1)(k-1/2)\pi}{N}$
DCT-III	$\phi_j(k) = \sqrt{\frac{2}{N}} c_k \cos \frac{(j-1/2)(k-1)\pi}{N}$
DCT-IV	$\phi_j(k) = \sqrt{\frac{2}{N}} \cos \frac{(j-1/2)(k-1/2)\pi}{N}$
DCT-V	$\phi_j(k) = \frac{2}{\sqrt{2N-1}} c_j c_k \cos \frac{(j-1)(k-1)\pi}{N-1/2}$
DCT-VI	$\phi_j(k) = \frac{2}{\sqrt{2N-1}} c_j d_k \cos \frac{(j-1)(k-1/2)\pi}{N-1/2}$
DCT-VII	$\phi_j(k) = \frac{2}{\sqrt{2N-1}} d_j c_k \cos \frac{(j-1/2)(k-1)\pi}{N-1/2}$
DCT-VIII	$\phi_j(k) = \frac{2}{\sqrt{2N+1}} \cos \frac{(j-1/2)(k-1/2)\pi}{N+1/2}$
DST-I	$\phi_j(k) = \sqrt{\frac{2}{N+1}} \sin \frac{jk\pi}{N+1}$
DST-II	$\phi_j(k) = \sqrt{\frac{2}{N}} d_j \sin \frac{j(k-1/2)\pi}{N}$
DST-III	$\phi_j(k) = \sqrt{\frac{2}{N}} d_k \sin \frac{(j-1/2)k\pi}{N}$
DST-IV	$\phi_j(k) = v_j(k) = \sqrt{\frac{2}{N}} \sin \frac{(j-1/2)(k-1/2)\pi}{N}$
DST-V	$\phi_j(k) = \frac{2}{\sqrt{2N+1}} \sin \frac{jk\pi}{N+1/2}$
DST-VI	$\phi_j(k) = \frac{2}{\sqrt{2N+1}} \sin \frac{j(k-1/2)\pi}{N+1/2}$
DST-VII	$\phi_j(k) = \frac{2}{\sqrt{2N+1}} \sin \frac{(j-1/2)k\pi}{N+1/2}$
DST-VIII	$\phi_j(k) = \frac{2}{\sqrt{2N-1}} d_j d_k \sin \frac{(j-1/2)(k-1/2)\pi}{N-1/2}$

Discrete sine transform (DST)

One of the most notable applications of DST is image and video coding. In particular, it has been shown that DST-VII optimally decorrelates 1D intra residual blocks following a Gaussian Markov model [45, 49]. In [46], it was pointed out that a fast algorithm is

available for DST-IV, a variation of DST-VII that can achieve a comparable compression gain. While DST-VII has been adopted by the HEVC standard [120], DST-IV is used in AV1 and AV2 standards [10], developed by the Alliance for Open Media (AOM). In this thesis, the term ADST refers to DST-IV as in AV1.

We denote $\mathbf{v}_j$ a DST-IV basis function, which has the form

$$v_j(k) = \sqrt{\frac{2}{N}} \sin \frac{(j-\frac{1}{2})(k-\frac{1}{2})\pi}{N}. \tag{1.11}$$

The vectors $\mathbf{v}_j$ are eigenvectors of

$$\mathbf{L}_{\text{DST-IV}} = \begin{pmatrix} 3 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}, \tag{1.12}$$

with corresponding eigenvalues $\delta_j = 2 - 2\cos((j-1/2)\pi/N)$. In addition, DST-IV is the GFT of a graph $\mathcal{G}_{\text{A}}$. The graph $\mathcal{G}_{\text{A}}$ corresponding to $N = 8$ is shown in Fig. 1.1(b).

1.3 A Roadmap

In the following chapters, we will study various graph-based approaches, such as transformations and filtering operations based those definitions introduced above. In Chapter 2 we explore conditions of graphs for fast GFT algorithms. Based on the results of Chapter 2, we consider efficient graph-based transform (including GFT and lapped graph Fourier transform (LGFT)) with applications in image and video coding in Chapter 3. Then, we investigate efficient DTT filtering methods using graph filter design approaches in Chapter 5. Finally, we study a variation of GFT, irregularity aware graph Fourier transform (IAGFT), in Chapter 6 to enhance image perceptual quality.

Chapter 2

Fast GFT based on graph symmetry and bipartition

The discrete Fourier transform (DFT) has a well-known divide and conquer algorithm: fast Fourier transform (FFT) [15]. The availability of FFT algorithm is very important for many signal processing problems, particularly in speech and biomedical signal processing. However, the GFT, as a generalization of the DFT, does not have fast algorithms in general. Note that the DFT basis functions are always even or odd symmetric, which can be exploited for obtaining fast algorithms. However, arbitrary graph topologies do not always lead to Laplacian eigenvectors with the same symmetry properties. Lack of fast algorithms is a significant drawback for GSP approaches, particularly when the GFT needs to be applied repeatedly. This has led researchers to investigate techniques for fast GFT computation. Magoarou et. al. have proposed a series of approaches for fast GFT approximation [60–62]. The work in [61] uses a gradient-descent-based optimization approach to approximate the GFT matrix by a product of sparse matrices, while [60] refines this method such that the resulting transform matrix can approximately diagonalize the graph Laplacian. In [62], a truncated Jacobi algorithm was introduced for picking the Givens rotations used in the approximate fast GFT, leading to an implementation with the structure shown in Figure 2.1. This approach was further analyzed in [63], which demonstrates that more Givens rotations are required to approximate the Laplacian eigenvectors whose corresponding eigenvalues are close.

Although these methods [60–62] are able to find approximate fast GFTs, they do so without taking advantage of structural properties of the original graph. The relation of topological properties of graphs, such as bipartition, repeated subgraphs, symmetry, and uniformity of weights, to the structure of the GFT bases is an important topic in GSP. In this chapter, we investigate graph topological properties that lead to fast implementations of the GFT, then derive the resulting fast GFT algorithms. In particular, we will show that for graphs with certain symmetry or bipartition properties, a *exact and fast* GFTs based on *Haar units* (as will be defined in Section 2.1) can be designed. We

Work in this chapter has been published in [72] with MATLAB and C code implementation available in [69].

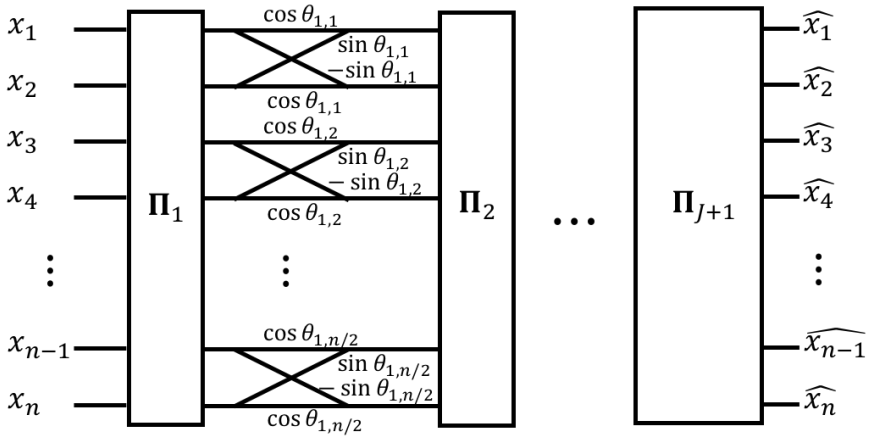
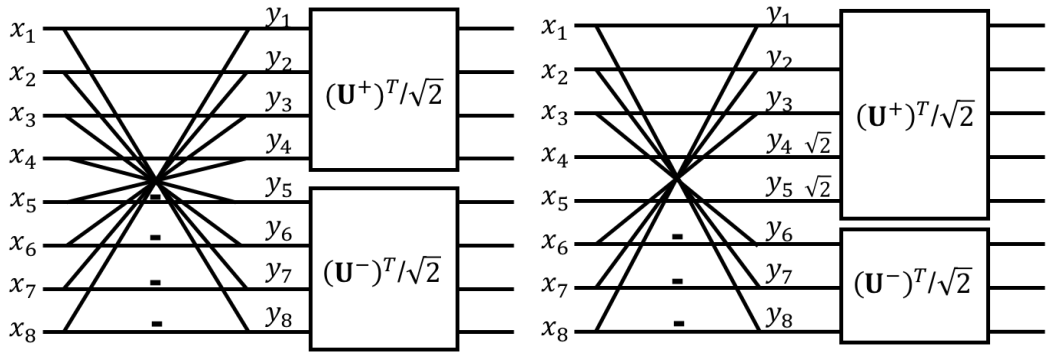


Figure 2.1: Fast transform using J layers of Givens rotations. The parameter $0 < \theta_{i,j} \leq \pi$ is the j-th rotation angle in the i-th butterfly stage, and $\mathbf{\Pi}_k$ are permutation operations.

propose divide-and-conquer fast GFT algorithms for symmetric graphs and demonstrate that the resulting fast GFTs lead to significant complexity reduction, leading to potential benefits in hardware implementation or in scenarios where the graph is fixed and the corresponding GFT is applied multiple times. We show that graphs for which such Haar-unit-based fast GFTs can be developed are useful in applications such as video coding and human action analysis. Unlike fast approximate GFTs [60–62], our fast GFTs are based on graph topological properties, and are exact. Experimental results show that as long as the desired graph symmetry property is available, our fast GFTs can provide a better speed improvement than [62].

The rest of this chapter is organized as follows. In Section 2.1 we introduce the so-called Haar unit and derive the conditions for a stage of Haar units to be available in a GFT implementation. In Section 2.2 we define the notion of symmetry for graphs that gives rise to a Haar unit stage, and propose a graph decomposition method for obtaining fast GFT algorithms. Some examples of fast GFTs are shown in Section 2.3. We present experimental results for the complexity of the derived fast GFTs in Section 2.4. Finally, Section 2.5 concludes this chapter.



(a) (b)

Figure 2.2: Two examples of fast algorithms using butterfly stages with $n = 8$.

2.1 Haar Units in GFTs

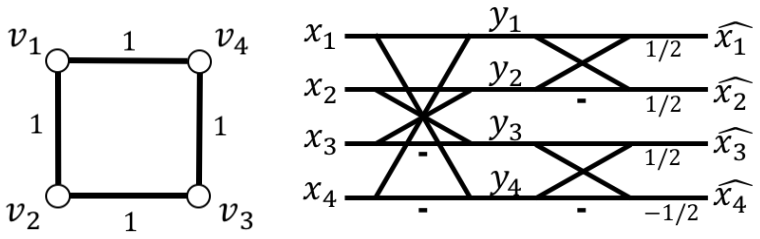
An n dimensional *Givens rotation* [39], commonly referred to as a *butterfly* [46, 62, 64], is a linear transformation that applies a rotation of angle θ to two coordinates, denoted as p and q. Its associated matrix $\mathbf{\Theta}(p, q, \theta)$ has the form:

$$\begin{cases} \Theta_{pp} = \Theta_{qq} = \cos\theta, & \\ \Theta_{qp} = -\Theta_{pq} = \sin\theta, & \\ \Theta_{ii} = 1, & i \neq p,\ q, \\ \Theta_{ij} = 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

A system using layers of parallel Givens rotations, such as the one in Figure 2.1, can be used to design an approximate fast transform. In particular, each Givens rotation can be implemented using three lifting steps [17], which further reduces the number of operations involved.

In this work, we aim to achieve computation efficiency by using Givens rotations with angle $\theta = \pi/4$, which typically require only an addition and a subtraction (the factor $1/\sqrt{2}$ can be absorbed into other stages of the transform computation). This is also equivalent to a 2×2 Haar transform, so we refer to this operator as *Haar unit*, as opposed to general Givens rotations. We say that a *butterfly stage* is a stage in a transform diagram with several parallel Givens rotations or Haar units. For example, in Figure 2.2(a), we call the stage that produces y_i from x_i a butterfly stage, and the operator that produces y_1 and y_8 from x_1 and x_8 a Haar unit of this butterfly stage.

We consider a divide and conquer framework based on stages of Haar units and parallel sub-transforms, as illustrated Figure 2.2. For each Haar unit, we always assume



(a) (b)

Figure 2.3: (a) The 4-node cycle graph and (b) a fast algorithm for its GFT.

that the two output variables, such as y_1 and y_8 in Figure 2.2(a), will be inputs of different sub-transforms in the next stage. Otherwise, this Haar unit, such that the one acting on x_1 and x_8, can be trivially absorbed into the next stage.

As a first example, we consider a 4-node cycle graph with unity edge weights as in Figure 2.3(a). It has a GFT matrix

$$\mathbf{U}_{\mathcal{C}_4} = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \end{pmatrix}.$$

Note that the second and third columns of $\mathbf{U}_{\mathcal{C}_4}$ both correspond to eigenvalue 2, which has multiplicity 2. This means that the GFT basis is not unique, because we can obtain another orthogonal basis for the eigenspace corresponding to eigenvalue 2. An example of another basis for this GFT is the length-4 DFT, which has a well-known fast algorithm [15]. Despite this non-uniqueness, a fast algorithm for a particular GFT basis would still be useful: we can first apply it to obtain coefficients for this particular basis, then apply an m-dimensional rotation ($m \times m$ orthogonal transform) on those m GFT coefficients associated to eigenvalues with a multiplicity $m > 1$, to obtain coefficients associated to another GFT basis. For example, if we apply a proper rotation to $\widehat{x_2}$ and $\widehat{x_3}$ in Figure 2.3(b), we can obtain the second and third DFT coefficients. In what follows, we study GFT implementations for which stages of Haar units are available. In cases where eigenvalues of high multiplicity are present, we favor the set of eigenvectors for the corresponding subspace that will lead to a more efficient implementation.

Based on the structure of $\mathbf{U}_{\mathcal{C}_4}$, it can be seen that the GFT can be implemented using two butterfly stages, as in Figure 2.3(b). We refer to those Haar units located at the first stage (such as the one acting on x_1 and x_4) as *left Haar units*, and those located at the last stage (such as the one producing $\widehat{x_1}$ and $\widehat{x_2}$) as *right Haar units*. We will explore the

conditions that allow a GFT to be factored into terms that include left and right Haar units, which will enable us to develop techniques for designing such fast GFTs. We will show that *right Haar units are associated to bipartite graphs (Section 2.1.1), while left Haar units are related to graph symmetries (Section 2.1.2).*

For a general graph with n nodes, we define the following $n \times n$ orthogonal matrix to represent a stage of p parallel Haar units (with $p \leq n/2$):

$$\mathbf{B}_{n,p} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{I}_p & \mathbf{0} & \mathbf{J}_p \\ \mathbf{0} & \sqrt{2}\mathbf{I}_{n-2p} & \mathbf{0} \\ \mathbf{J}_p & \mathbf{0} & -\mathbf{I}_p \end{pmatrix}. \tag{2.2}$$

Note that $\mathbf{B}_{n,p}^\top = \mathbf{B}_{n,p}$, and when we multiply a vector $\mathbf{x} = (x_1, \ldots, x_n)^\top$ by $\mathbf{B}_{n,p}$, we have

$$(\mathbf{B}_{n,p} \cdot \mathbf{x})_i = \begin{cases} \frac{1}{\sqrt{2}}(x_i + x_{n+1-i}), & i = 1, \ldots, p \\ x_i, & i = p+1, \ldots, n-p \\ \frac{1}{\sqrt{2}}(-x_i + x_{n+1-i}), & i = n-p+1, \ldots, n \end{cases}$$

For example, $\mathbf{B}_{8,4}$ are $\mathbf{B}_{8,3}$ are equivalent to the butterfly stages in Figures 2.2(a) and (b), respectively, with a scaling constant $1/\sqrt{2}$. The factors $\sqrt{2}$ and $1/\sqrt{2}$ are included in (2.2) so that the columns of $\mathbf{B}_{n,p}$ have unit norms; in this way, when an orthogonal matrix $\mathbf{U}$ satisfies $\mathbf{U} = \mathbf{B}_{n,p}\bar{\mathbf{U}}$, then $\bar{\mathbf{U}}$ is orthogonal as well, meaning that $\mathbf{U}$ can be factorized into a butterfly stage and another orthogonal transform.

2.1.1 Right Haar Units

Let an orthogonal transform $\mathbf{U}$ have a right butterfly stage with p Haar units, and assume without loss of generality that the entries of input and output vectors are properly ordered. Then, in compact notation, the GFT of input $\mathbf{x}$ can be written as

$$\mathbf{U}^\top \mathbf{x} = \mathbf{B}_{n,p} \cdot \mathrm{diag}(\mathbf{E}^\top, \mathbf{F}^\top) \cdot \mathbf{x}, \tag{2.3}$$

where $\mathbf{E} \in \mathbb{O}^{n-p}$ and $\mathbf{F} \in \mathbb{O}^p$. This means that

$$\mathbf{U} = \begin{pmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} & \mathbf{0} \\ \mathbf{E}_{21} & \mathbf{E}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F} \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{I}_p & \mathbf{0} & \mathbf{J}_p \\ \mathbf{0} & \sqrt{2}\mathbf{I}_{n-2p} & \mathbf{0} \\ \mathbf{J}_p & \mathbf{0} & -\mathbf{I}_p \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{E}_{11} & \sqrt{2}\mathbf{E}_{12} & \mathbf{E}_{11}\mathbf{J}_p \\ \mathbf{E}_{21} & \sqrt{2}\mathbf{E}_{22} & \mathbf{E}_{21}\mathbf{J}_p \\ \mathbf{F}\mathbf{J}_p & \mathbf{0} & -\mathbf{F} \end{pmatrix}, \tag{2.4}$$

where $\mathbf{E}_{11}$, $\mathbf{E}_{12}$, $\mathbf{E}_{21}$, and $\mathbf{E}_{22}$ are subblock components of $\mathbf{E}$. Recall that $\mathbf{J}$ flips a matrix left to right when right-multiplied. Thus, for $k = 1, \ldots, p$, if we denote the k-th column

Table 2.1: Definitions of symmetries for vectors, matrices, and graphs.

Subject	Terminology	Definition
Vector $\mathbf{v}$	Even symmetric	$\mathbf{v} = \mathbf{J}\mathbf{v}$
	Odd symmetric	$\mathbf{v} = -\mathbf{J}\mathbf{v}$
Matrix $\mathbf{M}$	Symmetric	$\mathbf{M} = \mathbf{M}^\top$
	Centrosymmetric [5]	$\mathbf{M} = \mathbf{J}\mathbf{M}^\top\mathbf{J}$
	Bisymmetric [5]	$\mathbf{M} = \mathbf{M}^\top = \mathbf{J}\mathbf{M}^\top\mathbf{J}$
Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathbf{W})$	ϕ-symmetric (Definition 2)	$w_{i,j} = w_{\phi(i),\phi(j)},\ \forall i, j$

of $\mathbf{U}$ as $\mathbf{u}_k = (\mathbf{e}_k^\top, \mathbf{f}_{n-p-k+1}^\top)^\top$, then the $(n-k+1)$-th column of $\mathbf{U}$ is $(\mathbf{e}_k^\top, -\mathbf{f}_{n-p-k+1}^\top)^\top$. GFT matrices with this structure arise for k-regular bipartite graphs (k-RBGs):

Lemma 1 ([52]). *Let $\mathbf{L}$ be the Laplacian of a k-RBG with $\mathcal{S}_1 = \{1, \ldots, p\}$ and $\mathcal{S}_2 = \{p+1, \ldots, n\}$. If $\mathbf{u} = (\mathbf{u}_1^\top,\ \mathbf{u}_2^\top)^\top$ with $\mathbf{u}_1 \in \mathbb{R}^p$ and $\mathbf{u}_2 \in \mathbb{R}^{n-p}$ is an eigenvector of $\mathbf{L}$ with eigenvalue λ, then $\hat{\mathbf{u}} = (\mathbf{u}_1^\top,\ -\mathbf{u}_2^\top)^\top$ is an eigenvector of $\mathbf{L}$ with eigenvalue $2k - \lambda$.*

If we consider eigenvectors of the *symmetric normalized Laplacian*, the same symmetry properties hold for any bipartite graph:

Lemma 2 ([14]). *Let $\boldsymbol{\mathcal{L}}$ be the normalized Laplacian of a bipartite graph with $\mathcal{S}_1 = \{1, \ldots, p\}$ and $\mathcal{S}_2 = \{p+1, \ldots, n\}$. If $\mathbf{u} = (\mathbf{u}_1^\top,\ \mathbf{u}_2^\top)^\top$ with $\mathbf{u}_1 \in \mathbb{R}^p$ and $\mathbf{u}_2 \in \mathbb{R}^{n-p}$ is an eigenvector of $\boldsymbol{\mathcal{L}}$ with eigenvalue λ, then $\hat{\mathbf{u}} = (\mathbf{u}_1^\top,\ -\mathbf{u}_2^\top)^\top$ is an eigenvector of $\boldsymbol{\mathcal{L}}$ with eigenvalue $2 - \lambda$.*

From Lemmas 1 and 2, we know that as long as the graph satisfies certain conditions (bipartite for $\boldsymbol{\mathcal{L}}$ or k-RBG for $\mathbf{L}$), then there exists a GFT matrix with a right butterfly stage as in (2.3), even though the GFT matrix may not be unique. The matrices $\mathbf{E}$ and $\mathbf{F}$ can be obtained by multiplying $\mathbf{B}_{n,p}$ in the right of a pre-computed $\mathbf{U}$:

$$\operatorname{diag}(\mathbf{E}, \mathbf{F}) = \mathbf{U} \cdot \mathbf{B}_{n,p}.$$

2.1.2 Left Haar Units

If n is even and the GFT has a butterfly stage in the left with exactly $n/2$ Haar units as in Figure 2.2(a), then

$$\mathbf{U} = \mathbf{B}_{n,n/2} \begin{pmatrix} \mathbf{U}^+ & \mathbf{0} \\ \mathbf{0} & \mathbf{U}^- \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{U}^+ & \mathbf{J}\mathbf{U}^- \\ \mathbf{J}\mathbf{U}^+ & -\mathbf{U}^- \end{pmatrix},$$

where $\mathbf{U}^+, \mathbf{U}^- \in \mathbb{O}^{n/2}$. From the right hand side we see that each column $\mathbf{u}_i$ of $\mathbf{U}$ must be either *even symmetric* (i.e., $\mathbf{u}_i = \mathbf{J}\mathbf{u}_i$) or *odd symmetric* (i.e., $\mathbf{u}_i = -\mathbf{J}\mathbf{u}_i$). In this case, the Laplacian must be *centrosymmetric* (symmetric around the center):

Lemma 3 ([5]). *Let n be even. An $n \times n$ matrix $\mathbf{Q}$ has a set of n linearly independent eigenvectors that are even or odd symmetric if and only if $\mathbf{Q}$ is centrosymmetric, i.e., $\mathbf{Q} = \mathbf{J}\mathbf{Q}^\top\mathbf{J}$.*

Note that the Laplacian matrix $\mathbf{L}$ of an undirected graph is symmetric ($\mathbf{L} = \mathbf{L}^\top$) by nature, but an additional centrosymmetry condition ($\mathbf{L} = \mathbf{J}\mathbf{L}^\top\mathbf{J}$) is required so that Lemma 3 holds. Such a matrix with both symmetries ($\mathbf{L} = \mathbf{L}^\top = \mathbf{J}\mathbf{L}^\top\mathbf{J}$) is called *bisymmetric*, and its entries are symmetric around both diagonals. The various types of symmetries considered in this chapter listed in Table 2.1.

Lemma 3 states that for even n, a GFT can be factored to include $n/2$ left Haar units if and only if the associated Laplacian matrix is bisymmetric (with nodes properly ordered). We now generalize this result to the case when there are only $p < n/2$ Haar units in the first butterfly stage, and with a possibly odd n. Again, we assume without loss of generality that the graph nodes, input, and output variables are properly ordered (notations defined for general node ordering will be introduced in Section 2.2). We let $\mathcal{V}_X = \{1, \ldots, p\}$, $\mathcal{V}_Z = \{p+1, \ldots, n-p\}$, and $\mathcal{V}_Y = \{n-p+1, \ldots, n\}$ be disjoint subsets of vertices. We define

$$\mathbf{G} := \mathbf{B}_{n,p}^\top \cdot \mathbf{L} \cdot \mathbf{B}_{n,p}, \tag{2.5}$$

and denote the corresponding subblock components of $\mathbf{L}$ and $\mathbf{G}$ as

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{XX} & \mathbf{L}_{XZ} & \mathbf{L}_{XY} \\ \mathbf{L}_{ZX} & \mathbf{L}_{ZZ} & \mathbf{L}_{ZY} \\ \mathbf{L}_{YX} & \mathbf{L}_{YZ} & \mathbf{L}_{YY} \end{pmatrix}, \ \mathbf{G} = \begin{pmatrix} \mathbf{G}_{XX} & \mathbf{G}_{XZ} & \mathbf{G}_{XY} \\ \mathbf{G}_{ZX} & \mathbf{G}_{ZZ} & \mathbf{G}_{ZY} \\ \mathbf{G}_{YX} & \mathbf{G}_{YZ} & \mathbf{G}_{YY} \end{pmatrix}. \tag{2.6}$$

Similar to (2.3) and (2.4), the GFT matrix with a first butterfly stage of p Haar units has the form

$$\mathbf{U} = \mathbf{B}_{n,p} \cdot \mathrm{diag}(\mathbf{U}^+, \mathbf{U}^-), \quad \mathbf{U}^+ \in \mathbb{O}^{n-p}, \ \mathbf{U}^- \in \mathbb{O}^p. \tag{2.7}$$

Then the following lemma describes the conditions for $\mathbf{L}$ to have a GFT with p left Haar units.

Lemma 4. *Let $\mathbf{L}$ be a graph Laplacian matrix, then there exists a GFT matrix $\mathbf{U}$ in the form of (2.7), i.e., the associated $\mathbf{G}$ in (2.5) is block-diagonal, if and only if*

$$\mathbf{L}_{YY} = \mathbf{J}\mathbf{L}_{XX}\mathbf{J}, \quad \mathbf{L}_{YX} = \mathbf{J}\mathbf{L}_{XY}\mathbf{J}, \quad \mathbf{L}_{ZY} = \mathbf{L}_{ZX}\mathbf{J}. \tag{2.8}$$

Note that when Z is empty, i.e., $p = n/2$, (2.8) implies that $\mathbf{L}$ has to be centrosymmetric, as in Lemma 3.

Proof: If $\mathbf{U}$ is a GFT matrix satisfying (2.7), we denote the subblocks of $\mathbf{U}^+$ as $\mathbf{U}^+_{XX}$, $\mathbf{U}^+_{XZ}$, $\mathbf{U}^+_{ZX}$, and $\mathbf{U}^+_{ZZ}$, and rewrite (2.7) as

$$\mathbf{U} = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{I}_p & \mathbf{0} & \mathbf{J}_p \\ \mathbf{0} & \sqrt{2}\mathbf{I}_{n-2p} & \mathbf{0} \\ \mathbf{J}_p & \mathbf{0} & -\mathbf{I}_p \end{pmatrix} \begin{pmatrix} \mathbf{U}^+_{XX} & \mathbf{U}^+_{XZ} & \mathbf{0} \\ \mathbf{U}^+_{ZX} & \mathbf{U}^+_{ZZ} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{U}^- \end{pmatrix} \tag{2.9}$$

Denote the matrix of eigenvalues of $\mathbf{L}$ as $\mathbf{\Lambda} = \mathrm{diag}(\mathbf{\Lambda}_X, \mathbf{\Lambda}_Z, \mathbf{\Lambda}_Y)$ with subblock sizes p, $n-2p$, and p, respectively. Then, we can express each subblock of $\mathbf{L}$ by expanding $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ with (2.9), and we can trivially verify that (2.8) holds.

To show the converse, we assume that (2.8) holds. Expanding the right hand side of (2.5), we can express the subblocks of $\mathbf{G}$ in terms of those of $\mathbf{L}$. In particular,

$$\mathbf{G}_{XY} = \frac{1}{2}\left(\mathbf{L}_{XX}\mathbf{J} + \mathbf{J}\mathbf{L}_{YX}\mathbf{J} - \mathbf{L}_{XY} - \mathbf{J}\mathbf{L}_{YY}\right), \quad \mathbf{G}_{ZY} = \frac{\sqrt{2}}{2}\left(\mathbf{L}_{ZX}\mathbf{J} - \mathbf{L}_{ZY}\right).$$

With these expressions, (2.8) implies that $\mathbf{G}_{XY}$, $\mathbf{G}_{ZY}$, and their transpose versions $\mathbf{G}_{YX}$, $\mathbf{G}_{YZ}$ are all zero. This means that $\mathbf{G}$ is block-diagonal, and thus has an eigendecomposition as

$$\mathbf{G} = \mathrm{diag}(\mathbf{V}_1, \mathbf{V}_2) \cdot \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \cdot \mathrm{diag}(\mathbf{V}_1, \mathbf{V}_2)^\top,$$

where $\mathbf{V}_1 \in \mathbb{O}^{n-p}$ and $\mathbf{V}_2 \in \mathbb{O}^p$. It follows that $\mathbf{B}_{n,p} \cdot \mathrm{diag}(\mathbf{V}_1, \mathbf{V}_2)$ is an eigenmatrix of $\mathbf{L} = \mathbf{B}_{n,p} \cdot \mathbf{G} \cdot \mathbf{B}^\top_{n,p}$ as in (2.7). □

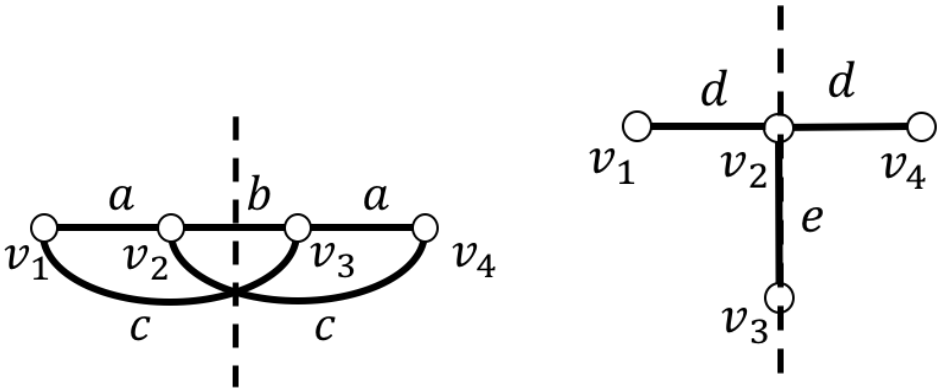
Under the conditions of (2.8), $\mathbf{G}$ reduces to

$$\mathbf{G} = \begin{pmatrix} \mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J} & \sqrt{2}\mathbf{L}_{XZ} & \mathbf{0} \\ \sqrt{2}\mathbf{L}^\top_{XZ} & \mathbf{L}_{ZZ} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY} \end{pmatrix}, \tag{2.10}$$

and $\mathbf{U}^+$ and $\mathbf{U}^-$ are respectively the eigenmatrices of

$$\mathbf{L}^+ := \begin{pmatrix} \mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J} & \sqrt{2}\mathbf{L}_{XZ} \\ \sqrt{2}\mathbf{L}^\top_{XZ} & \mathbf{L}_{ZZ} \end{pmatrix}, \quad \mathbf{L}^- := \mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY}. \tag{2.11}$$

A diagram with $n = 8$, $p = 3$ is shown in Figure 2.2(b) as an example.



(a) $\mathcal{G}_1$ (b) $\mathcal{G}_2$

Figure 2.4: Example of symmetric graphs. (a) A graph with a bisymmetric Laplacian matrix. (b) A graph with a Laplacian satisfying (2.8).

Note that the desired properties in $\mathbf{L}$ correspond to certain symmetry properties *in the graph topology*. If $\mathcal{V}_Z$ is empty, Lemma 4 implies that

$$w_{i,j} = w_{n+1-i,n+1-j}, \quad \forall i \in \mathcal{V},\ j \in \mathcal{V}. \tag{2.12}$$

In this case, when we plot the nodes in order on a 1D line, we can identify an axis in the middle, around which all edges and self-loops are symmetric. An example of a graph whose Laplacian is bisymmetric is shown in Figure 2.4(a).

More generally, if $\mathcal{V}_Z$ is nonempty, then the first two equations in (2.8) indicate that the sub-matrix of $\mathbf{L}$ associated to $\mathcal{V}_X$ and $\mathcal{V}_Y$ is bisymmetric. This means that $\mathcal{V}_X$ and $\mathcal{V}_Y$ contain vertices that are symmetric to each other. The third equation in (2.8) implies that when there is an edge connecting $k \in \mathcal{V}_Z$ and $i \in \mathcal{V}_X$, there must be an edge with the same weight connecting k and $n+1-i \in \mathcal{V}_Y$ as well. An example of this type of graph is shown in Figure 2.4(b), where $\mathcal{V}_X = \{1\}$, $\mathcal{V}_Z = \{2,3\}$, and $\mathcal{V}_Y = \{4\}$. Similar to Figure 2.4(a), we can identify a symmetry around the middle, though nodes in $\mathcal{V}_Z$ are not paired with symmetric counterparts.

Based on the observations above, we see that left Haar units are available when the graph has symmetry properties related to Lemma 4. However, Lemma 4 assumes that the nodes had been ordered properly, so that the Laplacian has the required bisymmetric structure. In general, the node labels of a graph will not be such that this condition is automatically met, even if the graph is symmetric. For example, if a different node labeling is applied to the graph of Figure 2.4(a), its corresponding Laplacian may not

be bisymmetric anymore. In the next section we study methods to identify graph symmetries directly using node pairing functions, which will allow us to design fast GFT algorithms, regardless of how the nodes are initially labeled.

2.2 Fast GFTs Based on Graph Symmetry

In this section, we will characterize how Lemma 4 relates to the graph topology. In particular, we define the symmetry properties observed in Figure 2.4 based on an involution (node-pairing function) in Section 2.2.1. Given an observed graph symmetry characterized by an involution, in Section 2.2.2 we define the node sets $\mathcal{V}_X$, $\mathcal{V}_Y$, and $\mathcal{V}_Z$. In Section 2.2.3, we propose a graph decomposition approach for searching fast GFTs in stages.

2.2.1 Graph Symmetry Based on Node Pairing

The symmetries of Figure 2.4 can be described in terms of complete (Figure 2.4(a)) and incomplete (Figure 2.4(b)) *node pairings*. Such pairings can be defined by bijective mappings that are their own inverses, namely, *involutions*:

Definition 1 (Involution [96])**.** *A permutation on a finite set $\mathcal{V}$ is called an involution if it is its own inverse, i.e., $\phi(\phi(i)) = i$ for all $i \in \mathcal{V}$.*

We will use them to identify graph symmetries.

Definition 2 (ϕ-symmetric graph)**.** *Let ϕ be an involution on the vertex set $\mathcal{V}$ of a graph $\mathcal{G}$, then $\mathcal{G}$ is ϕ-symmetric if $w_{i,j} = w_{\phi(i),\phi(j)}$ for all $i \in \mathcal{V}$, $j \in \mathcal{V}$.*

Note that in Definition 2 the required property has to hold also for $i = j$. That is, $s_i = w_{i,i} = w_{\phi(i),\phi(i)} = s_{\phi(i)}$, meaning that the self-loops on nodes i and $\phi(i)$ are required to have the same weight. Also note that, among permutations, only involutions are valid to define the symmetry described in Definition 2, since the pairing functions that lead to the conditions in (2.8) can only be induced by involutions.[1]

Let $\mathcal{V} = \{1, \ldots, n\}$ be the vertex set, and let us denote an involution ϕ as $\phi = (\phi(1), \phi(2), \ldots, \phi(n))$. For example, the involutions corresponding to the symmetries of

[1]Note that graph symmetry can be defined differently in different contexts. In algebraic graph theory, graph symmetry is defined based on transitivity of vertices and edges [38]. In [28], a graph is called symmetric if there exists a non-identical permutation ϕ (not necessarily an involution) on the graph nodes that leaves the graph unaltered. These definitions are beyond the scope of this work. When we refer to graph symmetry, we always assume an involution ϕ is specified such that Definition 2 holds.

graphs in Figures 2.4(a) and (b) are $\phi_a = (4,3,2,1)$ and $\phi_b = (4,2,3,1)$, respectively. We also denote the number of available Haar units for a given ϕ as

$$p_\phi := \frac{1}{2} \times |\{i \in \mathcal{V} : \ i \neq \phi(i)\}| \,. \tag{2.13}$$

2.2.2 Node Partitioning for Haar Units

Once we observe a graph symmetry and characterize it by an involution ϕ, we can identify the nodes on the axis of symmetry, $\mathcal{V}_Z := \{i \in \mathcal{V} : \ \phi(i) = i\}$, then partition the other nodes into two sets $\mathcal{V}_X$ and $\mathcal{V}_Y$ such that nodes in those sets belong to different sides of the symmetry axis. In this way, we can define an orthogonal matrix $\mathbf{B}_\phi$ as a permuted version of $\mathbf{B}_{n,p_\phi}$ based on ϕ, $\mathcal{V}_X$, $\mathcal{V}_Y$, and $\mathcal{V}_Z$ in the following way:

$$(\mathbf{B}_\phi)_{i,j} = \begin{cases} 1/\sqrt{2}, & i = j \in \mathcal{V}_X \\ -1/\sqrt{2}, & i = j \in \mathcal{V}_Y \\ 1, & i = j \in \mathcal{V}_Z \\ 1/\sqrt{2}, & i \in \mathcal{V}_X, \ j = \phi(i) \in \mathcal{V}_Y \\ 1/\sqrt{2}, & i \in \mathcal{V}_Y, \ j = \phi(i) \in \mathcal{V}_X \\ 0, & \text{otherwise} \end{cases} \tag{2.14}$$

This means that $\mathbf{L}_\phi := \mathbf{B}_\phi^\top \mathbf{L} \mathbf{B}_\phi$ is a permuted version of (2.10), whose block diagonal structure gives the following theorem.

Theorem 1 (Block-diagonalization of Laplacian based on graph symmetry)**.** *Let the graph $\mathcal{G}$ with Laplacian $\mathbf{L}$ be ϕ-symmetric. Then, $(\mathbf{L}_\phi)_{i,j} = (\mathbf{L}_\phi)_{j,i} = 0$ if $i \in \mathcal{V}_X \cup \mathcal{V}_Z$ and $j \in \mathcal{V}_Y$.*

While Theorem 1 is derived based on the unnormalized Laplacian $\mathbf{L}$, it holds for normalized Laplacian as well.

2.2.3 Main Approach–Decomposition of Symmetric Graphs

The block-diagonalization of (2.5) maps $\mathbf{L}$ to $\mathbf{G}$ via $\mathbf{B}_{n,p}$, where $\mathbf{G}$ in (2.10) can be regarded as the Laplacian of a graph with two connected components, with possibly negative edge weights and respective Laplacians $\mathbf{L}^+$ and $\mathbf{L}^-$ defined in (2.11). Thus, $\mathcal{G}$ with Laplacian $\mathbf{L}$ is decomposed into two separate graphs $\mathcal{G}^+$ and $\mathcal{G}^-$ with Laplacians $\mathbf{L}^+$ and $\mathbf{L}^-$, vertex sets $\mathcal{V}^+ := \mathcal{V}_X \cup \mathcal{V}_Z$ and $\mathcal{V}^- := \mathcal{V}_Y$, weight matrices $\mathbf{W}^+$ and $\mathbf{W}^-$, respectively. In this way, the GFT of $\mathbf{L}$ can be implemented using Haar units in $\mathbf{B}_{n,p}$, followed by two sub-GFTs characterized by Laplacians $\mathbf{L}^+$ and $\mathbf{L}^-$. Explicitly considering the graphs resulting from this decomposition is useful because the symmetry properties

in $\mathbf{L}^+$ and $\mathbf{L}^-$ can be further explored to determine whether additional reductions in complexity are possible. Moreover, considering the transforms after the Haar units as GFTs can potentially help us provide better interpretations of the overall GFT.

By definition of graph Laplacian, we can express the self-loop weights s_i and edge weights $w_{i,j}$ in terms of entries of the Laplacian matrix $\mathbf{L} = (l_{i,j})_{i,j}$, and vice versa:

$$s_i = \sum_{j \in \mathcal{V}} l_{i,j}, \quad w_{i,j} = -l_{i,j} \text{ for } i \neq j, \tag{2.15}$$

$$l_{ii} = s_i + \sum_{\substack{j \in \mathcal{V} \\ j \neq i}} w_{i,j}, \quad l_{i,j} = -w_{i,j} \text{ for } i \neq j. \tag{2.16}$$

Together with (2.11), we can express the self-loop/edge weights of $\mathcal{G}^+$ and $\mathcal{G}^-$ in terms of those of $\mathcal{G}$, as described in the following theorem.

Theorem 2. *If $\mathcal{G}$ is ϕ-symmetric with node partitions $\mathcal{V}_X$, $\mathcal{V}_Y$, and $\mathcal{V}_Z$, then the weights of $\mathcal{G}^+$ (with vertex set $\mathcal{V}^+ = \mathcal{V}_X \cup \mathcal{V}_Z$) and $\mathcal{G}^-$ (with vertex set $\mathcal{V}^- = \mathcal{V}_Y$) are given by*

$$w^+_{i,j} = \begin{cases} w_{i,j} + w_{i,\phi(j)}, & \text{if } i \in \mathcal{V}_X,\ j \in \mathcal{V}_X \\ \sqrt{2} w_{i,j}, & \text{if } i \in \mathcal{V}_X,\ j \in \mathcal{V}_Z \text{ or } i \in \mathcal{V}_Z,\ j \in \mathcal{V}_X \\ w_{i,j}, & \text{if } i \in \mathcal{V}_Z,\ j \in \mathcal{V}_Z, \end{cases}$$

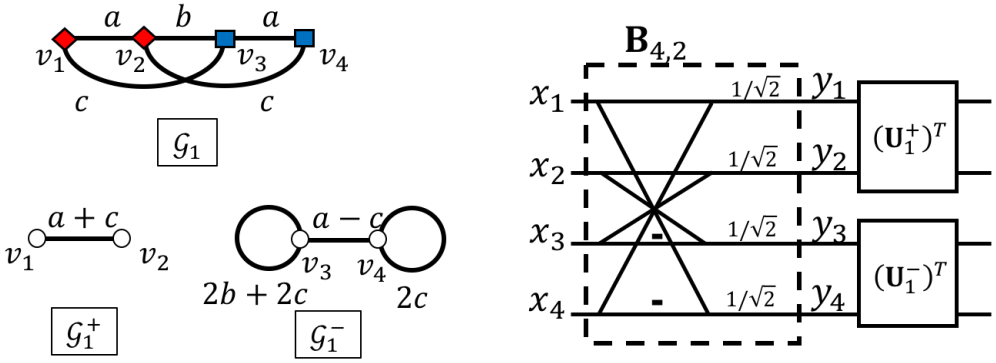
$$s^+_i = \begin{cases} s_i - (\sqrt{2}-1) \sum_{j \in \mathcal{V}_Z} w_{i,j}, & \text{if } i \in \mathcal{V}_X \\ s_i + (2-\sqrt{2}) \sum_{j \in \mathcal{V}_X} w_{i,j}, & \text{if } i \in \mathcal{V}_Z, \end{cases}$$

$$w^-_{i,j} = w_{i,j} - w_{i,\phi(j)}, \quad \forall i, j \in \mathcal{V}_Y, \quad i \neq j$$

$$s^-_i = s_i + 2 \sum_{j \in \mathcal{V}_X} w_{i,j} + \sum_{j \in \mathcal{V}_Z} w_{i,j}, \quad \forall i \in \mathcal{V}_Y.$$

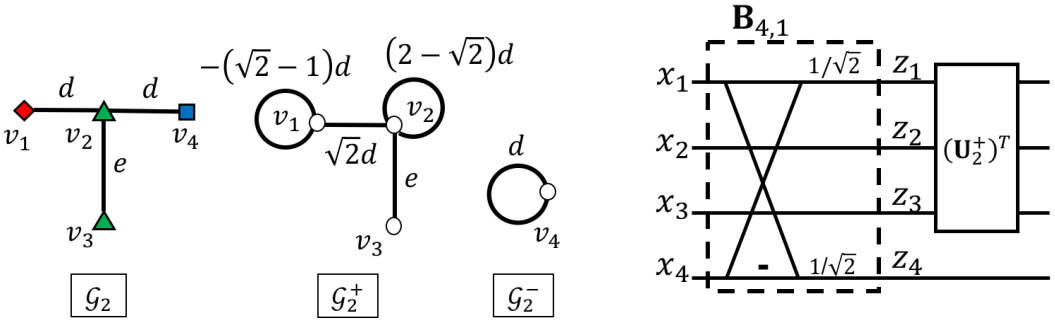
The proof of Theorem 2 refers to Appendix A.1. We use the toy examples of Figures 2.5 and 2.6 (with $\mathcal{V}_Z = \emptyset$ and $\mathcal{V}_Z \neq \emptyset$, respectively) to illustrate the graph decomposition. Note that $\mathcal{G}^+$ and $\mathcal{G}^-$ may have negative weights even if $\mathcal{G}$ does not. For any signal $\mathbf{x}$, we denote the "sum" (low-pass) and "difference" (high-pass) outputs of Haar units as $\mathbf{x}^+$ and $\mathbf{x}^-$: $((\mathbf{x}^+)^\top, (\mathbf{x}^-)^\top)^\top = \mathbf{B}_\phi^\top \mathbf{x}$. For example, in Figure 2.5, $\mathbf{x}^+ = (y_1, y_2)^\top$ and $\mathbf{x}^- = (y_3, y_4)^\top$. In Figure 2.6, $\mathbf{x}^+ = (z_1, z_2, z_3)^\top$ and $\mathbf{x}^- = (z_4)$.

The graph construction of Theorem 2 creates two disconnected sub-graphs by removing all edges between $\mathcal{V}_y$ and $\mathcal{V}_x \cup \mathcal{V}_z$ and preserving all other edges, but changing some of the weights and adding self-loops. Three types of cases lead to one or two edges being removed:



(a) (b)

Figure 2.5: (a) Symmetric graph decomposition for the graph in Figure 2.4(a). Red diamonds and blue squares represent nodes in $\mathcal{V}_X$ and $\mathcal{V}_Y$, respectively. (b) The associated fast GFT diagram for $\mathcal{G}_1$, where $\mathbf{U}_1^+$ and $\mathbf{U}_1^-$ are the GFTs of $\mathcal{G}_1^+$ and $\mathcal{G}_1^-$, respectively.



(a) (b)

Figure 2.6: (a) Symmetric graph decomposition for the graph in Figure 2.4(b). Red diamonds, green triangles, and blue squares represent nodes in $\mathcal{V}_X$, $\mathcal{V}_Z$, and $\mathcal{V}_Y$, respectively. (b) The associated fast GFT diagram for $\mathcal{G}_2$, where $\mathbf{U}_2^+$ is the GFT of $\mathcal{G}_2^+$.

1. **Edges connecting two symmetric nodes $i \in \mathcal{V}_X$ and $\phi(i) \in \mathcal{V}_Y$.** The edge with weight b in Figure 2.5(a) is an example of this case. These edges are removed and lead to self-loops with twice the original weight in $\mathcal{G}^-$ ($2b$ in this case).

2. **Two symmetric edges: each connecting a node in $\mathcal{V}_X$ to a node in $\mathcal{V}_Y$.** The two edges with weight c in Figure 2.5(a) are an example. These edges are removed, but lead to changes in two edge weights, with the weight of the edge in $\mathcal{G}^+$ increasing and that of the edge in $\mathcal{G}^-$ decreasing. Two self-loops are also added to the corresponding nodes in $\mathcal{G}^-$.

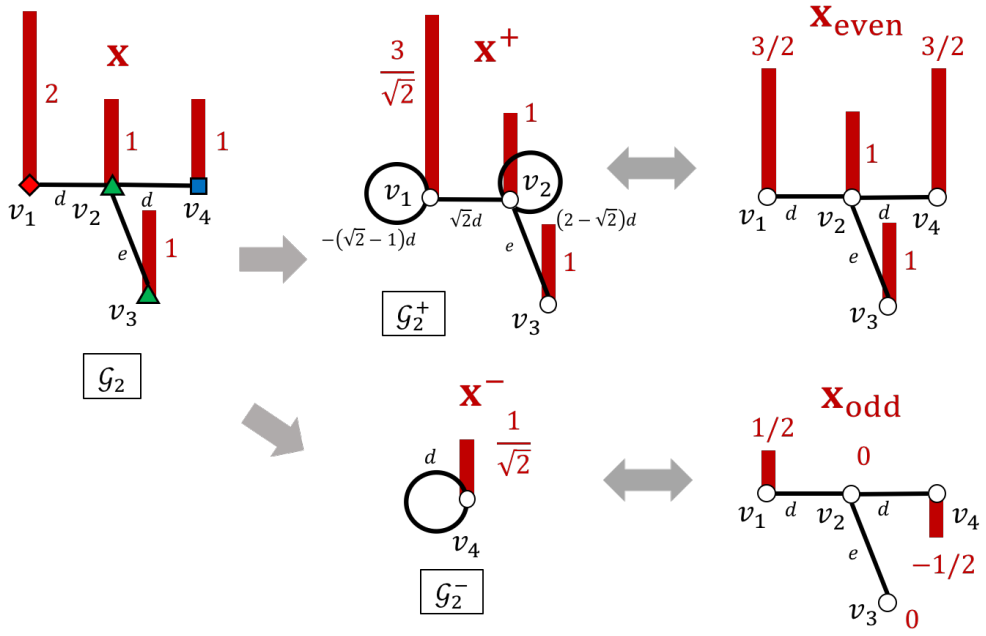


Figure 2.7: An example of even and odd symmetric components on the graph in Figure 2.6. Signals $\mathbf{x}^+$ and $\mathbf{x}^-$ are outputs of the Haar units. Signals $\mathbf{x}_{\text{even}}$ and $\mathbf{x}_{\text{odd}}$ are associated to $\mathbf{x}^+$ and $\mathbf{x}^-$ by (2.17).

3. **Two symmetric edges with a common node in $\mathcal{V}_Z$.** Edges with weight d in Figure 2.6(a) belong to this case. This case results in a single edge being kept, with a modified edge weight and two self-loops in $\mathcal{G}^+$, and a self-loop in $\mathcal{G}^-$.

Note that, the signals $\mathbf{x}^+$ and $\mathbf{x}^-$ correspond to $\mathcal{G}^+$ and $\mathcal{G}^-$, and can be regarded as even and odd symmetric components of the original graph signal $\mathbf{x}$. An example associated to Figure 2.6 is shown in Figure 2.7. We can see that a graph signal can be decomposed into two components $\mathbf{x}_{\text{even}}$ and $\mathbf{x}_{\text{odd}}$, which correspond to $\mathbf{x}^+$ and $\mathbf{x}^-$, respectively, by

$$\mathbf{x}_{\text{even}}(i) = \begin{cases} \mathbf{x}^+(i)/\sqrt{2}, & i \in \mathcal{V}_X \\ \mathbf{x}^+(\phi(i))/\sqrt{2}, & i \in \mathcal{V}_Y \\ \mathbf{x}^+(i), & i \in \mathcal{V}_Z \end{cases}, \quad \mathbf{x}_{\text{odd}}(i) = \begin{cases} \mathbf{x}^-(\phi(i))/\sqrt{2}, & i \in \mathcal{V}_X \\ -\mathbf{x}^-(i)/\sqrt{2}, & i \in \mathcal{V}_Y \\ 0, & i \in \mathcal{V}_Z \end{cases} \tag{2.17}$$

In particular, $\mathbf{x}_{\text{even}}$ and $\mathbf{x}_{\text{odd}}$ have even and odd symmetries based on the node pairing, i.e., $\mathbf{x}_{\text{even}}(i) = \mathbf{x}_{\text{even}}(\phi(i))$ and $\mathbf{x}_{\text{odd}}(i) = -\mathbf{x}_{\text{odd}}(\phi(i))$ for all $i \in \mathcal{V}$. This can be considered as a generalization of even and odd symmetric components decomposition for finite length time series. Components $\mathbf{x}_{\text{even}}$ and $\mathbf{x}_{\text{odd}}$ of the graph signal $\mathbf{x}$ can be regarded as intermediate results of the GFT coefficients.

The decomposition described in Theorem 2 enables us to search for further stages Haar units in the sub-GFTs $\mathbf{U}^+$ and $\mathbf{U}^-$ by inspecting their associated graphs $\mathcal{G}^+$ and $\mathcal{G}^-$. Once a symmetry based on an involution is found in $\mathcal{G}^+$ or $\mathcal{G}^-$, we can apply the decomposition again, and repeat until symmetry property cannot be found anymore. Some examples will be provided in the next section.

2.3 Examples and Applications

In practice, graphs with distinct weights on different edges or graphs learned from data without any topology constraints are not likely to have the desired bipartition and symmetry properties. Searching for an involution that induces symmetry in a given graph may involve a combinatorial problem because the number of involutions grows faster than polynomial with n [96]. However, bipartite and symmetric structures arise in graphs considered in certain fields. Examples of bipartite graphs include tree-structured graphs, whose GFTs are useful for designing wavelet transforms on graph [112]. Involution-based symmetries can be found in graphs with regular or partially regular topologies (e.g., line, cycle, and grid graphs), graphs that are symmetric by construction (e.g., human skeletal graphs), and uniformly weighted graphs. In the following, we show several classes of graphs with these desired properties.

2.3.1 Bipartite Graphs

As stated in Lemma 1, a k-regular bipartite graph has a GFT with a right butterfly stage. Let the sizes of the parts in the bipartite graph be $|\mathcal{S}_1| = p$, $|\mathcal{S}_2| = n - p$, then sub-GFTs $\mathbf{E}$ and $\mathbf{F}$ have sizes p and $n - p$, and the total number of multiplication operations will be $p^2 + (n - p)^2$. By Lemma 2, the same result can be derived for a GFT derived from the symmetric normalized Laplacian of any bipartite graph.

2.3.2 Graphs with 2-Sparse Eigenvectors

Conditions for 2-sparse graph eigenvectors to exist have been studied in [124, 125]:

Lemma 5 ([124,125])**.** *A Laplacian has an eigenvector* $\mathbf{u}$ *with only two nonzero elements* $\mathbf{u}(i) = 1/\sqrt{2}$, $\mathbf{u}(j) = -1/\sqrt{2}$ *if and only if*

$$\forall v \in \mathcal{V} \backslash \{i, j\}, \quad w_{v,i} = w_{v,j}. \tag{2.18}$$

In fact, the condition (2.18) is equivalent to having $\mathcal{G}$ $\phi_{i,j}$-symmetric, with $\phi_{i,j}(i) = j$, $\phi_{i,j}(j) = i$, and $\phi_{i,j}(k) = k$ for $k \neq i, j$. In this case, each of $\mathcal{V}_X = \{i\}$ and $\mathcal{V}_Y = \{j =$

$\phi(i)\}$ has only one node, and $\mathbf{U}^-$ reduces to a one by one identity matrix. Examples provided in [124] that satisfies Lemma 5 include uniformly weighted graphs with several types of topology: 1) star graph, 2) complete graph, 3) complete bipartite graph, and 4) cycle graph. We also note that, a graph with a clique (a complete subgraph), where at least two nodes in the clique are not connected to any other nodes outside the clique, also satisfies Lemma 5.

2.3.3 Symmetric Line Graphs

A Laplacian matrix associated to a line graph can be viewed as a precision matrix (inverse covariance matrix) of a first-order attractive GMRF, which can be used for modeling image and video pixels [66, 143]. One special case is the line graph with uniform weights, whose GFT is the well-known DCT.

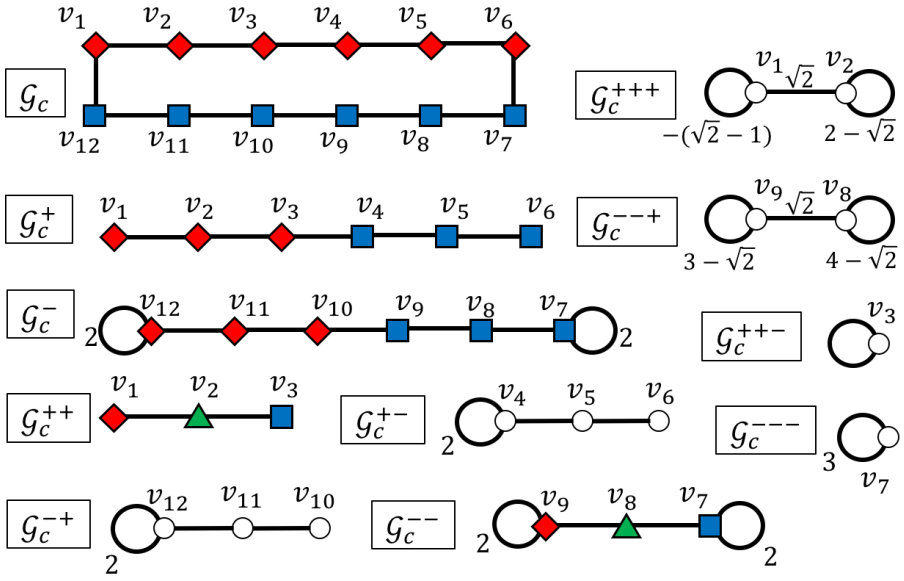
If a line graph $\mathcal{G}_l$ is symmetric around the middle, then it is $\phi = (n, n-1, \ldots, 1)$-symmetric $\mathcal{G}_l$ and its GFT has a left butterfly stage (whether n is even of odd). In Section 3.2, we will study the design of such a fast GFT on symmetric line graphs, and present coding results on inter-predicted residual blocks.

2.3.4 Steerable DFTs

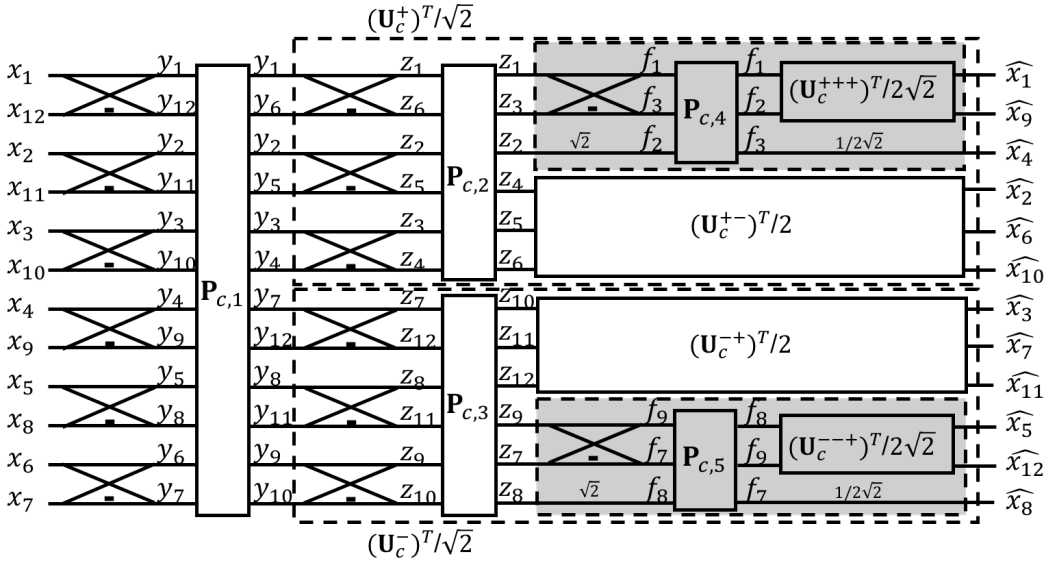
The Laplacian of an n-node cycle graph $\mathcal{G}_c$ with unit weights is a circulant matrix, i.e., each of its row is circularly shifted one element to the right relative to the previous row. Due to this circulant property, the Laplacian can be diagonalized by the DFT matrix, meaning that DFT is one of the GFTs of $\mathcal{G}_c$ (the GFTs of $\mathcal{G}_c$ are not unique since some of the Laplacian eigenvalues have multiplicities greater than one). The family of all GFTs of $\mathcal{G}_c$ is called the set of steerable DFTs (SDFTs) [31].

Fast SDFT algorithms based on graph symmetry

For any n, $\mathcal{G}$ is ϕ-symmetric, with $\phi = (n, n-1, \ldots, 3, 2, 1)$, which enables us to explore fast SDFT algorithms for $\mathcal{G}_c$ other than the FFT algorithm [15]. In fact, the transform shown in Figure 2.3(b) can be obtained using the proposed graph decomposition. Another example with $n = 12$ is shown in Figure 2.8, where two stages of Haar units are available, and some of the sub-GFTs after the first two stages can further be simplified. We also note that, for any SDFT with a length n that is a multiple of 4, the GFTs of $\mathcal{G}_c^{++}$ and $\mathcal{G}_c^{--}$ are DCT-II and DST-IV, respectively, which can also be implemented using fast DCT and ADST algorithms [29, 46, 59].

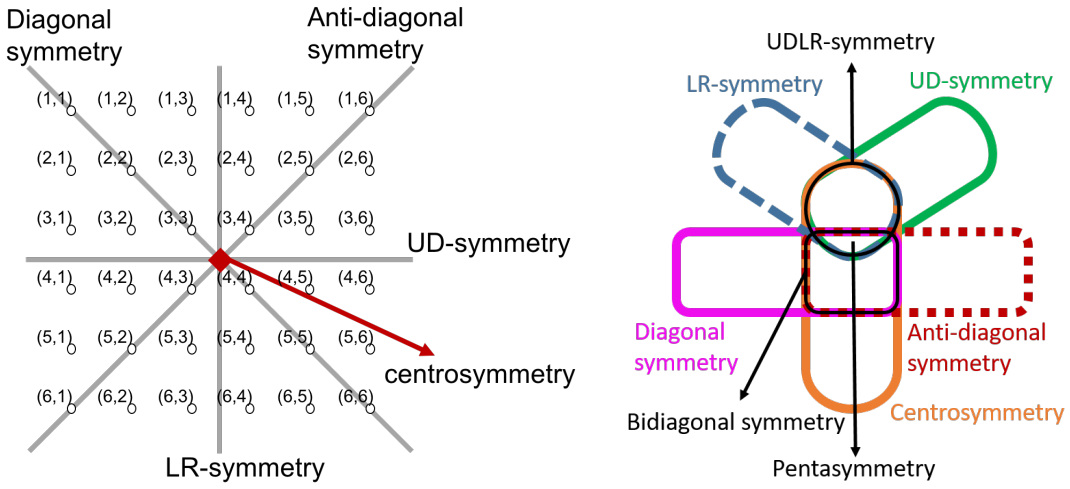


(a)



(b)

Figure 2.8: The 12-node cycle graph: (a) Graph decomposition. (b) The associated fast GFT diagram. Red diamonds, green triangles, and blue squares represent those nodes in $\mathcal{V}_X$, $\mathcal{V}_Z$, and $\mathcal{V}_Y$, respectively, for the *next* stage of decomposition. Unlabeled edges and self-loops have weights 1, and $\mathbf{P}_{c,i}$ are permutation operations. The two shaded sub-GFTs are $(\mathbf{U}_c^{++})^\top/2$ (top) and $(\mathbf{U}_c^{--})^\top/2$ (bottom), respectively.



(a) (b)

Figure 2.9: (a) The axes/point of symmetry for each symmetry type, with a 6×6 grid as an example. Node indices are represented based on the image coordinate system. (b) Relationships among types of symmetries.

Comparison between proposed fast SDFT algorithm and FFT

We also note that, SDFT can be implemented by applying the FFT algorithm, followed by proper rotations on graph frequencies with multiplicities 2 or larger. However, the proposed fast SDFT implementations have several advantages. First, unlike the conventional DFT, the derived GFT will have real operations only. Second, while the FFT algorithm cannot be easily applied when n is a prime number, our method gives at least one left butterfly stage for any n. Introducing this butterfly stage would reduce the number of operations by half (for even n) or nearly half (for odd n). To the best of our knowledge, fast implementations for SDFT other than the FFT algorithm have not been studied in the literature.

2.3.5 Symmetric Grid Graphs

The 2D DCT can be shown to be the GFT of an $N \times N$ uniformly weighted grid[2] and provides an optimal decorrelation of block data modeled by a 2D Gaussian Markov model [142]. Despite the use of DCT, 2D non-separable transforms such as the KLT have been shown to achieve a significantly compression gain over the DCT for some types of video blocks such as intra residual blocks with diagonal prediction direction [2]. Such

[2]The term "grid" refers to a graph with N^2 nodes lying in a regular grid pattern within a square block, where each node can be connected to its 8-neighbors.

Table 2.2: Types of symmetric $N \times N$ grids and their corresponding involutions. Indices k and l represent vertical and horizontal coordinates of grid nodes, as in Figure 2.9(a).

Symmetry type	Involution
Centrosymmetry	$\phi((k,l)) = (N+1-l, N+1-k)$
UD-symmetry	$\phi((k,l)) = (N+1-k, l)$
LR-symmetry	$\phi((k,l)) = (k, N+1-l)$
Diagonal symmetry	$\phi((k,l)) = (l, k)$
Anti-diagonal symmetry	$\phi((k,l)) = (N+1-l, N+1-k)$

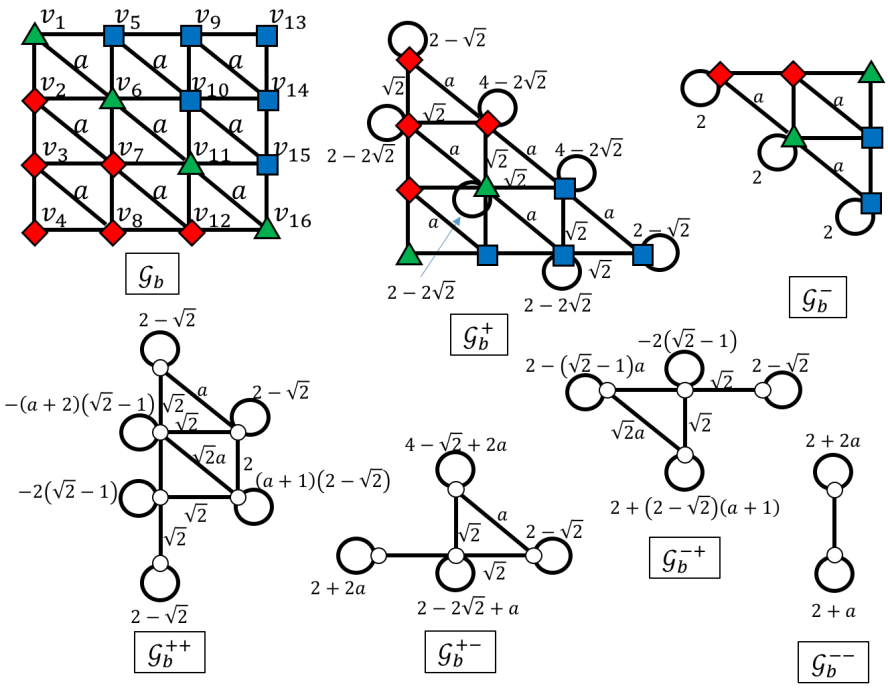
block-based non-separable transforms correspond to GFTs whose graphs are grids with $n = N^2$ nodes. Here, we call such a graph with N^2 nodes a grid, but do not impose any constraint on its graph topology to have more flexibility in designing the transforms.

One key limitation of non-separable transforms is that they typically have much higher computational complexities than separable ones. This issue can be partly addressed when the grid has symmetry properties for fast GFT algorithms. These grid symmetries can be defined based on different axes or point of symmetry, as shown in Figure 2.9(a), and described as follows. We also show in Figure 2.9(b) the relationships among different symmetry types.

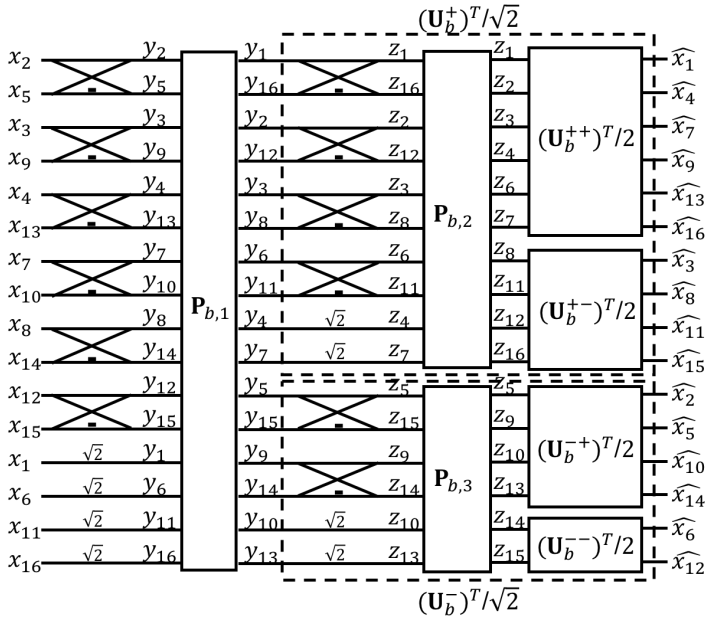
Definition 3 (Grid symmetries)**.** *A N by N grid graph is* UD-symmetric *if it is symmetric around the middle horizontal axis,* LR-symmetric *if it is symmetric around the middle vertical axis. It is called* centrosymmetric *if it is symmetric around the center,* diagonally symmetric *if it is symmetric around the main diagonal, and* anti-diagonally symmetric *if it is symmetric around the anti-diagonal. When a grid is UD- and LR-symmetric, it is called* UDLR-symmetric*. If it is diagonally and anti-diaognally symmetric, we say it is* bidiagonally symmetric*. Finally, if the grid has all symmetry properties above (symmetric around all the four axes and the center), we say it is* pentasymmetric*.*

In fact, fast GFTs for those grids can be derived based on the involutions shown in Table 2.2, which follow directly from grid symmetry conditions. Explicit forms of those fast GFTs can be found in [72]. The GFTs with exact or partial symmetry properties can lead to a gain in energy compaction with respect to the DCT [37].

In Figure 2.10, we show an example with a 4×4 grid $\mathcal{G}_b$ that is bi-diagonally symmetric (symmetric around both diagonals). We can first decompose $\mathcal{G}_b$ based on the diagonal symmetry into $\mathcal{G}_b^+$ and $\mathcal{G}_b^-$. Then, we observe that the symmetry around the anti-diagonal remains in $\mathcal{G}_b^+$ and $\mathcal{G}_b^-$, so further decomposition can be applied. As a result, the overall GFT has two butterfly stages of Haar units, and can be implemented using 4 sub-GFTs with length 6, 4, 4, and 2, as in Figure 2.10.

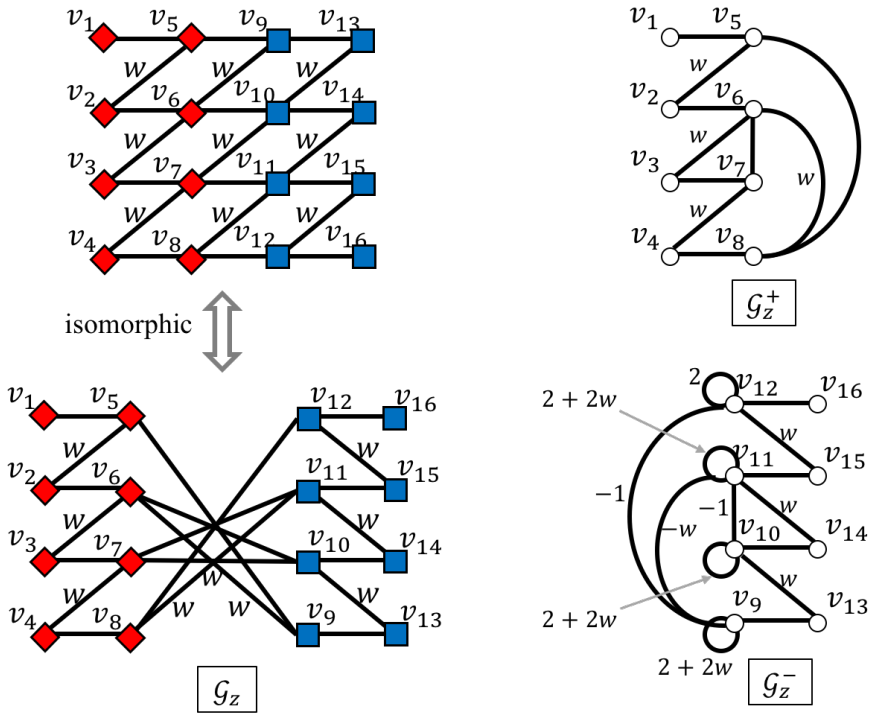


(a)

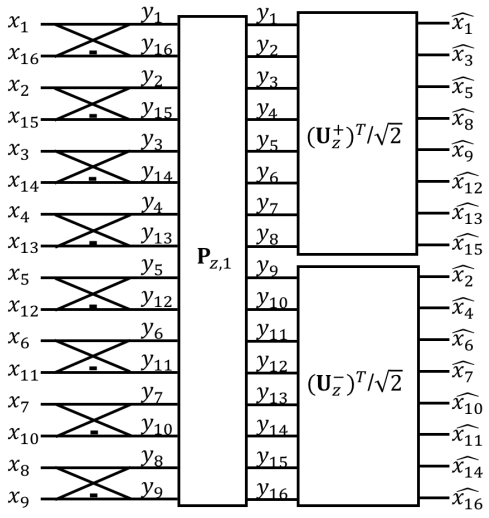


(b)

Figure 2.10: The bi-diagonally symmetric grid. (a) Graph decomposition. (b) The associated fast GFT diagram. Red diamonds, green triangles, and blue squares represent those nodes in $\mathcal{V}_X$, $\mathcal{V}_Z$, and $\mathcal{V}_Y$, respectively, for the *next* stage of decomposition. Unlabeled edges have weights 1, and $\mathbf{P}_{b,i}$ are permutation operations.



(a)



(b)

Figure 2.11: The 4×4 z-shaped grid: (a) Graph decomposition. (b) The associated fast GFT diagram. Red diamonds and blue squares represent those nodes in $\mathcal{V}_X$ and $\mathcal{V}_Y$, respectively, for the *next* stage of decomposition. Unlabeled edges have weights 1, and $\mathbf{P}_{z,1}$ is a permutation operation.

As another example, grid graphs with symmetry properties are also considered in the coding framework proposed in [100], where image blocks are classified into K classes, each with an associated GFT to be applied. These GFTs are derived from graph templates including 1) 4-connected grids with horizontal and vertical edges, 2) 4-connected grids with horizontal and anti-diagonal edges as shown in the top-left of Figure 2.11(a), which we refer to as *z-shaped grid*, and 3) rotated and flipped versions of the z-shaped grid. Those templates are weighted graphs with the constraint that all edges with the same orientation have a common weight. Without loss of generality, all GFTs from graph templates within types 2) or 3) can be characterized by the GFT of a z-shaped grid with horizontal weights 1 and anti-diagonal weights w. We denote this graph as $\mathcal{G}_z$ and derive its fast GFT in Figure 2.11 based on the centrosymmetry of the grid, characterized by the involution $\phi(i) = N + 1 - i$. Note that, if we flip the nodes v_9 to v_{16} up to down, then $\mathcal{G}_z$ becomes a left-right symmetric grid, based on which we can derive $\mathcal{G}_z^+$ and $\mathcal{G}_z^-$ as in Figure 2.11(a). The derived fast GFT diagram in Figure 2.11(b) can thus provide a computational speedup for the coding framework in [100].

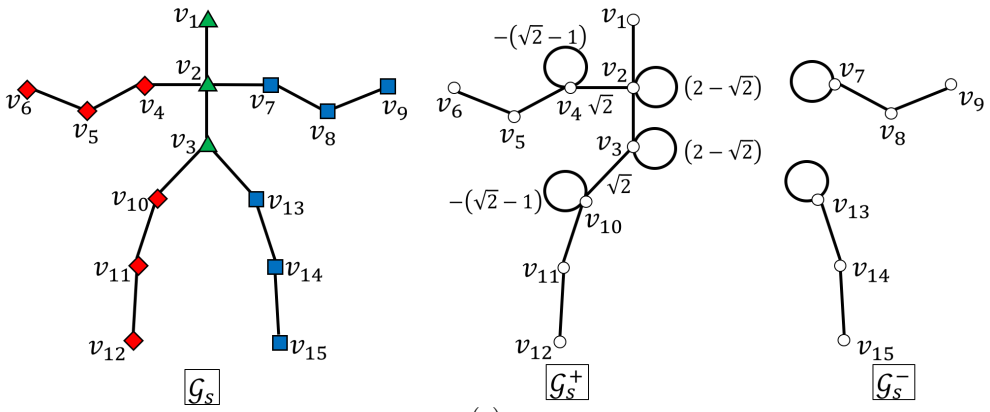
2.3.6 Skeletal Graphs

In human action analysis, the human body can be represented by a hierarchy of joints that are connected with bones. Many motion capture datasets, such as the Florence 3D dataset [110], use 3D coordinates of human joints to represent human actions. We can consider the human skeleton as a graph, and the motion vectors between consecutive frames on each nodes as a graph signals. Recent work has demonstrated that the GFT basis has localization properties corresponding to different human parts [54]. For example, the second GFT basis function has positive entries on joints in the upper body, and negative entries on those in the lower body. Thus, the resulting GFT coefficients can provide a discriminating power between different human actions.

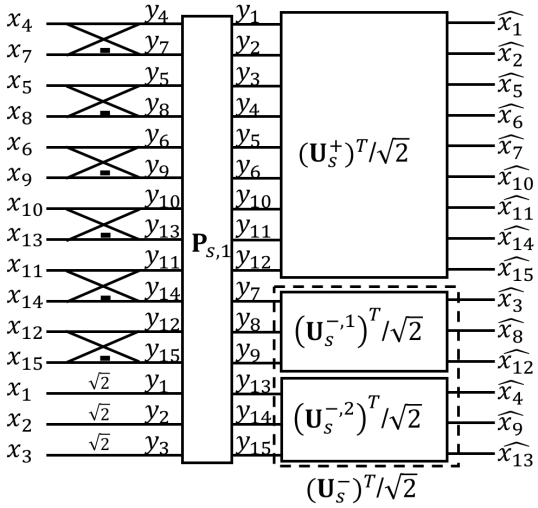
Typical skeletal graphs are symmetric by construction, so a fast GFT can be obtained, as shown in Figure 2.12, where a 15-node skeleton is considered. Note that the butterfly stage is also available for skeletal graphs with non-uniform weights or different topologies, as long as the desired symmetry properties hold.

2.3.7 Search of Symmetries in General Graphs

In the previous examples, symmetry properties of graphs can be easily identified by inspection. However, in general, and particularly for denser graphs, desired symmetry properties may not be straightforward to identify, or may not even exist. To design



(a)



(b)

Figure 2.12: The 15-node skeletal graph: (a) Graph decomposition. (b) The associated fast GFT diagram. Red diamonds, green triangles, and blue squares represent those nodes in $\mathcal{V}_X$, $\mathcal{V}_Z$, and $\mathcal{V}_Y$, respectively, for the *next* stage of decomposition. $\mathbf{U}_s^{-,1}$ and $\mathbf{U}_s^{-,2}$ are the GFTs corresponding to two connected components of $\mathcal{G}_s^-$, respectively. Unlabeled edges have weights 1, and $\mathbf{P}_{s,1}$ is a permutation operation.

fast GFTs for graphs beyond the previous examples, an algorithm for searching a valid involution would thus be useful.

The number of involutions on n elements is [58, Section 5.1.4]

$$T(n) = \sum_{k=0}^{\lfloor n/2 \rfloor} \frac{n!}{2^k (n-2k)! k!} \sim \left(\frac{n}{e}\right)^{n/2} \frac{e^{\sqrt{n}}}{(4e)^{1/4}}, \tag{2.19}$$

which asymptotically grows faster than a polynomial in n. This means that an exhaustive search of valid involutions among $T(n)$ possible candidates is a combinatorial problem. Here, we provide two methods to reduce the complexity of this search. More detailed illustrations and implementations of these methods can be found in Appendix B.

- **Pruning based on the degree list** Note that if $\mathcal{G}$ is ϕ-symmetric, then the degrees of nodes i and $\phi(i)$ must be equal for every i. This necessary condition for ϕ-symmetry allows us to prune the involution search: we can compute the list of degrees first, then prune those involutions φ where for some i the degrees d_i and $d_{\varphi(i)}$ are different. For graphs with many distinct weight values, we tend to have many distinct node degrees, and thus the number of involutions needed to be searched can be significantly reduced.

- **Searching of identical tree branches** Trees (i.e., graphs with no cycles) are connected graphs that have the smallest number of edges. This sparsity property implies that symmetry on trees can be characterized by pairs of identical subtrees (i.e., branches) whose roots are common or adjacent. For example, in Figure 2.12, the two arms in the skeletal graph are identical branches that share a common root, and so are the two legs. Based on an algorithm proposed in [30], we provide in Appendix B an algorithm with $\mathcal{O}(n \log n)$ complexity that, for any given tree $\mathcal{G}$, finds all involutions ϕ such that $\mathcal{G}$ is ϕ-symmetric.

2.4 Experimental Results

In addition to theoretical computational complexity analysis including the number of operations, we also conduct experiments to measure empirical computation complexities of the fast GFTs. We have implemented several fast GFTs in C, in order to simulate an environment closer to hardware.

2.4.1 Comparison with Matrix GFT

In the first experiment, we include the GFTs of several different graph topologies: the cycle graph with unit weights, the bi-diagonally symmetric 6-connected grid (as in Figure 2.10, with $a = 0.5$), the z-shaped grid with $w = 2$, and the skeletal graph. For each graph we implement two fast GFTs with different sizes, and compare the runtime of the matrix GFT implementation and the fast GFT with butterfly stages. We include those GFTs in Figures 2.8 to 2.12, together with larger graphs with the same topology types: the 80-node cycle graph, the 8×8 bi-diagonally symmetric 4-connected grid, the 8×8 z-shaped grid, and the 25-node skeletal graph used in [111]. Detailed design of their fast

Table 2.3: Runtime performance of proposed fast GFTs. The baseline for the runtime reduction rates is the matrix GFT implementation.

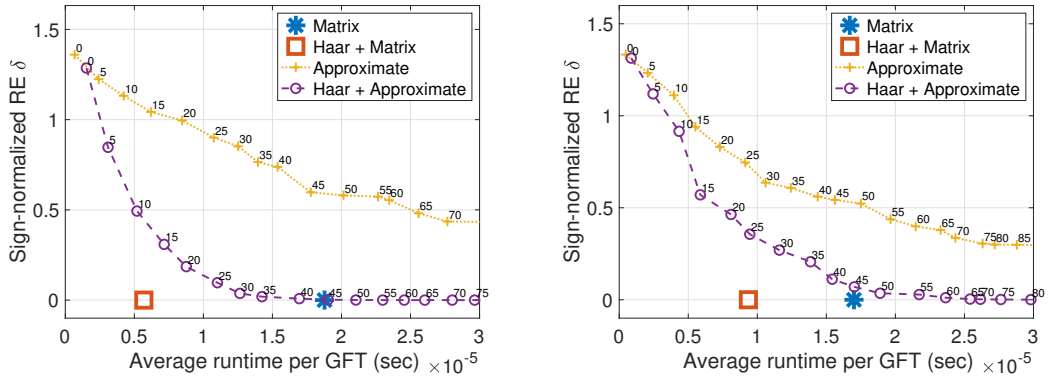
Topology	n	Number of Operations		Runtime
		Matrix ($\pm/*$)	Fast ($\pm/*$)	Reduction
Cycle	12	132/144	44/30	52.7%
	80	6320/6400	1224/1078	79.7%
6-connected grid	16	240/256	80/80	53.7%
	64	4032/4096	1104/1072	68.5%
Z-shaped grid	16	240/256	128/112	41.5%
	64	4032/4096	2048/2048	45.0%
Skeleton	15	210/225	96/102	45.5%
	25	600/625	272/282	47.5%

GFTs can be extended from the examples in Figures 2.8 to 2.12. For each GFT, we generate 20000 graph signals with a proper length, whose entries are i.i.d. uniform random variables with range $[0, 1]$. Then, we compute the percentage of runtime reduction for the symmetry-based fast GFT compared to the GFT realized by a single $n \times n$ matrix multiplication.

In Table 2.3, we show, for each GFT, the numbers of additions (including subtractions), multiplications, and the empirical computation time reduction rate compared to matrix GFT in C implementation. We see that the fast GFT on skeletal graph in Figure 2.12 with one butterfly stage leads to 45.5% speed improvement, and that on z-shaped grid in Figure 2.11 gives around 41.5% runtime saving. Fast GFTs on cycle graphs and 6-connected grids that have multiple butterfly stages yield higher runtime reduction rates. From the results in Table 2.3, we can see that the butterfly stages obtained from our proposed method lead to a significant speedup, and can be useful if the transform is required to be performed many times, and in a low-level or hardware implementation.

2.4.2 Comparison with Approximate Fast GFTs

In the second experiment, we compare our proposed method with an existing fast GFT approach [62] on graphs with symmetry properties. We consider two graphs for this experiment: the 8×8 bi-diagonally symmetric grid with $a = 0.5$, and the 8×8 z-shaped grid with $w = 2$. Note that, when the desired symmetry property is available, existing



(a) 8×8 bi-diagonally symmetric 6-connected grid $\mathcal{G}_b$

(b) 8×8 z-shaped grid $\mathcal{G}_z$

Figure 2.13: Runtime versus sign-normalized relative error δ for different GFT implementations on different graphs. The numbers labeled alongside the markers indicate the associated numbers of Givens rotation layers.

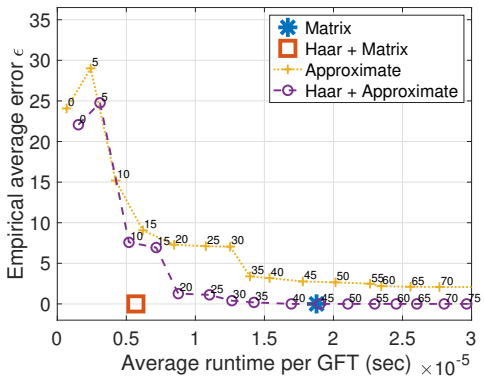
methods can be incorporated into the symmetry-based fast GFT scheme to speed up the computation of sub-GFTs such as $\mathbf{U}^+$ and $\mathbf{U}^-$. Thus, we can compare the following four GFT implementations:

1. *Matrix GFT*: an $n \times n$ matrix multiplication.
2. *Haar-matrix GFT*: symmetry-based fast GFT using Haar units, as shown in Figures 2.10(b) and 2.11(b), where the sub-GFTs are implemented by full matrix multiplications.
3. *PTJ-GFT [62]*: fast GFT using layers of Givens rotations found by the parallel truncated Jacobi (PTJ) algorithm–a greedy-based algorithm that progressively approximate $\hat{\mathbf{U}}^\top \mathbf{L} \hat{\mathbf{U}}$ to a diagonal matrix. The resulting GFT can be implemented using the schematic diagram as in Figure 2.1.
4. *Haar-PTJ-GFT*: symmetry-based fast GFT with sub-GFTs implemented using PTJ-GFT.

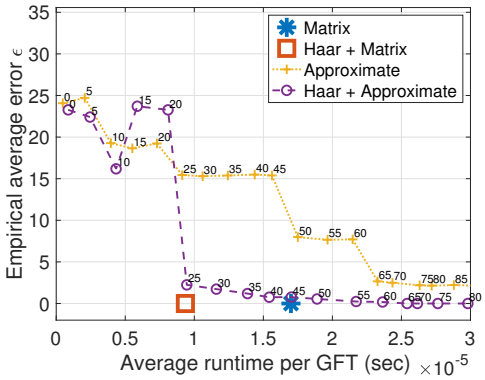
For a given GFT implementation, with GFT matrix $\hat{\mathbf{U}}$ that approximates the true GFT matrix $\mathbf{U}$, we define two error metrics as follows.

1. *Sign-normalized relative error (RE):* we consider the relative error between two $n \times n$ orthogonal matrices,

$$\mathrm{RE}(\hat{\mathbf{U}}, \mathbf{U}) = \frac{\|\mathbf{U} - \hat{\mathbf{U}}\|_F}{\|\mathbf{U}\|_F} = \frac{\|\hat{\mathbf{U}}^\top \mathbf{U} - \mathbf{I}\|_F}{\sqrt{n}}. \tag{2.20}$$



(a) 8×8 bi-diagonally symmetric 6 connected grid $\mathcal{G}_b$



(b) 8×8 z-shaped grid $\mathcal{G}_z$

Figure 2.14: Runtime versus empirical average error ϵ for different GFT implementations on different graphs. The numbers labeled alongside the markers indicate the associated numbers of Givens rotation layers.

Note that if $\hat{\mathbf{U}} = -\mathbf{U}$, the RE will be large although they share a common eigenstructure. To avoid this sign ambiguity, we modify (2.20) by taking absolute values elementwise on $\hat{\mathbf{U}}^\top \mathbf{U}$:

$$\delta(\hat{\mathbf{U}}, \mathbf{U}) := \frac{1}{\sqrt{n}} \| |\hat{\mathbf{U}}^\top \mathbf{U}| - \mathbf{I} \|_F.$$

2. *Empirical average error:* let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ be the set of input signals, we define

$$\epsilon(\hat{\mathbf{U}}, \mathbf{U}, \mathcal{X}) := \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{n} \left(|\mathbf{u}_j^\top \mathbf{x}_i| - |\hat{\mathbf{u}}_j{}^\top (\mathbf{x}_i)| \right)^2,$$

where $\mathbf{u}_j^\top \mathbf{x}_i$ and $\hat{\mathbf{u}}_j$ are the j-th true and approximate GFT coefficients of $\mathbf{x}_i$, and the absolute values are used to avoid the sign ambiguity.

For both graphs considered in this experiment, the eigenvalues of the Laplacians are all unique, so there is no rotation ambiguity in the GFT basis functions.

We apply the method in [62] to obtain the parameters (angle and node pairings for Givens rotations) for PTJ-GFT and Haar-PTJ-GFT, then implement the resulting fast algorithms in C, with different numbers of layers $J \in \{0, 5, 10, \ldots\}$. We use $M = 20000$ random samples as in Section 2.4.1, and obtain the error metrics, δ and ϵ, for each GFT implementation.

The runtime versus sign-normalized RE, and versus empirical average error are shown in Figures 2.13 and 2.14, respectively. We note that, first, the RE drops more steadily

than the empirical error when the number of layers increases. This is related to the order of GFT basis functions. When more layers of Givens rotations are introduced, more GFT basis functions will be ordered correctly (i.e., smaller error in the final permutation operation $\mathbf{\Pi}_{J+1}$ in Figure 2.1). Indeed, we observe that when the number of correctly ordered GFT coefficients increases, the decrease of the empirical error is usually more significant than that of the relative error. The second observation is that for the two graphs with $n = 64$ nodes, the PTJ-based approach typically takes more than 20 layers to yield a sufficiently accurate GFT in terms of both error metrics. However, when more than 20 layers are used, the computation complexity becomes comparable or higher than Haar-matrix GFT, which provides exact GFT coefficients. Finally, we see that in both Figures 2.13 and 2.14, the error of Haar-PTJ-GFT drops faster than that of PTJ-GFT. This means that by applying the symmetry property, we can obtain a significantly higher convergence rate for the PTJ algorithm. This is a reasonable consequence, as our divide-and-conquer method reduces the dimension of the problem for the PTJ algorithm.

2.5 Conclusion

We have explored the relationship between the graph topology and properties in the corresponding GFT for fast GFT algorithm based on butterfly stages. In particular, we focus on a component of the butterfly stage called Haar unit, and discuss the conditions for a stage of Haar units to be available in the GFT implementation. We have shown that a graph has a right butterfly stage with Haar units if it is k-regular bipartite (or if it is bipartite when we consider the symmetric normalized graph Laplacian). On the other hand, a left butterfly stage is available if the graph has symmetry properties. We have formally defined the relevant graph symmetry based on involution, i.e., pairing of nodes. Then, we have proposed an approach, where once a graph symmetry is identified, we can decompose a graph $\mathcal{G}$ into two smaller graphs, $\mathcal{G}^+$ and $\mathcal{G}^-$, whose GFTs corresponds to the two parallel sub-transforms after a butterfly stage of Haar units. Again, from $\mathcal{G}^+$ and $\mathcal{G}^-$ we can explore subsequent butterfly stages if any desired symmetry property holds in them. Thus, this method enables us to explore butterflies stage by stage.

The desired symmetry properties typically arise in graphs that are nearly regular, symmetric by construction, or uniformly weighted. We have discussed several classes of those graphs: bipartite graphs, graphs with 2-sparse eigenvectors such as star and complete graphs, symmetric line and grid graphs, cycle graphs, and skeletal graphs. Relevant applications of those GFTs include video compression and human action analysis. In particular, the fast GFT of a grid graph provides an efficient implementation of a non-separable transform for video blocks; a fast GFT on skeletal graph can also speed

up the feature extraction procedure for further action classification tasks. Finally, we implement the fast GFT algorithms in C and compute the runtime saving for several graphs. The experiment results show that our method provides a significant computation time reduction compared to the GFT computed by matrix multiplication. It also outperforms existing fast approximate GFT approaches in terms of both complexity and accuracy for graphs with desired symmetry properties.

Chapter 3

Data-Driven Fast GFTs for Video Coding

In Chapter 2, we have discussed conditions of graphs for fast GFT algorithms. Here, we apply some of the results to an important graph signal processing application: image and video coding, with a focus on transform coding.

Transform coding is a key component in image and video coding applications [41]. Its basic idea is to decorrelate pixel data using a transform kernel, and hence reduce the redundancies among pixels. The DCT [117] is still by far the most widely used transform in image and video coding due to its fast implementation. However, it has been demonstrated that better compression efficiency can be achieved by data-driven transforms that are obtained based on statistical properties of the residual blocks [2, 146, 147]. The Karhunen-Loève Transform (KLT), which is known to provide the optimal decorrelation, has been proposed for designing mode-dependent transforms of residual blocks [139]. However, its computational complexity is typically very high, which has limited its practical use.

Recently, it has been shown that GFT is useful for decorrelating particular types of pixel data [24, 33, 50]. However, one of the major challenges of GFT in transform coding is that, similar to the KLT, those GFTs that optimally adapt to the data are usually computationally expensive. To address this complexity issue, we note that based on discussions in Chapter 2, certain graph structural properties, such as symmetry and bipartition, would lead to a computation cost reduction in GFT implementations. Indeed, once the graph to be learned is restricted to satisfy those structural properties, a GFT with fast algorithm can be obtained.

In this chapter, we study data-driven fast GFTs to enhance efficiency in transform coding techniques. To achieve this goal, we consider a graph learning problem to obtain the graph from sample data. This can be viewed as the problem of learning a data-driven GFT that adapts to statistical properties of target image/video blocks. Furthermore, we restrict the graphs to be symmetric or to have a butterfly stage so that a fast algorithm

---

Work in this chapter has been published in [70, 71].

is available to compute the GFT. First, we introduce a general framework for designing data-driven fast GFT in Section 3.1. Under this framework, we optimize parameters of symmetric line graphs (as discussed in Section 2.3.3) to obtain efficient separable transforms that provide a coding gain in Section 3.2. Then, we propose a method for non-separable transform extensions with unknown butterfly stages in Section 3.3. Experimental results are shown in Section 3.4.

3.1 Learning Fast GFTs: A General Framework

Graph learning is an important problem in graph signal processing [22]. This problem arises from the fact that, given real-world data such as pixel blocks, we may not know ahead of time what the best graph is to develop graph signal processing tools. In practice, the graph can be learned from data based on the statistical properties [25], global smoothness [25], diffusion kernel estimation [92, 108], causal dependency [84] and so on. In what follows, among different graph learning problem formulations, we adopt the one formulated in [25] and incorporate structural constraints associated to fast GFT implementations. This allows us to obtain practically useful GFTs that have fast algorithms and adapt to particular set of pixel data of interest.

We start by formulating a data-driven fast GFT design problem. We denote $\mathbf{x}_i$'s training samples (e.g., these could be image or video blocks used for training), and $\boldsymbol{\mu}$ their sample mean (which is usually assumed to be $\mathbf{0}$ when dealing with prediction residual blocks of a video). Then, we let $\mathbf{S} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top$ be the empirical covariance matrix. We start with the Gaussian maximum likelihood (ML) estimation problem [25] for the graph Laplacian precision matrix $\mathbf{L}$:

$$\underset{\mathbf{L}\in\mathbb{L}(\mathcal{E})}{\text{minimize}} \quad -\log|\mathbf{L}|_\dagger + \text{trace}(\mathbf{LS}), \tag{3.1}$$

where the objective function is the negative log-likelihood function of an attractive GMRF whose precision matrix is $\mathbf{L}$. In (3.1), $\mathbb{L}(\mathcal{E})$ is the set of graph Laplacians with edge set $\mathcal{E}$. The pseudo-determinant (product of nonzero eigenvalues) $|\mathbf{L}|_\dagger$ is required here when a combinatorial graph Laplacian matrix, which is singular, is considered. Note that, for any $\mathcal{E}$, (3.1) is a convex problem, where general solution can be obtained using efficient iterative approaches [25, 94].

For image and video coding, pixels are usually modeled as 1st order GMRFs, where each node is connected to its nearest neighbors in the associated graph. In particular, the following graph structures $\mathcal{E}$ (as shown in Figure 3.1) may be considered:

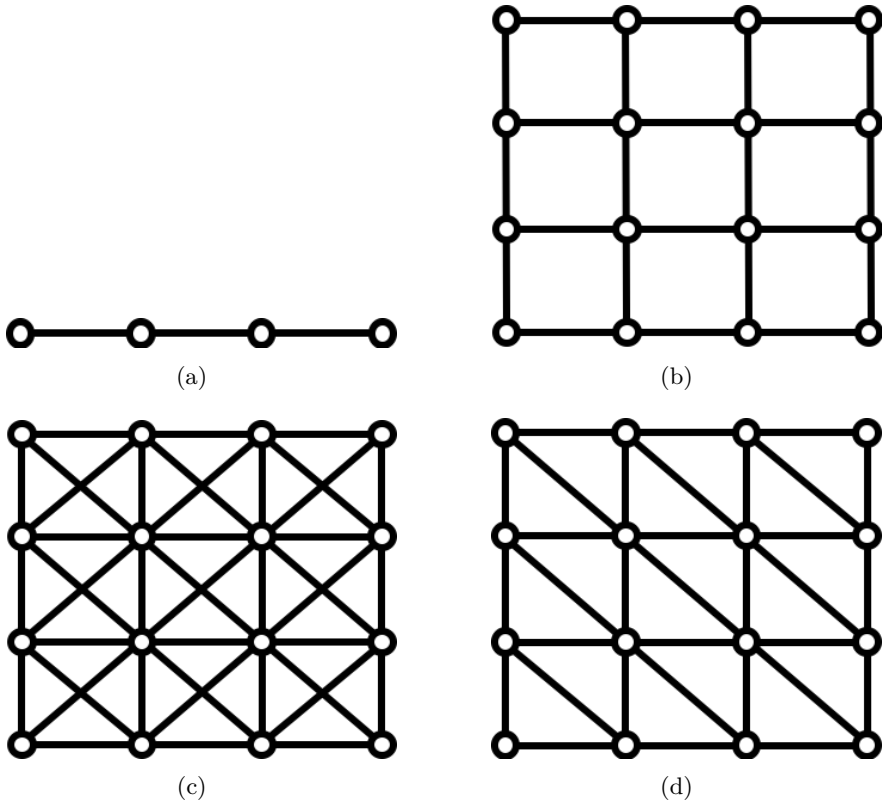


Figure 3.1: Useful graph structures in modeling image and video pixels: (a) length-4 line graph (b) 4×4 4-connected grid, (c) 4×4 8-connected grid, and (d) 4×4 6-connected grid.

- 1D line graph: it corresponds to a 1st order GMRF for 1D pixel data (a row or a column of a 2D pixel block). Its associated GFT could be used as a 1D row or column transform.
- 2D 4-connected grid graph: it is associated to a 1st order GMRF for 2D pixel data. The resulting GFT corresponds to a 2D block transform.
- 2D 8-connected and 6-connected grid graphs: diagonal edges can be added to the 4-connected grid to capture correlations between each pixel and its 8-neighbors. Those connections are particularly useful when the image/video content has diagonal orientations.

These graphs with length 4 or grid size 4×4 are shown in Figure 3.1.

Our results from Chapter 2 for fast GFTs are mainly based on factorization of Laplacian $\mathbf{L} = \mathbf{BGB}^\top$ into a matrix of left Haar units $\mathbf{B}$ and a block-diagonal matrix $\mathbf{G}$ (see

Section 2.1.2.). The fast GFT associated to $\mathbf{L}$ is $\mathbf{U} = \mathbf{B}\mathbf{U}_\mathbf{G}$, where $\mathbf{U}_\mathbf{G}$ is the eigenvector matrix of $\mathbf{G}$. This speedup is led by the fact that $\mathbf{B}$ is sparse, and $\mathbf{U}_\mathbf{G}$ has the block-diagonal structure as $\mathbf{G}$ does. Here, we generalizing the factorization $\mathbf{L} = \mathbf{B}\mathbf{G}\mathbf{B}^\top$ to $\mathbf{L} = \mathbf{H}\mathbf{R}\mathbf{H}^\top$ with a sparse $\mathbf{H}$ (not necessarily $\mathbf{B}$) and block-diagonal $\mathbf{R}$, such that there is a fast GFT of $\mathbf{L}$. In particular, we write $\mathbf{L} = \mathbf{H}\mathbf{R}\mathbf{H}^\top$, with constraints on $\mathbf{H}$ and $\mathbf{G}$ given by criteria C1 and C2:

C1 $\mathbf{H}$ is orthogonal, and each of its columns is a constant multiple of a vector, on which the projection can be implemented efficiently. That is, $\mathbf{H} = \mathbf{G}_\mathbf{H}\mathbf{D}_\mathbf{H}$, where $\mathbf{D}_\mathbf{H}$ is diagonal, $\mathbf{G}_\mathbf{H}$ has orthogonal columns, and $\mathbf{G}_\mathbf{H}^\top\mathbf{x}$ has fast implementation. Here are two specific criteria that allow fast implementations:

- C1-A All entries in $\mathbf{G}_\mathbf{H}$ are 0, 1, or -1.
- C1-B $\mathbf{G}_\mathbf{H}$ is sparse.

C2 $\mathbf{R}$ is block diagonal, and as sparse as possible, i.e., having many smaller blocks is preferable.

Let the eigenmatrix of $\mathbf{R}$ be $\mathbf{U}_\mathbf{R}$, then the GFT of signal $\mathbf{x}$ is $\mathbf{U}_\mathbf{R}^\top\mathbf{D}_\mathbf{H}^\top\mathbf{G}_\mathbf{H}^\top\mathbf{x}$. Note that multiplying by $\mathbf{G}_\mathbf{H}^\top$ is efficient, and $\mathbf{U}_\mathbf{R}^\top\mathbf{D}_\mathbf{H}^\top$ has the same block diagonal structure as $\mathbf{R}$ does. Thus, the sparsity of $\mathbf{R}$ enables reduction in the computation of this GFT.

Writing $\mathbf{L} = \mathbf{H}\mathbf{R}\mathbf{H}^\top$ with C1 and C2 imposed, we have $\det(\mathbf{L}) = \det(\mathbf{R})$ and $\operatorname{trace}(\mathbf{L}\mathbf{S}) = \operatorname{trace}(\mathbf{H}\mathbf{R}\mathbf{H}^\top\mathbf{S}) = \operatorname{trace}(\mathbf{R}(\mathbf{H}^\top\mathbf{S}\mathbf{H}))$. Thus, (3.1) can be rewritten as

$$\begin{aligned} \underset{\mathbf{H},\mathbf{R}}{\text{minimize}} \quad & -\log\det(\mathbf{R}) + \operatorname{trace}(\mathbf{R}(\mathbf{H}^\top\mathbf{S}\mathbf{H})) \\ \text{subject to} \quad & \mathbf{R} \succeq 0,\ \left(\mathbf{H}\mathbf{R}\mathbf{H}^\top\right)_{i,j} = 0 \text{ for } (i,j) \notin \mathcal{E}, \quad \left(\mathbf{H}\mathbf{R}\mathbf{H}^\top\right)_{i,j} \le 0 \text{ for } (i,j) \in \mathcal{E} \\ & \mathbf{H} \text{ satisfies C1, } \mathbf{R} \text{ satisfies C2.} \end{aligned} \tag{3.2}$$

The problem (3.2) involves a product $\mathbf{H}\mathbf{R}\mathbf{H}^\top$ of unknown matrices, so it is a nonconvex problem that may not have efficient solvers. However, when $\mathbf{H}$ that satisfies C1 is available, (3.2) reduces to a convex problem. This special case will be studied as follows.

3.1.1 Solving (3.2) with a Particular Symmetry Type

As discussed in Section 2.2, when the graph is symmetric in a certain way, we obtain a fast GFT: by Lemma 3, $\mathbf{L} = \mathbf{B}_N\mathbf{G}\mathbf{B}_N^\top$ with block-diagonal $\mathbf{G}$ if $\mathbf{L}$ is centrosymmetric, which is associated to symmetry properties of the graph. In this case, a fast GFT $\mathbf{U}$ can be obtained using the diagram in Figure 2.2(a), which can be expressed as

$$\mathbf{U} = \mathbf{B}_N \cdot \operatorname{diag}(\mathbf{U}^+, \mathbf{U}^-),$$

where $\mathbf{B}_N$ is the orthogonal matrix defined in (2.2) that corresponds to a butterfly stage of Haar units, and $\text{diag}(\mathbf{U}^+, \mathbf{U}^-)$ is a block-diagonal matrix associated to two parallel length $N/2$ transforms.

From the above statement, if we restrict the graph to be symmetric, an $\mathbf{H}$ with some desirable properties will arise, and the sizes of block diagonal elements of $\mathbf{R}$ will be known. Then, we can show that (3.2) would reduce to a convex problem. We denote $\mathbf{R} = \text{diag}(\mathbf{R}_1, \ldots, \mathbf{R}_M)$, where element $\mathbf{R}_i$ has size $k_i \times k_i$ and $\sum_{i=1}^M k_i = n$. We also write $\mathbf{H}$ as $\mathbf{H} = (\mathbf{H}_1, \ldots, \mathbf{H}_M)$, where $\mathbf{H}_i$, with size $n \times k_i$, is a vertical slice of $\mathbf{H}$ containing k_i columns. Thus, $\mathbf{H}\mathbf{R}\mathbf{H}^\top = \sum_{l=1}^M \mathbf{H}_l \mathbf{R}_l \mathbf{H}_l^\top$, and

$$\left(\mathbf{H}\mathbf{R}\mathbf{H}^\top\right)_{i,j} = \mathbf{e}_i^\top \left(\sum_{l=1}^M \mathbf{H}_l \mathbf{R}_l \mathbf{H}_l^\top\right) \mathbf{e}_j = \sum_{l=1}^M \mathbf{h}(i,l) \mathbf{R}_l \mathbf{h}(j,l)^\top,$$

where $\mathbf{h}(i,l)$ denotes the i-th row of $\mathbf{H}_l$. We define $\mathbf{\Theta}$ as

$$\mathbf{\Theta} := \mathbf{H}^\top \mathbf{S} \mathbf{H} = \begin{pmatrix} \mathbf{\Theta}_{1,1} & \mathbf{\Theta}_{1,2} & \ldots & \mathbf{\Theta}_{1,M} \\ \mathbf{\Theta}_{2,1} & \mathbf{\Theta}_{2,2} & \ldots & \mathbf{\Theta}_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Theta}_{M,1} & \mathbf{\Theta}_{M,2} & \ldots & \mathbf{\Theta}_{M,M} \end{pmatrix},$$

where $\mathbf{\Theta}_{i,j}$ has size $k_i \times k_j$, then (3.2) can be reduced to a convex problem:

$$\begin{aligned} \underset{\mathbf{R}_i \in \mathbb{R}^{k_i \times k_i}}{\text{minimize}} \quad & \sum_{l=1}^M \left[-\log \det(\mathbf{R}_l) + \text{trace}(\mathbf{R}_l \mathbf{\Theta}_{l,l})\right] \\ \text{subject to} \quad & \mathbf{R}_i \succeq 0, \ \sum_{l=0}^M \mathbf{h}(i,l) \mathbf{R}_l \mathbf{h}(j,l)^\top \leq 0, \ i \neq j. \end{aligned} \tag{3.3}$$

The number of variables is reduced from $N^2 = (\sum_{l=1}^M k_l)^2$ to $\sum_{l=1}^M k_l^2$. Since $\mathbf{H}$ is typically sparse, the constraints in (3.3) are simplified in many practical cases. Note that if $\mathbf{\Theta}_{i,j}$ is zero for all $i \neq j$, the solution of (3.3) is the same as that of (3.1).

Based on this framework of data-driven fast GFT, we devote the next two sections to particular cases: symmetric line graph (Section 3.2) and non-separable fast GFT without any specified symmetry (Section 3.3).

3.2 Symmetric Line Graph Transforms (SLGT)

As introduced in 3.1, first order GMRF are associated to line graphs. Based on the connection between graph symmetry and Haar units we have introduced in Section 2.2,

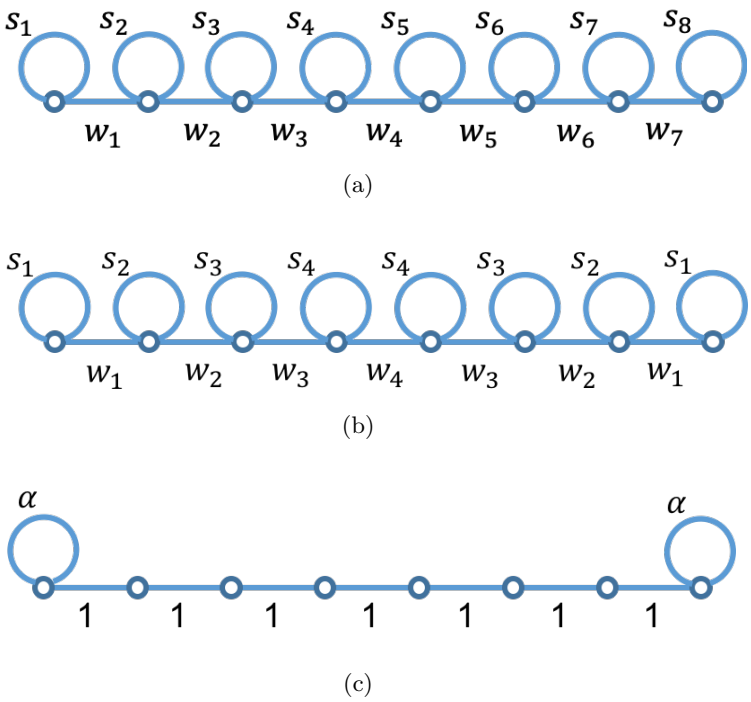
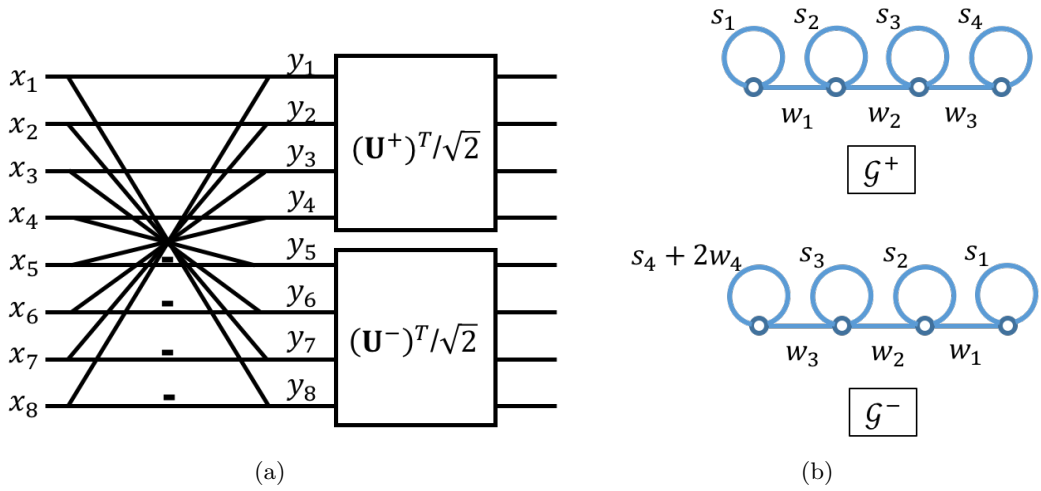


Figure 3.2: (a) An arbitrary length 8 line graph. (b) A length 8 SLGT. (c) A special length-8 SLGT.

we note that among all line graphs, those having a butterfly stage of Haar units are symmetric line graphs. The GFT of a symmetric line graph is thus called a *symmetric line graph transform* (SLGT).

With a length 8, an arbitrary line graph can be represented as in Figure 3.2(a), while a symmetric line graph represented in Figure 3.2(b). The flow diagram for the computation of a general 8×8 SLGT is shown in Figure 3.3, where $\mathbf{U}^+$ and $\mathbf{U}^-$ are 4×4 transforms characterized by the subgraphs $\mathcal{G}^+$ and $\mathcal{G}^-$, obtained from the decomposition described in Section 2.2.3. This fast GFT diagram can be extended to any even size N[1]. For any even number N, both $\mathbf{U}^+$ and $\mathbf{U}^-$ have at most $N^2/4$ multiplications, meaning that the overall $N \times N$ SLGT has at most $N^2/2$ multiplications. This is half the number of multiplications required by a general $N \times N$ transform such as the KLT.

[1]As we have pointed out in Section 2.3.3, there also exists a butterfly stage of Haar units for a symmetric line graph with an *odd* number of nodes. However, we leave the discussion for future work since transforms with odd lengths are rarely used in existing image and video coding systems.



(a) (b)

Figure 3.3: (a) The diagram of general 8×8 SLGT implementation. (b) Graphs that characterize the sub-transforms in the fast GFT diagram.

3.2.1 DTTs as SLGTs

There are some well-known transforms that can be viewed as SLGTs. For example, among 16 types of DTTs, the DCT-II, DST-I, and DST-II, as defined in Section 1.2.4 correspond to Laplacian matrices $\mathbf{L}_{\text{DCT-II}}$ (1.10), and

$$\mathbf{L}_{\text{DST-I}} = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad \mathbf{L}_{\text{DST-I}} = \begin{pmatrix} 3 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 3 \end{pmatrix},$$

respectively. Their associated graphs are uniform line graphs with two self loops with weights $\alpha = 0$, $\alpha = 1$, and $\alpha = 2$, respectively, as shown in Figure 3.2(c). These three transforms are known to have fast algorithms for lengths satisfying certain conditions [9, 134, 140].

3.2.2 Closed-Form Graph Learning Solution with Tree Topologies

Here, we highlight a property that provides a closed form solution for learning a simple (i.e., loopless) line graph[2]. More generally, the close-form solution would be available if

[2]The main result of this section has been published in our work [77].

the graph to be learned is a tree (acyclic graph), where line graph is a particular case. Those results are potentially useful because 1) those results applies to trees that are more general than line graphs 2) trees are useful graphs that provide computational benefits. In particular, trees are the sparsest connected graphs, so graph based filters [105, 114] have the lowest complexity when implemented on trees. Furthermore, tree are bipartite, hence perfect reconstruction filterbanks on graphs can be easily implemented [87].

We consider the graph estimation problem with an ℓ_1 regularization term, which is more general than (3.1):

$$\underset{\mathbf{L}\in\mathbb{L}(\mathcal{E})}{\text{minimize}} \quad -\log|\mathbf{L}|_\dagger + \text{trace}(\mathbf{LS}) + \alpha\|\mathbf{L}\|_{1,\text{off}}, \tag{3.4}$$

where α is a Lagrange multiplier and $\|\mathbf{L}\|_{1,\text{off}}$ is the absolute sum of all off-diagonal elements of $\mathbf{L}$. The role of the last regularization term in (3.4) is to promote graph sparsity, which also corresponds to an exponential prior of the GMRF parameters [25].

In fact, closed-form solution for (3.4) is available if the topology set $\mathcal{E}$ corresponds to a tree graph:

Theorem 3. *If $\mathcal{E}$ corresponds to a simple tree graph $\mathcal{G}$, then the graph weights for the optimal CGL solution of (3.4) are given by*

$$w_{s,t} = \begin{cases} \left[(\mathbf{e}_s - \mathbf{e}_t)^\top \mathbf{K}(\mathbf{e}_s - \mathbf{e}_t)\right]^{-1} = \left[\frac{1}{N}\sum_{i=1}^N (\mathbf{x}_i(s) - \mathbf{x}_i(t))^2 + 2\alpha\right]^{-1}, & (s,t)\in\mathcal{E}, \\ 0, & (s,t)\notin\mathcal{E}. \end{cases} \tag{3.5}$$

where $\mathbf{K} = \mathbf{S} + \alpha(\mathbf{I} - \mathbf{1}\mathbf{1}^\top)$.

Proof: see Appendix A.3.

Using this closed form solution, the complexity of solving a tree CGL can be significantly reduced. Given the matrix $\mathbf{K}$, constructing the weights from (3.5) has an overall complexity of $\mathcal{O}(n-1) = \mathcal{O}(n)$. In contrast, the algorithm proposed in [25] has $\mathcal{O}(T_p(n) + n^2)$ complexity per block-coordinate descent iteration, where $T_p(n)$ is the complexity of solving a nonnegative quadratic program of size n.

3.3 Data-driven Non-separable Fast GFTs

In Section 3.1.1 we solved (3.2) for known $\mathbf{H}$. In this section, we provide an approximate method when $\mathbf{H}$ in (3.2) is not known ahead of time.

When $\mathbf{H}$ is not given, (3.2) is a nonconvex problem. We propose an efficient approximation method as follows. We factor N as $N = N_1 N_2$ and approximate $\mathbf{S}$ by the Kronecker product of two matrices of sizes $N_2 \times N_2$ and $N_1 \times N_1$:

$$\mathbf{S} \approx \mathbf{S}_2 \otimes \mathbf{S}_1. \tag{3.6}$$

The optimal approximation in terms of Frobenius norm is given by the optimal rank-1 approximation of $\tilde{\mathbf{S}}$, a matrix with size $N_2^2 \times N_1^2$, whose entries are those of $\mathbf{S}$ with rearrangement [98, 131]:

$$\tilde{\mathbf{S}} = \begin{pmatrix} \mathrm{vec}(S_{1,1})^\top \\ \vdots \\ \mathrm{vec}(S_{N_2,1})^\top \\ \mathrm{vec}(S_{1,2})^\top \\ \vdots \\ \mathrm{vec}(S_{N_2,N_2})^\top \end{pmatrix}, \quad \text{with } \mathbf{S} = \begin{pmatrix} S_{1,1} & S_{1,2} & \dots & S_{1,N_2} \\ S_{2,1} & S_{2,2} & \dots & S_{2,N_2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{N_2,1} & S_{N_2,2} & \dots & S_{N_2,N_2} \end{pmatrix},$$

where $S_{i,j}$'s are $N_1 \times N_1$ block components of $\mathbf{S}$. It can be shown that $\mathbf{S}_2$ and $\mathbf{S}_1$ can be obtained by solving the first left and right singular vectors of $\tilde{\mathbf{S}}$ followed by vector reshaping:

Lemma 6 ([131]). *For an $N \times N$ matrix $\mathbf{S}$, the optimal solution of*

$$(\mathbf{S}_1^*, \mathbf{S}_2^*) = \underset{\mathbf{S}_1 \in \mathbb{R}^{N_1 \times N_1},\, \mathbf{S}_2 \in \mathbb{R}^{N_2 \times N_2}}{\mathit{argmin}} \quad \|\mathbf{S} - \mathbf{S}_2 \otimes \mathbf{S}_1\|^2$$

can be obtained from the rank-1 approximation of $\tilde{\mathbf{S}}$. In particular, denote $\boldsymbol{\mu} = (\mu_1, \dots, \mu_{N_2^2})^\top$ and $\boldsymbol{\nu} = (\nu_1, \dots, \nu_{N_1^2})^\top$ the first left and right singular vectors of $\tilde{\mathbf{S}}$ associated to singular value σ_1, then

$$\mathbf{S}_1^* = \begin{pmatrix} \nu_1 & \nu_{N_1+1} & \dots & \nu_{N_1^2-N_1+1} \\ \nu_2 & \nu_{N_1+2} & \dots & \nu_{N_1^2-N_1+2} \\ \vdots & \vdots & \ddots & \vdots \\ \nu_{N_1} & \nu_{2N_1} & \dots & \nu_{N_1^2} \end{pmatrix}, \quad \mathbf{S}_2^* = \begin{pmatrix} \mu_1 & \mu_{N_2+1} & \dots & \mu_{N_2^2-N_2+1} \\ \mu_2 & \mu_{N_2+2} & \dots & \mu_{N_2^2-N_2+2} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{N_2} & \mu_{2N_2} & \dots & \mu_{N_2^2} \end{pmatrix}.$$

Next, with $\mathbf{S}_1$ and $\mathbf{S}_2$ obtained from Lemma 6, we denote the eigendecomposition of $\mathbf{S}_2$ as $\mathbf{S}_2 = \mathbf{E}_2 \mathbf{D}_2 \mathbf{E}_2^\top$, then

$$\mathbf{S} \approx \left(\mathbf{E}_2 \mathbf{D}_2 \mathbf{E}_2^\top\right) \otimes \left(\mathbf{I}_{N_1} \mathbf{S}_1 \mathbf{I}_{N_1}\right) = \left(\mathbf{E}_2 \otimes \mathbf{I}_{N_1}\right) \left(\mathbf{D}_2 \otimes \mathbf{S}_1\right) \left(\mathbf{E}_2 \otimes \mathbf{I}_{N_1}\right)^\top. \tag{3.7}$$

Algorithm 1 Proposed Approximation Method for Non-separable Fast GFT

Require: $\mathbf{S} \in \mathbb{R}^{N \times N}$, a chosen factorization $N = N_1 N_2$

Ensure: $(\mathbf{H}, \mathbf{R})$ as a solution of (3.2)

1: Evaluate $\mathbf{S}_1$ and $\mathbf{S}_2$ such that $\mathbf{S} \approx \mathbf{S}_2 \otimes \mathbf{S}_1$

2: Evaluate the eigenmatrix $\mathbf{E}_2$ of $\mathbf{S}_2$

3: $\mathbf{H} \leftarrow \mathbf{E}_2 \otimes \mathbf{I}_{N_1}$

4: Solve (3.3) for $\mathbf{R}_i$, given $\mathbf{H}$ and $k_i = N_1$ for each i

5: $\mathbf{R} \leftarrow \text{diag}(\mathbf{R}_1, \ldots, \mathbf{R}_{N_2})$

If N_2 is small, $\mathbf{E}_2 \otimes \mathbf{I}_{N_1}$ is sparse; thus, it is a legitimate matrix satisfying C1-B that approximately block-diagonalizes $\mathbf{S}$ to $\mathbf{D}_2 \otimes \mathbf{S}_1$. Therefore, we pick $\mathbf{E}_2 \otimes \mathbf{I}_{N_1}$ as the $\mathbf{H}$ matrix, and use the block structure of $\mathbf{D}_2 \otimes \mathbf{S}_1$, to solve (3.3) and obtain $\mathbf{R}$ which has $M = N_2$ nonzero $N_1 \times N_1$ diagonal blocks.

We summarize this proposed method in Algorithm 1. For the choice of N_1 and N_2, we note that it is preferable to choose $N_1 \approx N_2$. With the resulting $\mathbf{H}$ and $\mathbf{R}$ matrices, the number of multiplications required for the fast GFT is $N(N_1 + N_2)$, which is smaller when N_1 and N_2 are closer to each other. In addition, when N_1 is closer to N_2, there are more degrees of freedom in $\mathbf{S}_1 \otimes \mathbf{S}_2$, giving a potentially better approximation in (3.6).

If the signal is a pixel block with size $N_1 \times N_2$, the eigenmatrices $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{S}_1$ and $\mathbf{S}_2$ characterize the column and row transforms in a separable scheme. Therefore, the GFT obtained by the proposed method can be regarded as a non-separable transform obtained from a separable one: the solver of (3.3) updates $\mathbf{D}_2 \otimes \mathbf{S}_1$ in (3.7) into an arbitrary block diagonal matrix (not necessarily a Kronecker product matrix), which has more degrees of freedom. The resulting fast GFT can be realized by a common row transform applied to all rows followed by different column transforms applied to different columns[3]. The number of multiplications is the same as a separable transform. The number of required coefficients is $NN_1 + N_2^2$, which is between the numbers in the separable case $(N_1^2 + N_2^2)$ and the nonseparable case (N^2).

3.4 Experimental Results

In this section, we apply data-driven GFTs introduced in Sections 3.2 and 3.3 to video residual blocks, and obtain the resulting rate-distortion (RD) performance. In Section 3.4.1, we obtain a data-driven separable GFT composed of SLGTs (as described in

[3]Note that when the input covariance matrix has row-first node ordering, the same method will yield an approximate transform with a common column transform and different row transforms.

Table 3.1: Parameters of learned symmetric line graphs, and those associated with DCT. The parameters are weights of self loops and edges, as shown in Figure 3.2(b).

	s_1	s_2	s_3	s_4	w_1	w_2	w_3	w_4
$\mathcal{G}_{\mathrm{V}}$	0.15	0	0.02	0.02	0.94	0.96	1	0.90
$\mathcal{G}_{\mathrm{H}}$	0.08	0	0.01	0.03	1	0.93	1	0.94
$\mathcal{G}_{\mathrm{DCT}}$	0	0	0	0	1	1	1	1

Table 3.2: Average percentage of bit rate reduction of SLGTs as compared to DCT at 32dB-38dB PSNR. Results with and without frequency weighting are compared. Negative values mean compression gain.

	w/o freq. weighting	w/ freq. weighting
`BQMall`	0.04	-0.10
`BasketballDrill`	0.09	-0.19
`Kimono1`	-0.75	-1.40
`Cactus`	-0.03	-0.20
`ParkScene`	-0.82	-0.58
`BQTerrace`	-0.31	-0.17
Average	-0.39	-0.51

Section 3.2), and evaluate its rate-distortion (RD) performance on a dataset of inter predicted blocks. Then, in Section 3.4.2, we apply the approach introduced in Section 3.3 to learn mode-dependent non-separable GFTs for intra predicted blocks in particular prediction modes, and present the resulting RD performance.

3.4.1 Data-Driven SLGTs for Inter Predictive Coding

In this experiment, we obtain a 2D separable transform composed of a row SLGT and a column SLGT, and apply it to inter predictive coding. We use a dataset of 8×8 inter predicted luma blocks, extracted with the HEVC test model (HM-16.9) from several video sequences: `BasketballDrive`, `BQMall`, `BasketballDrill`, `Kimono1`, `Cactus`, `ParkScene`, and `BQTerrace`, where `BasketballDrive` is used as the training set and the other sequences are used for testing.

We solve the problem (3.3) with $\mathbf{H} = \mathbf{B}_N$ and $\mathcal{E}$ corresponding to a line graph topology. For the column transform, we use 1D vertical vectors of length 8 taken from the 8×8 residuals in the training set, to obtain $\mathbf{L}_{\mathrm{V}}$, the 8×8 empirical covariance matrix. For the row transform, we repeat the same procedure using 1D horizontal vectors taken from the same training set to obtain the empirical covariance matrix $\mathbf{L}_{\mathrm{H}}$. Since most residual blocks are expected to be well approximated by only a few low frequency eigenvectors at compression rates of interest, we modify the optimization criterion of (3.3) by increasing

the penalty on the approximation error on the lower frequency basis. To achieve this, we apply a simple heuristic based on frequency weighting to the covariance matrix [95]. Letting $\mathbf{L}_{\mathrm{V}} = \mathbf{U}_{\mathrm{V}}\mathbf{\Lambda}_{\mathrm{V}}\mathbf{U}_{\mathrm{V}}^{\top}$ and $\mathbf{L}_{\mathrm{H}} = \mathbf{U}_{\mathrm{H}}\mathbf{\Lambda}_{\mathrm{H}}\mathbf{U}_{\mathrm{H}}^{\top}$, we replace $\mathbf{\Lambda}_{\mathrm{V}}$ and $\mathbf{\Lambda}_{\mathrm{H}}$ by $\mathbf{\Lambda}_{\mathrm{V}}^2$ and $\mathbf{\Lambda}_{\mathrm{H}}^2$, respectively, then use the modified covariance matrix as input of the program (3.3) to estimate the Laplacian matrices $\mathbf{L}_{\mathrm{V}}$ and $\mathbf{L}_{\mathrm{H}}$ and hence the symmetric line graphs $\mathcal{G}_{\mathrm{V}}/\mathcal{G}_{\mathrm{H}}$. In this way, the effects of eigenvectors corresponding to low frequencies are magnified in the objective function of (3.3). As a result, the first few basis functions of the learned SLGT are expected to better approximate those of KLT. In the implementation of this experiment, self-loops are allowed to provide more degrees of freedoms for better coding results[4]. We use the CVX package [42] to obtain the parameters. Those weights of self loops and edges, as depicted in Figure 3.2(b), are shown in Table 3.1. The eigenmatrices of $\mathbf{L}_{\mathrm{V}}$ and $\mathbf{L}_{\mathrm{H}}$ are SLGTs that we use as column and row transforms for testing data.

The rate-distortion (RD) performance of the proposed transform is compared with that of DCT. After the transforms, the coefficients are uniformly quantized, then entropy-encoded using the AGP encoder [102]. The AGP codec combines an alphabet partitioning and a set partitioning techniques, which are learned based on the distribution of the transform coefficients; thus, this codec can provide a fair comparison between different transforms. Since the column/row transform scheme is always applied to all columns/rows, no side information is required for uniquely decoding the blocks.

The bitrate gains in Bjontegaard-delta measure (BD-rate) [4] at 32dB-38dB peak-signal-to-noise-ratio (PSNR), with and without frequency weighting, are shown in Table 3.2, where the coding results with DCT coefficients is used as the baseline. According to the table, we have the following remarks from the results. First, the proposed transform achieves an average bitrate gain of 0.51% as compared to DCT. In average, the gain of SLGT is the highest at distortion level of 32dB, and drops when it gets higher. As our proposed SLGT better approximates the first few basis functions of a learned KLT, better decorrelation results at lower bit rate is expected. Second, the frequency weighting procedure provides a more consistent gain in the 6 test sequences compared to the scheme without frequency weighting. The most significant bitrate gain is observed in `Kimono1` sequence, with frequency weighting applied. Indeed, the residual signal of `Kimono1` has the smallest energy in average, and such blocks are more likely to be represented by a small number of SLGT coefficients. Thus, the first few basis functions at low frequency, where the SLGT approximates the KLT with a better accuracy, are particularly important for those smooth blocks.

[4]If self-loops are not allowed, the convex program can be solved using the closed form solution described in Section 3.2.2.

Table 3.3: Bit rate reduction of separable KLT, fast GFT, and hybrid DCT/ADST as compared to the 2D separable DCT, tested on residual blocks with intra mode-2 and mode-16. Negative values mean compression gain, measured in the Bjontegaard metric (percentage of bit rate reduction).

	Separabe KLT		Fast GFT		DCT+ADST	
	Mode-2	Mode-16	Mode-2	Mode-16	Mode-2	Mode-16
`BQMall`	-9.7	-6.3	-12.9	-8.2	-0.7	-1.3
`BasketballDrill`	-11.9	-8.6	-14.5	-10.2	-3.4	-2.5
`Crew`	-17.5	-14.8	-20.7	-16.2	-3.2	-1.7
`Harbour`	-26.6	-24.9	-29.7	-23.0	-1.7	-3.2
`Ice`	-28.4	-11.8	-35.5	-16.8	-3.7	-1.7
`Soccer`	-14.6	-12.7	-15.6	-11.2	-3.9	-1.5
Average	-13.2	-10.4	-20.2	-11.8	-2.8	-2.1

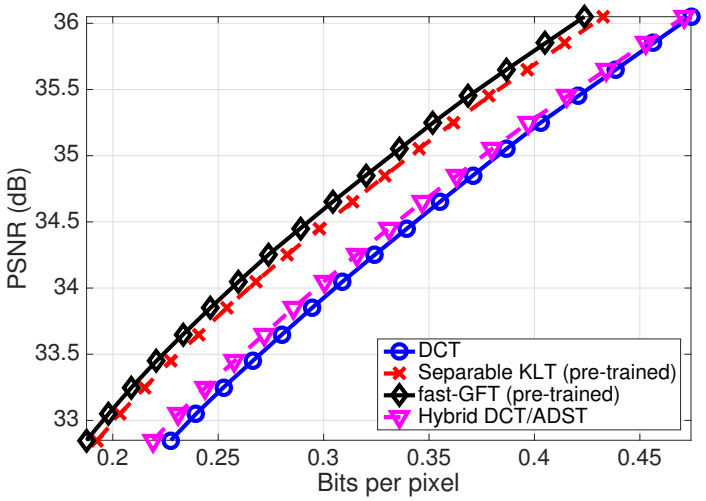


Figure 3.4: Rate-distortion performance of separable KLT, fast GFT, and hybrid DCT/ADST. The testing blocks in this figure are mode-2 intra residual blocks.

3.4.2 Data-Driven Nonseparable GFTs for Intra Blocks

In the second experiment, we apply the learning method proposed in Section 3.3 to intra-predicted blocks. We extract intra-predictive residual blocks from test video sequences `BQMall, BasketballDrill, City, Crew, Harbour, Ice` and `Soccer`, again with HM-16.9. Only 8×8 luma residual blocks with intra prediction mode-2 (HOR+8) and mode-16 (HOR-6), which have nearly diagonal directions of prediction, are considered. We train a fast GFT for mode-2 (resp. mode-16) based on Algorithm 1, using mode-2 (resp. mode-16) blocks in the `City` sequence as the training set. The convex problem (3.2) is

solved using CVX [42] toolbox. Then, we test the results on mode-2 (resp. mode-16) blocks in the other 6 sequences, not including the one for training.

To compare the results, we consider several other transforms with comparable or lower computational costs: the 2D separable DCT, separable KLT, and the mode-dependent hybrid DCT/ADST transform. In the hybrid scheme, we apply a mode-dependent combination of DCT and ADST, as suggested in [107], for the row and column transform. Each KLT (for mode-2 and mode-16, respectively) is pre-trained using the same training set as the fast GFT. The transform coefficients are uniformly quantized using various quantization steps that yield PSNR levels ranging from 29dB to 38dB. As in Section 3.4.1, the quantized coefficients are encoded using the AGP codec [102].

Table 3.3 shows the coding gain in BD-rate (percentages of bit rate reduction) over 2D DCT. The RD plot for mode-2 blocks is shown in Figure 3.4. The resulting fast GFT achieves an average of 20.2% bit rate gain over DCT for mode-2 blocks, and 11.8% for mode-16 blocks. It also outperforms the hybrid transform for each sequence/mode. In most cases, the fast GFT also gives a higher coding gain than the pre-trained separable KLT. In fact, the Laplacian constraint that requires the transform to be a GFT can be viewed as a regularization, which prevents overfitting. In contrast, a separable KLT that is learned and tested on different datasets is not regularized, and may suffer from overfitting.

We also note that in this experiment, data-driven transforms (separable KLT and fast GFT) yield a very significant gain ($> 10\%$) compared to the scheme with 2D DCT or hybrid DCT+ADST. While data-driven transforms provide a better adaptation to statistical properties of blocks, the significant gain we obtain here may not be easily achievable in practice. In fact, part of the gain we observe is thanks to the fact that the coding scheme we implement here is an open-loop system, where residual blocks were extracted from HEVC encoder, and then processed using an independent entropy encoder. In addition, during block collection process in the HEVC encoder, the decision of intra prediction mode is based on the RD cost, which is dependent of the transform coefficients. However, the separable KLT and fast GFT were trained after those blocks were collected, and prediction modes may be chosen differently when RD cost was evaluated based on KLT or GFT coefficients. Thus, while data-driven transforms give a significant gain on the blocks we extracted, such a big gain may not be attainable when they are embedded into a real codec as a component of a closed-loop system.

3.5 Conclusion

In this chapter, we have investigated several data-driven fast GFTs for video coding. A general graph learning framework for fast GFT was introduced in Section 3.1, where constrains associated to fast GFT algorithms were incorporated. We showed that if the desired butterfly stage(s), associated to given graph symmetry properties, is specified, then the problem is convex and can be solved using existing algorithms. Two approaches under this framework were considered: in Section 3.2, we learned GFTs corresponding to symmetric line graphs, and introduced a learning approach that approximates KLT basis functions using SLGTs. In Section 3.3, we extended the graph learning problem to non-separable transforms. Without any prior information on graph topology or specified butterfly stage(s), we proposed an approximate solution that gives a GFT with the same complexity as a separable transform. Experimental results were shown in Section 3.4, where the learned SLGT and non-separable fast GFT achieve consistent compression gains for inter and intra predicted residual signals, respectively.

Chapter 4

Lapped Graph Fourier Transform

In image and video coding, block-based transforms such as the DCT, are applied to different blocks independently. In this way, the correlation between pixels on the boundary of two adjacent blocks cannot be captured. This leads to the so-called blocking artifact: an artificial discontinuity across the block boundary in the reconstructed signal. A well-known approach that addresses blocking artifacts is based on the design of lapped transforms [79, 80], where a transform is applied to blocks with overlap. This method has been shown to reduce significantly blocking artifacts. Designs of lapped transforms include the lapped orthogonal transform (LOT) [81], and a pre- and post-filtering framework [127], which has been adopted in the Daala codec [130].

However, as many signal processing techniques have been extended to a graph-based framework, to our knowledge, lapped transforms have not been studied in the context of graph signal processing. Similar to the block-based DCT, most existing graph-based transforms for pixel data are applied block-wise without overlap, and may lead to blocking artifacts. Thus, an LOT-like graph-based transform that can mitigate blocking artifacts would be of practical interest. In this chapter, we extend the notion of lapped transform to graph signals, with particular focus on line graphs with non-uniform weights, as in Figure 3.2(a). We propose the lapped graph Fourier transform (LGFT): a family of lapped transforms that have different basis functions for different blocks so as to capture distinct local statistical properties for different blocks, while having perfect reconstruction and orthogonality properties. We will show that the conventional LOT can be seen as approximately optimizing the transform coding gain for signals with a *uniform line graph model*, while our proposed LGFT is a generalization of the LOT that considers *more general graph-based models*. The LGFT matrix for each block can be obtained from a Kron reduction [23] of the graph, followed by an eigen-decomposition. Experimental results show that for data associated to a non-uniform line graph model, the LGFT can provide a better transform coding gain as compared to existing transforms such as LOT and DCT.

Work in this chapter has been published in [73].

The rest of this chapter is organized as follows. In Section 4.1 we review basic concepts of the LOT. In Section 4.2, we formulate the optimal lapped graph Fourier transform design problem, and propose an LGFT design method. Experimental results are shown in Section 4.3. Finally we conclude this work in Section 4.4.

4.1 Review of Lapped Orthogonal Transforms

The lapped orthogonal transform (LOT) is a lapped transform that has orthogonal basis functions, and is applied to blocks with length $2M$ and overlap length M. Given a block size $2M$, the LOT matrix is a $2M \times M$ matrix $\mathbf{R} = (\mathbf{E}^\top,\ \mathbf{F}^\top)^\top$, where $\mathbf{E}$ and $\mathbf{F}$ are two square matrices to be designed. If we denote $\mathbf{x}$ as the input signal and $\mathbf{y}$ as the transform domain vector, then we can define a transform matrix $\mathbf{T}$ such that $\mathbf{y} = \mathbf{T}^\top\mathbf{x}$, and the reconstructed signal $\hat{\mathbf{x}} = \mathbf{T}\mathbf{y}$, where

$$\mathbf{T} = \begin{pmatrix} \ddots & & & \\ & \mathbf{R} & & \\ & & \mathbf{R} & \\ & & & \ddots \end{pmatrix} = \begin{pmatrix} \ddots & & & \\ \ddots & \mathbf{E} & & \\ & \mathbf{F} & \mathbf{E} & \\ & & \mathbf{F} & \ddots \\ & & & \ddots \end{pmatrix}.$$

The transform $\mathbf{R}$ is a valid LOT if and only if $\mathbf{T}\mathbf{T}^\top = \mathbf{T}^\top\mathbf{T} = \mathbf{I}$, meaning that $\hat{\mathbf{x}} = \mathbf{x}$ and columns of $\mathbf{T}$ are orthogonal. The general solution has the form [118]:

$$\mathbf{E} = \mathbf{P}\mathbf{Q}, \quad \mathbf{F} = (\mathbf{I} - \mathbf{P})\mathbf{Q}, \tag{4.1}$$

where $\mathbf{P}$ can be any symmetric projection matrix, and $\mathbf{Q}$ can be any orthogonal matrix, both with size $M \times M$.[1]

Many lapped transform designs use the DCT as a key component. One example of such a design is:

$$\hat{\mathbf{E}} = \frac{1}{2}\left(\mathbf{U}_e - \mathbf{U}_o,\ (\mathbf{U}_e - \mathbf{U}_o)\mathbf{Z}\right), \quad \hat{\mathbf{F}} = \frac{1}{2}\left(\mathbf{J}(\mathbf{U}_e - \mathbf{U}_o),\ -\mathbf{J}(\mathbf{U}_e - \mathbf{U}_o)\mathbf{Z}\right), \tag{4.2}$$

where $\mathbf{U}_e$ and $\mathbf{U}_o$ are $M \times M/2$ matrices whose columns are the length-M DCT functions of even and odd symmetry, respectively. The matrix $\mathbf{Z}$ is a cascade of plane rotations

[1] In the literature, the general solution is sometimes represented as $\mathbf{E} = \mathbf{Q}\mathbf{P}$, $\mathbf{F} = \mathbf{Q}(\mathbf{I} - \mathbf{P})$. In fact, this form and (4.1) are interchangeable, and we focus on (4.1) here as it can be extended to the graph-based design more easily.

[81] or a product of DST-IV and DCT-II [80], and $\mathbf{J}$ is the order reversal permutation matrix (1.1). Based on the fact that DCT approximates the KLT, the design in (4.2) approximates the optimal solution characterized in [79]. The projection and orthogonal matrices corresponding to (4.2) are

$$\hat{\mathbf{P}} = \frac{1}{2}(\mathbf{I} - \mathbf{U}_e\mathbf{U}_o^\top - \mathbf{U}_o\mathbf{U}_e^\top), \quad \hat{\mathbf{Q}} = (\mathbf{U}_e,\ -\mathbf{U}_o\mathbf{Z}). \tag{4.3}$$

4.2 Lapped Transforms on Graphs

We now propose the lapped graph Fourier transform (LGFT) by first investigating the conditions for perfect reconstruction and orthogonality, and then incorporating the graph variation (1.5) into the optimality criterion.

4.2.1 Conditions of Perfect Reconstruction and Orthogonality

We denote a graph signal $\mathbf{x} \in \mathbb{R}^{NM}$ as $\mathbf{x} = (\mathbf{x}_1^\top, \ldots, \mathbf{x}_N^\top)^\top$, modeled by an attractive GMRF:

$$\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{L}^\dagger = \mathbf{C} = \begin{pmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} & \cdots & \mathbf{C}_{1,N} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} & \cdots & \mathbf{C}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{N,1} & \mathbf{C}_{N,2} & \cdots & \mathbf{C}_{N,N} \end{pmatrix}\right),$$

where $(\mathbf{x}_k^\top, \mathbf{x}_{k+1}^\top)^\top$ corresponds to the k-th block with length $2M$, and $\mathbf{C}_{p,q}$ is the (p,q)-th $M \times M$ block component of the covariance matrix $\mathbf{C}$. Unlike the conventional LOTs, where a common model is used for all blocks, here we consider different models for different blocks, such as a line graph model with non-uniform weights (e.g., Figure 3.2(a)), where $\mathbf{C}_{k,k}$ are different for different k. Under this assumption, we revisit the conditions of perfect reconstruction and orthogonality. First, we define a lapped transform matrix $\mathbf{R}_k = (\mathbf{E}_k^\top, \mathbf{F}_k^\top)^\top$ for the k-th block $(\mathbf{x}_k^\top, \mathbf{x}_{k+1}^\top)^\top$, where the $\mathbf{R}_k$'s are different in order to capture distinct statistical properties for different blocks. Based on this definition, the overall transform matrix $\mathbf{T}_{\text{LGFT}}$ is

$$\mathbf{T}_{\text{LGFT}} = \begin{pmatrix} \ddots & & & & \\ & \mathbf{R}_k & & & \\ & & \mathbf{R}_{k+1} & & \\ & & & \ddots & \end{pmatrix} = \begin{pmatrix} \ddots & & & & \\ & \mathbf{E}_k & & & \\ & \mathbf{F}_k & \mathbf{E}_{k+1} & & \\ & & \mathbf{F}_{k+1} & & \\ & & & \ddots & \end{pmatrix}.$$

Then, we obtain the output signal $\mathbf{y} = (\mathbf{y}_1^\top, \ldots, \mathbf{y}_N)$ and the reconstructed signal $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1^\top, \ldots, \hat{\mathbf{x}}_N^\top)$ with

$$\mathbf{y}_k = \mathbf{E}_k^\top \mathbf{x}_k + \mathbf{F}_k^\top \mathbf{x}_{k+1},$$
$$\hat{\mathbf{x}}_k = \mathbf{F}_{k-1}\mathbf{y}_{k-1} + \mathbf{E}_k \mathbf{y}_k = \left(\mathbf{E}_k\mathbf{E}_k^\top + \mathbf{F}_{k-1}\mathbf{F}_{k-1}^\top\right)\mathbf{x}_k + \mathbf{F}_{k-1}\mathbf{E}_{k-1}^\top \mathbf{x}_{k-1} + \mathbf{E}_k\mathbf{F}_k^\top \mathbf{x}_{k+1}.$$

By comparing $\mathbf{x}_k$ and $\hat{\mathbf{x}}_k$, we obtain the conditions for perfect reconstruction and aliasing cancellation. In addition, the orthogonality constraint, $\mathbf{R}_k^\top \mathbf{R}_k = \mathbf{I}$, is equivalent to $\mathbf{E}_k^\top \mathbf{E}_k + \mathbf{F}_k^\top \mathbf{F}_k = \mathbf{I}$. Thus, for each k, the desired transform should satisfy

$$\mathbf{E}_k\mathbf{E}_k^\top + \mathbf{F}_{k-1}\mathbf{F}_{k-1}^\top = \mathbf{I}, \qquad \text{(Perfect reconstruction)} \tag{4.4}$$
$$\mathbf{E}_k\mathbf{F}_k^\top = \mathbf{F}_k\mathbf{E}_k^\top = \mathbf{0}, \qquad \text{(Aliasing cancellation)} \tag{4.5}$$
$$\mathbf{E}_k^\top\mathbf{E}_k + \mathbf{F}_k^\top\mathbf{F}_k = \mathbf{I}. \qquad \text{(Orthogonality)} \tag{4.6}$$

Due to the highly nonlinear nature of these constraints as well as the large number of degrees of freedom in designing $\mathbf{E}_k$ and $\mathbf{F}_k$, we propose a LGFT construction that generalizes the LOT solution of (4.1). We select

$$\mathbf{E}_k = \mathbf{P}\mathbf{Q}_k, \quad \mathbf{F}_k = (\mathbf{I} - \mathbf{P})\mathbf{Q}_k, \tag{4.7}$$

where the projection matrix $\mathbf{P}$ is common for all k so that (4.4) can be satisfied. Based on (4.7), one can verify that (4.4)-(4.6) are always satisfied as long as $\mathbf{P}$ is a symmetric projection matrix and the $\mathbf{Q}_k$ are orthogonal matrices. Thus, (4.7) is a sufficient condition of perfect reconstruction and orthogonality for LGFT.

4.2.2 Proposed LGFT Construction

An optimality criterion for the conventional LOT based on the transform coding gain can be defined as [53, 80]:

$$G_{TC} = \frac{\frac{1}{M}\sum_{i=1}^{M}\sigma_{k,i}^2}{\left(\prod_{i=1}^{M}\sigma_{k,i}^2\right)^{1/M}}, \tag{4.8}$$

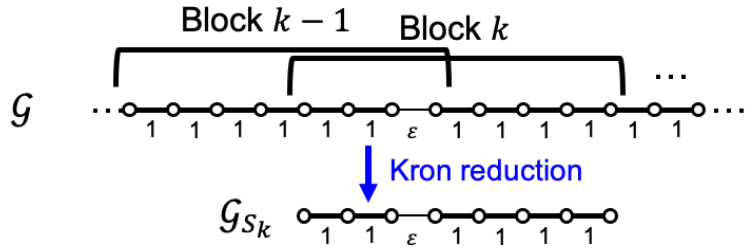


Figure 4.1: An example of Kron reduction on a line graph, where $\mathcal{G}_{S_k}$ is the graph obtained from Kron reduction of $\mathcal{G}$ with vertex subset S_k.

where $\sigma_{k,i}^2 = \mathrm{Var}(\mathbf{y}_k(i))$ is the variance of the i-th transform coefficient in block k. With $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$, $\sigma_{k,i}^2$ is the (i,i) entry of

$$\mathbb{E}[\mathbf{y}_k \mathbf{y}_k^\top] = \begin{pmatrix} \mathbf{E} \\ \mathbf{F} \end{pmatrix}^\top \begin{pmatrix} \mathbf{C}_{k,k} & \mathbf{C}_{k,k+1} \\ \mathbf{C}_{k+1,k} & \mathbf{C}_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \mathbf{E} \\ \mathbf{F} \end{pmatrix} \tag{4.9}$$

$$= \mathbf{Q}^\top \underbrace{\begin{pmatrix} \mathbf{P} \\ \mathbf{I}-\mathbf{P} \end{pmatrix}^\top \begin{pmatrix} \mathbf{C}_{k,k} & \mathbf{C}_{k,k+1} \\ \mathbf{C}_{k+1,k} & \mathbf{C}_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \mathbf{P} \\ \mathbf{I}-\mathbf{P} \end{pmatrix}}_{\mathbf{G}_k} \mathbf{Q}. \tag{4.10}$$

It has been shown [80] that, for a fixed $\mathbf{P}$, the optimal $\mathbf{Q}$ that maximizes G_{TC} is the eigenmatrix of $\mathbf{G}_k$. In the data model considered in conventional LOT, $\mathbf{G}_k = \mathbf{G}$ is common for all k. In (4.1), $\mathbf{Z}$ is typically chosen as an orthogonal transform with fast implementation such that $\hat{\mathbf{Q}}$ approximates the eigenmatrix of $\mathbf{G}$.

Here, we extend the LOT design problem to LGFT design as follows. Let the signal $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{C} = \mathbf{L}^\dagger)$ be an attractive GMRF, where $\mathbf{L}$ is a graph Laplacian corresponding to graph $\mathcal{G}$ and $\mathbf{C}_{k,k}$ can be different for different k. Note that the signal in block k is also an attractive GMRF with length $2M$:

$$\begin{pmatrix} \mathbf{x}_k \\ \mathbf{x}_{k+1} \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \overline{\mathbf{L}}_{S_k}^\dagger\right), \quad \overline{\mathbf{L}}_{S_k} := \begin{pmatrix} \mathbf{C}_{k,k} & \mathbf{C}_{k,k+1} \\ \mathbf{C}_{k+1,k} & \mathbf{C}_{k+1,k+1} \end{pmatrix}^\dagger.$$

The matrix $\overline{\mathbf{L}}_{S_k}$ is a Laplacian obtained by Kron reduction [23] of the graph nodes $\mathcal{V}_{S_k} = \{(k-1)M+1, \ldots, (k+1)M\}$ corresponding to block k. In general, the Kron reduction of a subset of graph nodes $\mathcal{V}_S \in \mathcal{V}$ can be expressed as

$$\overline{\mathbf{L}}_S = \mathbf{L}_{S,S'} \mathbf{L}_{S',S'}^{-1} \mathbf{L}_{S',S}, \quad \mathcal{V}_{S'} = \mathcal{V} \backslash \mathcal{V}_S.$$

As in (4.9), we consider (4.7) and replace the block components of $\mathbf{C}$ by $\overline{\mathbf{L}}_{S_k}$. With the assumption that $\overline{\mathbf{L}}_{S_k}$ can be different for different k, we would like to choose $\mathbf{Q}_k$ that diagonalizes

$$\mathbf{H}_k := \begin{pmatrix} \mathbf{P} \\ \mathbf{I} - \mathbf{P} \end{pmatrix}^\top \overline{\mathbf{L}}_{S_k} \begin{pmatrix} \mathbf{P} \\ \mathbf{I} - \mathbf{P} \end{pmatrix}. \tag{4.11}$$

In particular, if $\mathbf{L}$ corresponds to a line graph, then the Kron reduction $\overline{\mathbf{L}}_{S_k}$ is the Laplacian of the line graph segment of $\mathcal{G}$ with nodes $\mathcal{V}_{S_k}$. Thus, the retrieval of the k-th LGFT component $\mathbf{R}_k$ boils down to choosing the line graph segment with length $2M$ associated to block k, which can be different for different k. An illustrative example is shown in Figure 4.1.

While the choice of $\mathbf{Q}_k$ is straightforward with a given $\mathbf{P}$, finding $\mathbf{P}$ for globally optimal solution in term of transform coding gain, even in conventional LOT design, is a challenging problem [81]. Following the LOT design, we adopt the matrix $\hat{\mathbf{P}}$ in (4.3) associated to (4.2). With this choice of $\mathbf{P}$, this LGFT design can be regarded as a generalization of the LOT, where $\mathbf{R}_k$ reduces to LOT when $\mathbf{Q}_k = (\mathbf{U}_e, \ -\mathbf{U}_o\mathbf{Z})$.

To summarize, given a line graph $\mathcal{G}$ with Laplacian $\mathbf{L}$ and block size M, the proposed LGFT can be constructed as follows:

1. Pick $\mathbf{P}$ as in (4.3).
2. Pick $\mathbf{Q}_k$ for each k as the eigenmatrix of the $\mathbf{H}_k$ in (4.11).
3. Obtain LGFT components as given in (4.7).

Similar to the LOT, which achieves nearly optimal G_{TC} when $\mathbf{L}$ is associated to a uniform line graph, the proposed LGFT construction, based on a fixed $\mathbf{P} = \hat{\mathbf{P}}$, can approximately optimize G_{TC} when the line graph has non-uniform weights.

4.3 Experimental Results for LGFT

We evaluate the proposed LGFT on a class of line graphs, where some edges have weights ε that characterize weak correlations, and other edges have weights 1. This line graph model has been used for encoding intra-predicted images [49] and piecewise smooth images [50, 144].

4.3.1 Transform Coding Gain with a Particular Line Graph

First, we consider a line graph with length $n = 400$ and $\varepsilon = 0.05$, where weak weights are located at edges $(h, h+1)$ with $h \in \{15, 30, 45, 60, \ldots, 390\}$. A covariance matrix

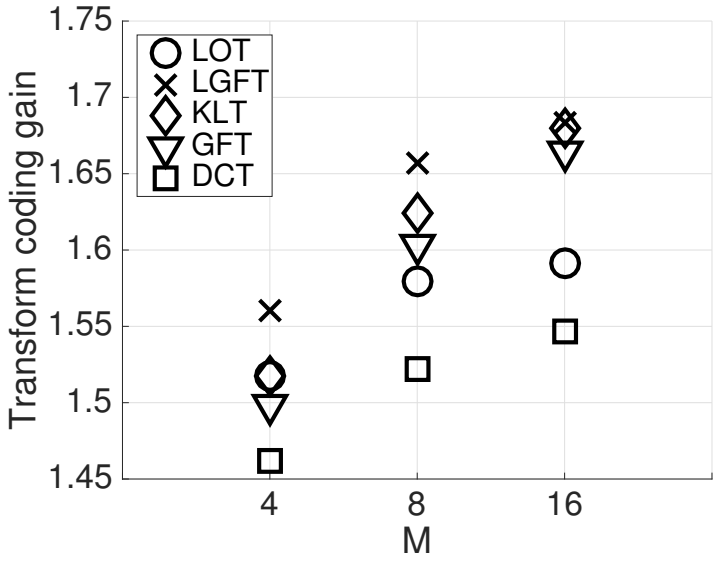


Figure 4.2: Transform coding gains for different transforms and block sizes M with signals modeled by a particular line graph. The KLT, GFT, and DCT shown here are defined in a block-based manner.

$\mathbf{C} = (\mathbf{L} + 0.2\mathbf{I})^{-1}$ is used in this experiment to avoid the matrix singularity issue. We compare the transform coding gains with LOT, LGFT, DCT, KLT, and GFT, where the three latter transforms are designed and applied in a non-overlapping block-based manner. The implementation of LOT follows from (4.2), where $\mathbf{Z}$ is composed of a DCT-II and a DST-IV, as suggested in [79]. For each transform, block sizes of $M = 4$, 8, and 16 are considered.

In Figure 4.2 we show the transform size versus the transform coding gain G_{TC}. For this model $\mathcal{N}(\mathbf{0}, \mathbf{C})$, G_{TC} is upper-bounded by 1.736, derived from the length-n KLT of the overall model. We can see that the LGFT yields the highest gain for all block sizes included in this experiment. For some block sizes larger than $M = 16$, the transform coding gains with the block-based KLT and GFT are higher than those of the LGFT since there are fewer block boundaries. In practical coding scenarios, additional information such as the positions of weak edges may be required as bit rate overhead. This has not been considered in the analysis here.

We show the LGFT basis functions $\mathbf{R}_2$ with $M = 8$ in Figure 4.3. Note that $\mathbf{R}_2$ is designed for the block $(\mathbf{x}_2^\top, \mathbf{x}_3^\top)^\top$ with nodes 9 to 24, where a weak edge $(15, 16)$ lies between the 7th and 8th nodes within this block. We can observe in most basis functions a discontinuity between entries 7 and 8 that corresponds to the lower correlation, showing

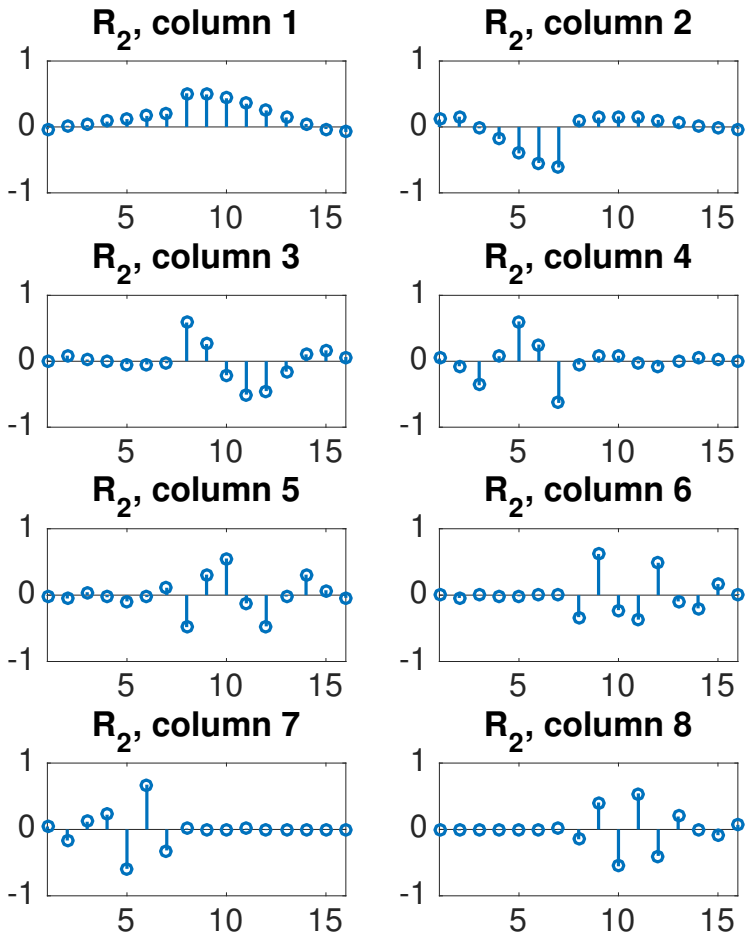
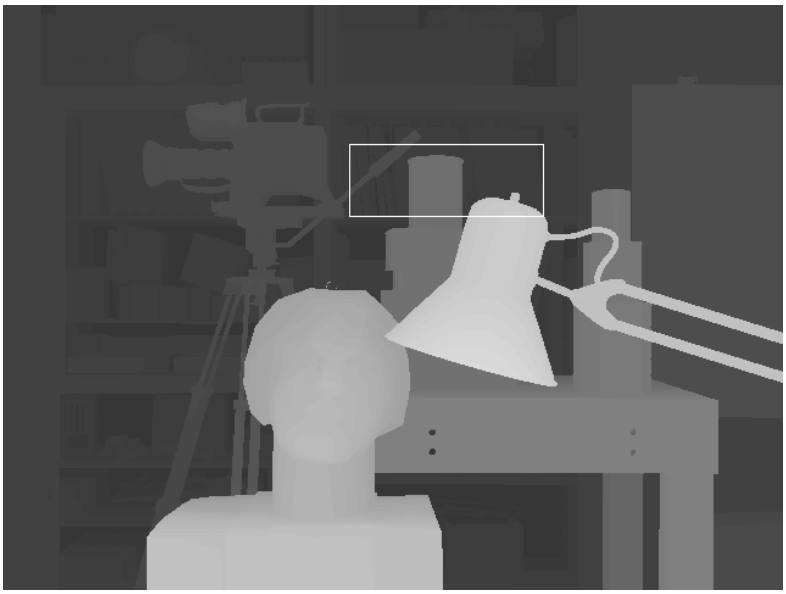


Figure 4.3: Basis functions of the LGFT for $k = 2$ with $M = 8$.

that the LGFT basis can capture the local weak correlation in the graph topology through different choices of $\mathbf{Q}_k$.

4.3.2 LGFT for Image Coding

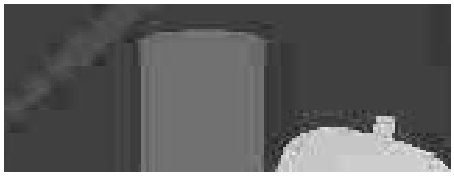
In the second experiment, we apply an LGFT to image coding. A 480×640 piecewise smooth image from the `Tsukuba` dataset [97] is used for this experiment. For demonstration and comparison purpose, we apply different transforms (LGFT, LOT, DCT, and GFT) horizontally, then a common transform (block-based DCT) vertically. We quantize the coefficients and apply inverse horizontal transforms and vertical block-based DCT to



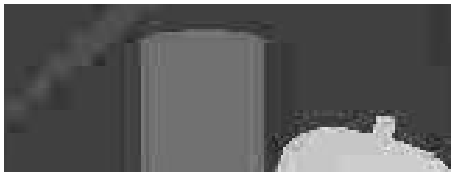
(a) Full Image

(b) Original

(c) Edge map



(d) LGFT (PSNR=42.27dB)



(e) LOT (PSNR=42.14dB)

(f) GFT (PSNR=42.57dB)

(g) DCT (PSNR=42.33dB)

Figure 4.4: Subjective comparison different transforms with QP=30. (a) The full piecewise smooth image and a 160×60 patch. (b) Original image patch. (c) Sobel edge map. (d)-(f) Recovered image patches with different transforms.

reconstruct the signal. To define the line graphs for LGFT and GFT, we apply a Sobel edge detector [40] to obtain an edge map. For each image row we define a line graph based on image discontinuity information in the Sobel edge map: an edge weight of the

QP	PSNR (dB)			
	LGFT	LOT	GFT	DCT
25	48.24	48.12	48.63	48.38
30	44.84	44.74	45.11	44.93
35	41.94	41.83	42.11	41.91
40	37.74	37.73	37.69	37.62
45	33.15	33.15	33.09	33.06

Table 4.1: Quality comparison of different horizontal transforms on full test image.

line graph is $\varepsilon = 0.7$ if any of the two corresponding pixels is an edge point in the Sobel map. We use a block size $M = 8$, and quantization parameters (QP) ranging from 25 to 45, corresponding to quantization factors $2^{(\mathrm{QP}-4)/6}$. Note that, in a practical image coding scheme, the edge location can be transmitted as side information to the decoder for unique decodability. In this experiment, we only compare the distortion but not the number of bits required for encoding. This means that we can fairly compare LGFT with GFT, but not with DCT and LOT, which do not require side information.

Table 4.1 shows the peak-signal-to-noise ratio (PSNR) of different transforms with different QPs. We note that, similar to the GFT, which provides a higher PSNR than the DCT, the LGFT gives a higher PSNR than LOT for almost all QPs. This gain results from the fact that the line graph model, which LGFT and GFT are based on, can better capture the discontinuities in the image signal. In Figure 4.4 we show the original image, the Sobel edge map, and the reconstructed images using different transforms with QP=30. With this quantization level, while LGFT does not give higher PSNRs than GFT, it yields reduced blocking artifacts (reduced vertical image discontinuities) as compared to the GFT.

4.4 Conclusion

In this chapter, we have extended the well-known lapped transform to a graph-based setting. We derived the conditions for perfect reconstruction and orthogonality of the lapped graph Fourier transform (LGFT), which is more general than the lapped orthogonal transform (LOT). Then, we proposed a design of LGFT on an arbitrary line graph, where different transform functions are applied to different blocks to adapt to different local statistical properties. Experimental results showed that on a nonuniform line graph, the LGFT can achieve a better energy compaction than the block-based graph Fourier transform and a conventional LOT in terms of transform coding gain. The extensions to different lengths of overlap and to graphs with more general topologies, as well as fast implementations of the LGFT, will be explored in future work.

Chapter 5

Efficient DCT and DST Filtering with Sparse Graph Operators

Filtering, where frequency components of a signal are attenuated or amplified, is a fundamental operation in signal processing. Similar to conventional filters in digital signal processing, which manipulate signals in Fourier domain, a graph filter selectively reduces or amplifies graph Fourier coefficients, and can be characterized by a frequency response that indicates how much the filter amplifies each graph frequency component. This notion of frequency selection leads to various applications, including graph signal denoising [7, 89, 138], classification [78] and clustering [128], and graph convolutional neural networks [19, 55].

For an undirected graph, we recall (1.7) in Section 1.2.2, which represents the frequency domain graph filter operation:

$$\mathbf{H} = \mathbf{\Phi} \cdot h(\mathbf{\Lambda}) \cdot \mathbf{\Phi}^\top, \quad h(\mathbf{\Lambda}) := \mathrm{diag}(h(\lambda_1), \cdots, h(\lambda_N)). \tag{1.7}$$

The operation $\mathbf{y} = \mathbf{H}\mathbf{x}$ with input signal $\mathbf{x}$ and filter matrix $\mathbf{H}$ involves a forward graph Fourier transform (GFT) $\mathbf{\Phi}^\top$, a frequency selective scaling operation $h(\mathbf{\Lambda})$, and an inverse GFT $\mathbf{\Phi}$. However, as fast GFT algorithms may not exist for any arbitrary graphs (see Chapter 2), GFT often introduces a very high computational overhead when the graph is large. To address this issue, graph filters can usually be implemented with polynomial operations in vertex domain as (recall again in Section 1.2.2)

$$\mathbf{H} = \sum_{k=0}^{K} g_k \mathbf{Z}^k, \quad \text{with } \mathbf{Z}^0 = \mathbf{I}, \tag{1.6}$$

where the g_k's are coefficients and $\mathbf{Z}$ is called the *fundamental graph operator*, or *graph operator* for short. With this expression, graph filtering can be applied in the vertex (sample) domain via $\mathbf{y} = \mathbf{H}\mathbf{x}$, which does not require GFT computations. A graph filter in the form of (1.6) is usually called an *FIR graph filter* [16, 51] as it can be viewed as

This chapter is an extended version of the work in [74, 75].

an analogy to conventional FIR filters with order K, which are polynomials of the delay z. In what follows, we call the filters defined as in (1.6) *polynomial graph filters (PGFs)*.

Various methods for designing vertex domain graph filters given a desired frequency response have been studied in the literature. Least squares design of polynomial filters given a target frequency response was introduced in [105]. The recurrence relations of Chebyshev polynomials provide computational benefits in distributed filter implementations as shown in [44, 115]. In [109] an extension of graph filter operations to a node-variant setting is proposed, and polynomial approximation approaches using convex optimization are introduced. Autoregressive moving average (ARMA) graph filters, whose frequency responses are characterized by rational polynomials, have been investigated in [51, 68] in both static and time-varying settings. Design strategies of ARMA filters are further studied in [67], which provides comparisons to PGFs. Furthermore, in [16], state-of-the-art filtering methods have been extended to an edge-variant setting.

Recently, it has been pointed out that *multiple operators* can be obtained for cycle graphs [34] and line graphs [74]. For those graphs, multiple graph operators $\mathcal{Z} = \{\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(m)}\}$ that are jointly diagonalizable (i.e., have a common eigenbasis) can be obtained. Essentially, those operators are by themselves graph filter matrices with different frequency responses. Thus, unlike (1.6), which is a polynomial of a single operator, we can design graph filters as follows:

$$\mathbf{H}_{\mathcal{Z},K} = p_K(\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(m)}), \tag{5.1}$$

where $p_K(\cdot)$ stands for a multivariate polynomial with degree K and arbitrary coefficients. Iterative algorithms for the implementation of such graphs filters have been further studied in [27]. Since $\mathbf{H}_{\{\mathbf{Z}\},K} = p_K(\mathbf{Z})$ reduces to (1.6), the form (5.1) is a generalization of the PGF expression. We refer to (5.1) as *multivariate polynomial graph filter (MPGF)*.

In this chapter, we focus on filter operations based on the well-known DCTs and DSTs. Notably, the DCT is the GFT associated to a uniform line graph, which means that DCT filters are essentially graph filters. A DCT filter [8] is defined as the following operation: 1) applying the DCT, 2) scaling DCT coefficients selectively, and 3) performing the inverse DCT. While implementation of DCT filter is typically done using forward and inverse DCT, in fact, we can apply graph filter approaches to design and implement DCT filters. More generally, it has been shown in [99] that all members of the family of discrete trigonometric transforms (DTTs), i.e., 8 types of DCTs and 8 types of discrete sine transforms (DSTs), are associated with line graphs. This allows us to exploit

graph filter approaches to all DTT filters, whose applications include image resizing [91], biomedical signal processing [113], medical imaging [129], and video coding [143].

Our work approaches the design of efficient sample domain (graph vertex domain) graph filters, with particular focus on DTT filters. In particular, we show that if the GFT is one among the 16 DTTs, then, in addition to the graph operator obtained from the well-known line graph model [99], we can derive a family $\mathcal{Z}$ of sparse graph operators with closed form expressions. In this way, efficient DTT filters can be obtained using PGF and MPGF design approaches, yielding a lower complexity than a DTT filter implementation in the transform domain. Experimental results in video coding are presented to demonstrate the effectiveness of the proposed DCT and DST filters.

In the remainder of this chapter, we summarize our contributions with respect to previous work in Section 5.1. In Section 5.2, we consider DTTs, where sparse operators can be obtained by extending well-known properties of DTTs. We also extend the results to 2D DTTs and provide some remarks on sparse operators for general graphs. Section 5.3 introduces PGF and MPGF design approaches using least squares and minimax criteria. An efficient filter design for Laplacian quadratic form approximation is also presented. Experimental results are shown in Section 5.4 to demonstrate the effectiveness of our methods in graph filter design as well as applications in video coding. The conclusion of the chapter will be drawn in Section 5.5.

5.1 Summary of Contributions

The first contribution of this work is to introduce novel DTT filter design methods for graph vertex domain implementation. We note that DTT filters are special cases of graph filters, which can be implemented using PGFs and MPGFs. The DTT case has not been explored in existing work on general graph filters [16, 51, 67, 68, 109, 115]. We also introduce multiple sparse graph operators specific to DTTs and allowing fast MPGF implementations. In addition, while in related work [12, 82, 99], DTT filtering is typically performed in the transform domain using convolution-multiplication properties, we introduce sample domain DTT filter implementations based on PGF and MPGF designs, and show that our designs with low degree polynomials lead to faster implementations as compared to those designs that require forward and inverse DTTs, especially in cases where DTT size is large. Furthermore, in addition to the well-known least squares graph filter design, we propose a novel minimax design approach for both PGFs and MPGFs.

Second, we provide novel insights on multiple *sparse* graph operators. Motivated by the previous work [27, 34], we use multiple graph operators for filter design, but focus particularly on operators that are sparse. Notably, we show that for each of the 16 DTTs,

whose associated graph is a uniform line graph or its variants, there exist a series of operators that are sparse and have closed form expressions. We demonstrate that those operators lead to more efficient implementations, as compared to conventional PGF designs, for DTT filters with frequency responses that are non-smooth (e.g., ideal low-pass filter) or non-monotonic (e.g., bandpass filter). Note that [27] studies MPGFs with a focus on distributed filter implementations, but does not investigate design approaches for those filters or how sparse operators for generic graphs can be obtained other than cycle and Cartesian product graphs. Our work complements the study in [27] by considering 1) the case where GFT is a DTT, which corresponds to various line graphs, and 2) design approaches for MPGFs. To the best of our knowledge, sparse DTT operators other than those derived from line graphs are not known in the literature.

Third, we demonstrate experimentally the benefits of sparse DTT operators in image and video compression applications. In addition to filter operation, our approach can also be used to evaluate the transform domain weighted energy given by the Laplacian quadratic form, which has been used for rate-distortion optimization in the context of image and video coding [33,50]. We implement the proposed method in AV1, a real-world codec, where our method provides a speedup in the transform type search procedure.

5.2 Sparse DCT and DST Operators

Classical PGFs can be extended to MPGFs [27] if multiple graph operators are available [34]. Let $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\top$ be a Laplacian with GFT $\mathbf{\Phi}$, we assume that we have a series of graph operators $\mathcal{Z} = \{\mathbf{Z}^{(k)}\}_{k=1}^{M}$ that share the same eigenvectors as $\mathbf{L}$, but with different eigenvalues:

$$\mathbf{Z}^{(k)} = \mathbf{\Phi}\mathbf{\Lambda}^{(k)}\mathbf{\Phi}^\top, \quad \mathbf{\Lambda}^{(k)} = \mathrm{diag}(\boldsymbol{\lambda}^{(k)}) = \mathrm{diag}(\lambda_1^{(k)}, \ldots, \lambda_N^{(k)}),$$

where $\boldsymbol{\lambda}^{(k)} = (\lambda_1^{(k)}, \ldots, \lambda_N^{(k)})^\top$ denotes the vector of eigenvalues of $\mathbf{Z}^{(k)}$. When the polynomial degree $K = 1$ in (5.1), we have:

$$\mathbf{H}_{\mathcal{Z},1} = g_0\mathbf{I} + \sum_{m=1}^{M} g_m \mathbf{Z}^{(m)}, \tag{5.2}$$

where g_k are coefficients. When $K = 2$, we have

$$\begin{aligned}\mathbf{H}_{\mathcal{Z},2} = g_0\mathbf{I} + \sum_{m=1}^{M} g_m\mathbf{Z}^{(m)} \\ + g_{M+1}\mathbf{Z}^{(1)}\mathbf{Z}^{(1)} + g_{M+2}\mathbf{Z}^{(1)}\mathbf{Z}^{(2)} + \cdots + g_{2M}\mathbf{Z}^{(1)}\mathbf{Z}^{(M)} \\ + g_{2M+1}\mathbf{Z}^{(2)}\mathbf{Z}^{(2)} + \cdots + g_{3M-1}\mathbf{Z}^{(2)}\mathbf{Z}^{(M)} \\ + \dots \\ + g_{(M^2+3M)/2}\mathbf{Z}^{(M)}\mathbf{Z}^{(M)},\end{aligned} \tag{5.3}$$

where the terms $\mathbf{Z}^{(j)}\mathbf{Z}^{(i)}$ with $j > i$ are not required because all operators commute, i.e., $\mathbf{Z}^{(i)}\mathbf{Z}^{(j)} = \mathbf{Z}^{(j)}\mathbf{Z}^{(i)}$. Expressions with a higher degree can be obtained with polynomial kernel expansion [47]. We also note that, since $\mathbf{H}_{\{\mathbf{Z}\},K}$ reduces to the form of $\mathbf{H}$ in (1.6), $\mathbf{H}_{\mathcal{Z},K}$ is a generalization of PGF and thus provides more degrees of freedom for the filter design procedure.

As pointed out in the introduction, DTT filters are essentially graph filters. This means that they can be implemented as PGFs (1.6) without applying any forward or inverse DTT. Next, we will go one step further by introducing multiple sparse operators for each DTT, which allows the implementation of DTT filters using MPGF.

5.2.1 Sparse DCT-II Operators

Recall that $\mathbf{u}_j$ denotes the DCT functions as in (1.9), and $\mathbf{L}_{\mathrm{D}}$ represents the Laplacian of a uniform line graph as in (1.10). Here, we revisit a validation in [117] to verify that $\mathbf{u}_j$ is an eigenvector of $\mathbf{L}_{\mathrm{D}}$ with eigenvalue $\omega_j = 2 - 2\cos((j-1)\pi/N)$ for each $j = 1, \dots N$. Then, we extend it to a generalized version, which contains not only $\mathbf{L}_{\mathrm{D}}$ but a family of sparse graph filters that all have $\mathbf{u}_j$ as an eigenvector.

To verify that $\mathbf{L}_{\mathrm{D}} \cdot \mathbf{u}_j = \omega_j \mathbf{u}_j$, it suffices to consider an equivalent equation: $\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j = (2 - \omega_j)\mathbf{u}_j$, where

$$\mathbf{Z}_{\text{DCT-II}} = 2\mathbf{I} - \mathbf{L}_{\mathrm{D}} = \begin{pmatrix} 1 & 1 & & & \\ 1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & 1 \\ & & & 1 & 1 \end{pmatrix}. \tag{5.4}$$

Table 5.1: Matrix structure and corresponding eigenpairs of sparse operators associated to all DCTs and DSTs. The indice j ranges from 1 to N.

GFT	Structure of $\mathbf{Z}^{(\ell)}$	(Eigenvalue, Eigenvector) of $\mathbf{Z}^{(\ell)}$
DCT-I	Figure 5.2(a), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1)\pi}{N-1}\right),\ \boldsymbol{\phi}_j\right)$
DCT-II	Figure 5.2(b), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell(j-1)\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DCT-III	Figure 5.2(c), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DCT-IV	Figure 5.2(d), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DCT-V	Figure 5.2(e), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1)\pi}{N-1/2}\right),\ \boldsymbol{\phi}_j\right)$
DCT-VI	Figure 5.2(f), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1)\pi}{N-1/2}\right),\ \boldsymbol{\phi}_j\right)$
DCT-VII	Figure 5.2(g), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N-1/2}\right),\ \boldsymbol{\phi}_j\right)$
DCT-VIII	Figure 5.2(h), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N+1/2}\right),\ \boldsymbol{\phi}_j\right)$
DST-I	Figure 5.3(a), $\ell = 1, \ldots, N+1$	$\left(2\cos\left(\frac{\ell j\pi}{N+1}\right),\ \boldsymbol{\phi}_j\right)$
DST-II	Figure 5.3(b), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell j\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DST-III	Figure 5.3(c), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DST-IV	Figure 5.3(d), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N}\right),\ \boldsymbol{\phi}_j\right)$
DST-V	Figure 5.3(e), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell j\pi}{N+1/2}\right),\ \boldsymbol{\phi}_j\right)$
DST-VI	Figure 5.3(f), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell j\pi}{N+1/2}\right),\ \boldsymbol{\phi}_j\right)$
DST-VII	Figure 5.3(g), $\ell = 1, \ldots, N$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N+1/2}\right),\ \boldsymbol{\phi}_j\right)$
DST-VIII	Figure 5.3(h), $\ell = 1, \ldots, N-1$	$\left(2\cos\left(\frac{\ell(j-1/2)\pi}{N-1/2}\right),\ \boldsymbol{\phi}_j\right)$

For $1 \leq p \leq N$, the p-th element of $\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j$ is

$$(\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j)_p = \begin{cases} u_j(1) + u_j(2), & p = 1 \\ u_j(p-1) + u_j(p+1), & 2 \leq p \leq N-1 \\ u_j(N-1) + u_j(N), & p = N. \end{cases}$$

Following the expression in (1.9), we extend the definition of $u_j(k)$ to an arbitrary integer k. The even symmetry of the cosine function at 0 and π gives $u_j(0) = u_j(1)$ and $u_j(N) = u_j(N+1)$, and thus

$$\begin{aligned}(\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j)_1 &= u_j(0) + u_j(2),\\ (\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j)_N &= u_j(N-1) + u_j(N+1).\end{aligned} \tag{5.5}$$

This means that for all $p = 1, \ldots, N$,

$$(\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j)_p = u_j(p-1) + u_j(p+1) \tag{5.6a}$$

$$= \sqrt{\frac{2}{N}} c_j \left[\cos \frac{(j-1)(p-\frac{3}{2})\pi}{N} + \cos \frac{(j-1)(p+\frac{1}{2})\pi}{N}\right] \tag{5.6b}$$

$$= 2\sqrt{\frac{2}{N}} c_j \cos \frac{(j-1)(p-\frac{1}{2})\pi}{N} \cos \frac{(j-1)\pi}{N} \tag{5.6c}$$

$$= (2 - \omega_j) u_j(p), \tag{5.6d}$$

which verifies $\mathbf{Z}_{\text{DCT-II}} \cdot \mathbf{u}_j = (2 - \omega_j)\mathbf{u}_j$.

Note that in (5.6b), we have applied the sum-to-product trigonometric identity:

$$\cos\alpha + \cos\beta = 2\cos\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right). \tag{5.7}$$

Indeed, when $u_j(q \pm 1)$ is replaced by $u_j(q \pm \ell)$ in (5.6a), this identity also applies, which generalizes (5.6a)-(5.6d) to

$$\begin{aligned}u_j(p-\ell) + u_j(p+\ell) &= \sqrt{\frac{2}{N}} c_j \left[\cos \frac{(j-1)(p-\ell-\frac{1}{2})\pi}{N} + \cos \frac{(j-1)(p+\ell-\frac{1}{2})\pi}{N}\right]\\ &= 2\sqrt{\frac{2}{N}} c_j \cos \frac{(j-1)(p-\frac{1}{2})\pi}{N} \cos \frac{\ell(j-1)\pi}{N}\\ &= \left(2\cos \frac{\ell(j-1)\pi}{N}\right) u_j(p).\end{aligned} \tag{5.8}$$

As in (5.5), we can apply even symmetry of the cosine function at 0 and π, to replace indices $p - \ell$ or $p + \ell$ that are out of the range $[1, N]$ by those within the range:

$$u_j(p-\ell) = u_j(-p+\ell+1), \quad u_j(p+\ell) = u_j(-p-\ell+2N+1).$$

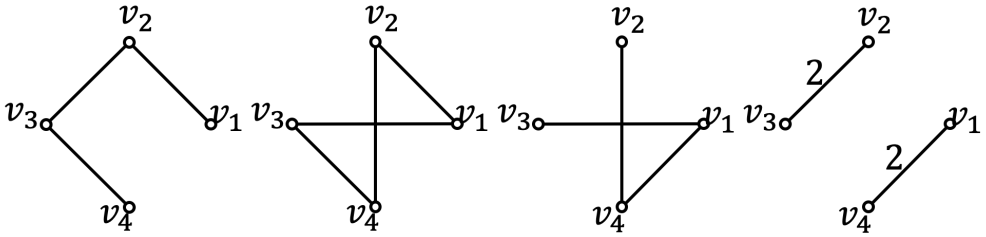
Then, an $N \times N$ matrix $\mathbf{Z}^{(\ell)}_{\text{DCT-II}}$ can be defined such that the left hand side of (5.8) corresponds to $(\mathbf{Z}^{(\ell)}_{\text{DCT-II}} \cdot \mathbf{u}_j)_p$.

$$\begin{pmatrix}1&1&0&0\\1&0&1&0\\0&1&0&1\\0&0&1&1\end{pmatrix}\quad\begin{pmatrix}0&1&1&0\\1&0&0&1\\1&0&0&1\\0&1&1&0\end{pmatrix}\quad\begin{pmatrix}0&0&1&1\\0&1&0&1\\1&0&1&0\\1&1&0&0\end{pmatrix}\quad\begin{pmatrix}0&0&0&2\\0&0&2&0\\0&2&0&0\\2&0&0&0\end{pmatrix}$$

(a) $\mathbf{Z}^{(1)}_{\text{DCT-II}}$, $\mathbf{Z}^{(2)}_{\text{DCT-II}}$, $\mathbf{Z}^{(3)}_{\text{DCT-II}}$, and $\mathbf{Z}^{(4)}_{\text{DCT-II}}$

$$\begin{pmatrix}1&-1&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&-1&1\end{pmatrix}\quad\begin{pmatrix}2&-1&-1&0\\-1&2&0&-1\\-1&0&2&-1\\0&-1&-1&2\end{pmatrix}\quad\begin{pmatrix}2&0&-1&-1\\0&1&0&-1\\-1&0&1&0\\-1&-1&0&2\end{pmatrix}\quad\begin{pmatrix}2&0&0&-2\\0&2&-2&0\\0&-2&2&0\\-2&0&0&2\end{pmatrix}$$

(b) $\mathbf{L}^{(1)}_{\text{DCT-II}}$, $\mathbf{L}^{(2)}_{\text{DCT-II}}$, $\mathbf{L}^{(3)}_{\text{DCT-II}}$, and $\mathbf{L}^{(4)}_{\text{DCT-II}}$



(c) $\mathcal{G}^{(1)}_{\text{DCT-II}}$, $\mathcal{G}^{(2)}_{\text{DCT-II}}$, $\mathcal{G}^{(3)}_{\text{DCT-II}}$, and $\mathcal{G}^{(4)}_{\text{DCT-II}}$

Figure 5.1: (a) Sparse operators $\mathbf{Z}^{(j)}_{\text{DCT-II}}$, (b) their associated Laplacian matrices $\mathbf{L}^{(j)}_{\text{DCT-II}} = 2\mathbf{I} - \mathbf{Z}^{(j)}$, and (c) associated graphs $\mathcal{G}^{(j)}_{\text{DCT-II}}$ for the length-4 DCT-II.

Proposition 1. *For $\ell = 1, \ldots, N-1$, we define $\mathbf{Z}^{(\ell)}_{DCT\text{-}II}$ as a $N \times N$ matrix, whose p-th row has only two non-zero elements specified as follows:*

$$\left(\mathbf{Z}^{(\ell)}_{DCT\text{-}II}\right)_{p,q_1} = 1, \quad q_1 = \begin{cases} p-\ell, & \text{if } p-\ell \geq 1 \\ -p+\ell+1, & \text{otherwise} \end{cases}$$

$$\left(\mathbf{Z}^{(\ell)}_{DCT\text{-}II}\right)_{p,q_2} = 1, \quad q_2 = \begin{cases} p+\ell, & \text{if } p+\ell \leq N \\ -p-\ell+2N+1, & \text{otherwise} \end{cases}$$

This matrix $\mathbf{Z}^{(\ell)}_{DCT\text{-}II}$ has eigenvectors $\mathbf{u}_j$ with associated eigenvalues $2\cos(\ell(j-1)\pi/N)$ for $j = 1, \ldots, N$.

Note that $\mathbf{Z}^{(1)}_{\text{DCT-II}} = \mathbf{Z}_{\text{DCT-II}}$ as in (5.4). Taking $\ell = 2$ and $\ell = 3$ and following Proposition 1, we see that nonzero elements in $\mathbf{Z}^{(2)}_{\text{DCT-II}}$ and $\mathbf{Z}^{(3)}_{\text{DCT-II}}$ form rectangular-like patterns similar to that in $\mathbf{Z}_{\text{DCT-II}}$:

$$\mathbf{Z}^{(2)}_{\text{DCT-II}} = \begin{pmatrix} & 1 & 1 & & \\ 1 & & & 1 & \\ & & & & \ddots \\ 1 & & & & \\ & 1 & & & 1 \\ & & \ddots & & \\ & & & & 1 \\ & & & 1 & 1 \end{pmatrix}, \quad \mathbf{Z}^{(3)}_{\text{DCT-II}} = \begin{pmatrix} & & 1 & 1 & & \\ & 1 & & & \ddots & \\ 1 & & & & & 1 \\ 1 & & & & & 1 \\ & \ddots & & & 1 & \\ & & 1 & 1 & & \end{pmatrix} \tag{5.9}$$

For $\ell = N$, the derivations in (5.8) are also valid, but with $\mathbf{Z}^{(N)}_{\text{DCT-II}} = 2\mathbf{J}$. The rectangular patterns we observe in (5.9) can be simply extended to any arbitrary transform length N (e.g., all such operators with $N = 6$ are shown in Fig. 5.2(b)). We also show the associated eigenvalues of $\mathbf{Z}^{(\ell)}_{\text{DCT-II}}$ with arbitrary N in Table 5.1.

Example–Length 4 DCT-II Operators

We show in Figure 5.1(a) all sparse operators $\mathbf{Z}^{(\ell)}_{\text{DCT-II}}$ of DCT-II for $N = 4$. In fact, those matrices can be regarded as standard operators on different graphs: by defining $\mathbf{L}^{(\ell)}_{\text{DCT-II}} = 2\mathbf{I} - \mathbf{Z}^{(\ell)}_{\text{DCT-II}}$, we can view $Lm^{(\ell)}_{\text{DCT-II}}$ as a Laplacian matrix of a different graph $\mathcal{G}^{(\ell)}_{\text{DCT-II}}$. For example, all the resulting $\mathbf{L}^{(\ell)}_{\text{DCT-II}}$'s and $\mathcal{G}^{(\ell)}_{\text{DCT-II}}$'s for a length-4 DCT-II are shown in Figure 5.1(b) and (c), respectively. The rectangular patterns we observe in (5.9) can be simply extended to any arbitrary transform length N. We also show the associated eigenvalues of $\mathbf{Z}^{(\ell)}_{\text{DCT-II}}$ with arbitrary N in Table 5.1.

We observe that, among all graphs in Figure 5.1(c), $\mathcal{G}^{(4)}_{\text{DCT-II}}$ is a disconnected graph with two connected components. It is associated to the operator

$$\mathbf{Z}^{(4)}_{\text{DCT-II}} = \mathbf{\Phi}_{\text{DCT-II}} \cdot \text{diag}(2, -2, 2, -2) \cdot \mathbf{\Phi}^{\top}_{\text{DCT-II}}.$$

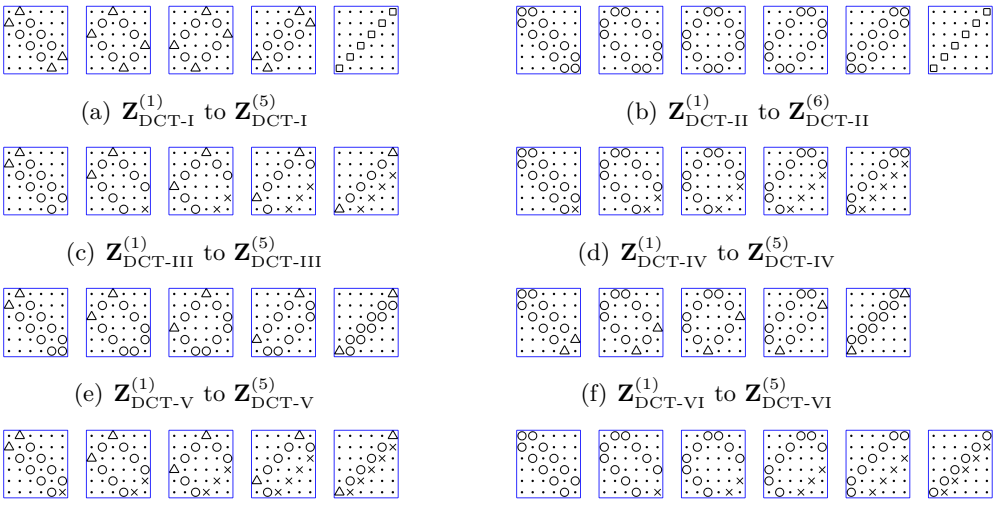
Note that, while $\mathbf{Z}^{(4)}_{\text{DCT-II}}$ is associated to a disconnected graph, it can still be used as a graph operator for DCT-II filter because it is diagonalized by $\mathbf{\Phi}_{\text{DCT-II}}$. However, $\mathbf{Z}^{(4)}_{\text{DCT-II}}$, as well as its polynomials, have repeated eigenvalues, i.e., eigenvalues with multiplicity 2. This means that a filter whose frequency response has distinct values (e.g. low-pass filter with $h(\lambda_1) > \cdots > h(\lambda_4)$ cannot be realized as a PGF of $\mathbf{Z}^{(4)}_{\text{DCT-II}}$.

Based on the previous observation, we can see that those operators associated to disconnected graphs, and those having eigenvalues with high multiplicities would provide less degrees of freedoms in PGF and MPGF filter designs, as compared to an operators with distinct eigenvalues such as $\mathbf{Z}^{(1)}_{\text{DCT-II}}$.

5.2.2 Sparse Operators of 16 DTTs

In fact, the approach in Section 5.2.1 can be adapted to all 16 DTTs to obtain their sparse operators. For illustrative purpose, we present in Section A.2 the derivations for DST-VI, DST-VII, and DCT-V, which share the same right boundary condition with DCT-II, but have different left boundary condition (see Table 1.1). Results for those DTTs with other combinations of left/right boundary condition can be easily extended.

In Table 5.1, we list the sparse operators and their associated eigenpairs for all DTTs. Figures 5.2 and 5.3 show the operators for $N = 6$, which can be easily extended to any



(a) $\mathbf{Z}^{(1)}_{\text{DCT-I}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-I}}$ (b) $\mathbf{Z}^{(1)}_{\text{DCT-II}}$ to $\mathbf{Z}^{(6)}_{\text{DCT-II}}$

(c) $\mathbf{Z}^{(1)}_{\text{DCT-III}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-III}}$ (d) $\mathbf{Z}^{(1)}_{\text{DCT-IV}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-IV}}$

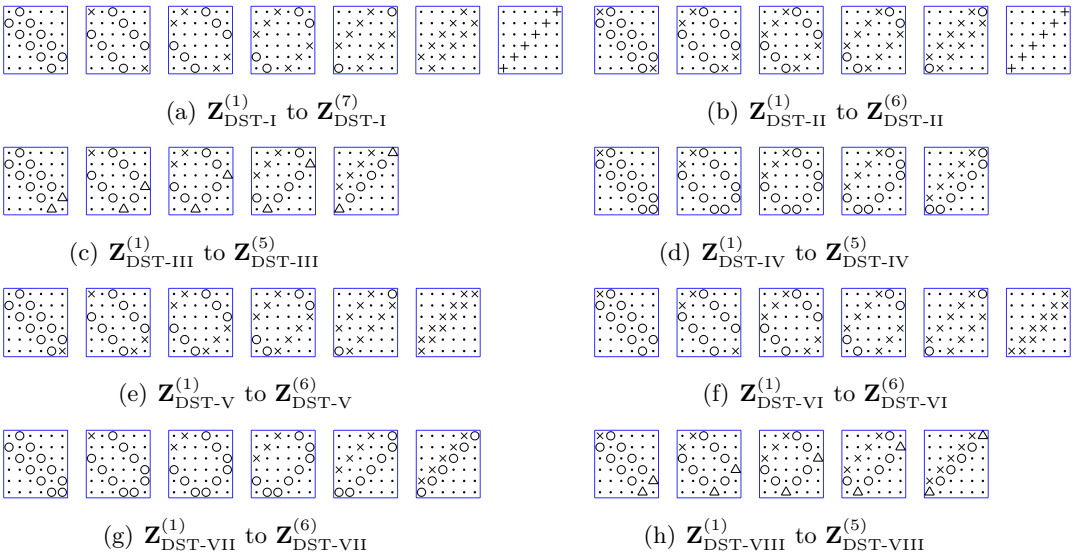
(e) $\mathbf{Z}^{(1)}_{\text{DCT-V}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-V}}$ (f) $\mathbf{Z}^{(1)}_{\text{DCT-VI}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-VI}}$

(g) $\mathbf{Z}^{(1)}_{\text{DCT-VII}}$ to $\mathbf{Z}^{(5)}_{\text{DCT-VII}}$ (h) $\mathbf{Z}^{(1)}_{\text{DCT-VIII}}$ to $\mathbf{Z}^{(6)}_{\text{DCT-VIII}}$

Figure 5.2: Sparse graph operators with length $N = 6$ that associated to DCT-I to DCT-VIII. Different symbols represent different values: $\times = -1$, $\cdot = 0$, $\bigcirc = 1$, $\triangle = \sqrt{2}$, and $\square = 2$.

arbitrary length. Interestingly, we observe that the non-zero entries in all sparse operators have rectangle-like patterns. Indeed, the 16 DTTs are constructed with combinations of 4 types of left boundary conditions and 4 types of right boundary conditions, which are associated to 4 types of upper-left rectangle edges and 4 types of lower-right rectangle edges in Figures 5.2 and 5.3.

Some of DCT sparse operators and their variants in Figures 5.2 and 5.3 were already known such as $\mathbf{Z}^{(1)}_{\text{DCT-I}}$ [57], $\mathbf{I} + \mathbf{Z}^{(1)}_{\text{DCT-III}}$ and $\mathbf{I} + \mathbf{Z}^{(1)}_{\text{DCT-IV}}$ [43], whose results are then summarized in [103] under a more general framework. In [99], left and right boundary conditions have been exploited to obtain sparse matrices with DTT eigenvectors, which correspond to the first operator $\mathbf{Z}^{(1)}$ for each DTT. However, to the best of our knowledge, graph operators with $\ell > 1$ (i.e., $\mathbf{Z}^{(2)}$ to $\mathbf{Z}^{(N-1)}$ for each DTT) have not been studied in the literature.



(a) $\mathbf{Z}^{(1)}_{\text{DST-I}}$ to $\mathbf{Z}^{(7)}_{\text{DST-I}}$ (b) $\mathbf{Z}^{(1)}_{\text{DST-II}}$ to $\mathbf{Z}^{(6)}_{\text{DST-II}}$

(c) $\mathbf{Z}^{(1)}_{\text{DST-III}}$ to $\mathbf{Z}^{(5)}_{\text{DST-III}}$ (d) $\mathbf{Z}^{(1)}_{\text{DST-IV}}$ to $\mathbf{Z}^{(5)}_{\text{DST-IV}}$

(e) $\mathbf{Z}^{(1)}_{\text{DST-V}}$ to $\mathbf{Z}^{(6)}_{\text{DST-V}}$ (f) $\mathbf{Z}^{(1)}_{\text{DST-VI}}$ to $\mathbf{Z}^{(6)}_{\text{DST-VI}}$

(g) $\mathbf{Z}^{(1)}_{\text{DST-VII}}$ to $\mathbf{Z}^{(6)}_{\text{DST-VII}}$ (h) $\mathbf{Z}^{(1)}_{\text{DST-VIII}}$ to $\mathbf{Z}^{(5)}_{\text{DST-VIII}}$

Figure 5.3: Sparse graph operators with length $N = 6$ that associated to DST-I to DST-VIII. Different symbols represent different values: $+ = -2$, $\times = -1$, $\cdot = 0$, $\bigcirc = 1$, and $\triangle = \sqrt{2}$.

5.2.3 Sparse 2D DCT/DST Filters

In image and video coding, the DTTs are often applied to 2D pixel blocks, where a combination of 1D DTTs can be applied to columns and rows of the blocks. We consider a $N_1 \times N_2$ block (with N_1 pixel rows and N_2 pixel columns),

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,N_2} \\ X_{2,1} & X_{2,2} & \dots & X_{2,N_2} \\ \vdots & \vdots & \vdots & \vdots \\ X_{N_1,1} & X_{N_1,2} & \dots & X_{N_1,N_2} \end{bmatrix}.$$

We use a 1D vector $\mathbf{x} \in \mathbb{R}^{N_1 N_2}$ to denote $\mathbf{X}$ with column-first ordering:

$$\mathbf{x} = (X_{1,1}, X_{2,1}, \dots, X_{N_1,1}, X_{1,2}, X_{2,2}, \dots, X_{N1,2}, \dots, X_{N_1,N_2})^\top$$

We assume that the GFT $\mathbf{\Phi} = \mathbf{\Phi}_r \otimes \mathbf{\Phi}_c$ is separable with row transform $\mathbf{\Phi}_r$ and column transform $\mathbf{\Phi}_c$. In such cases, sparse operators of 2D separable GFTs can be obtained from those of 1D transforms:

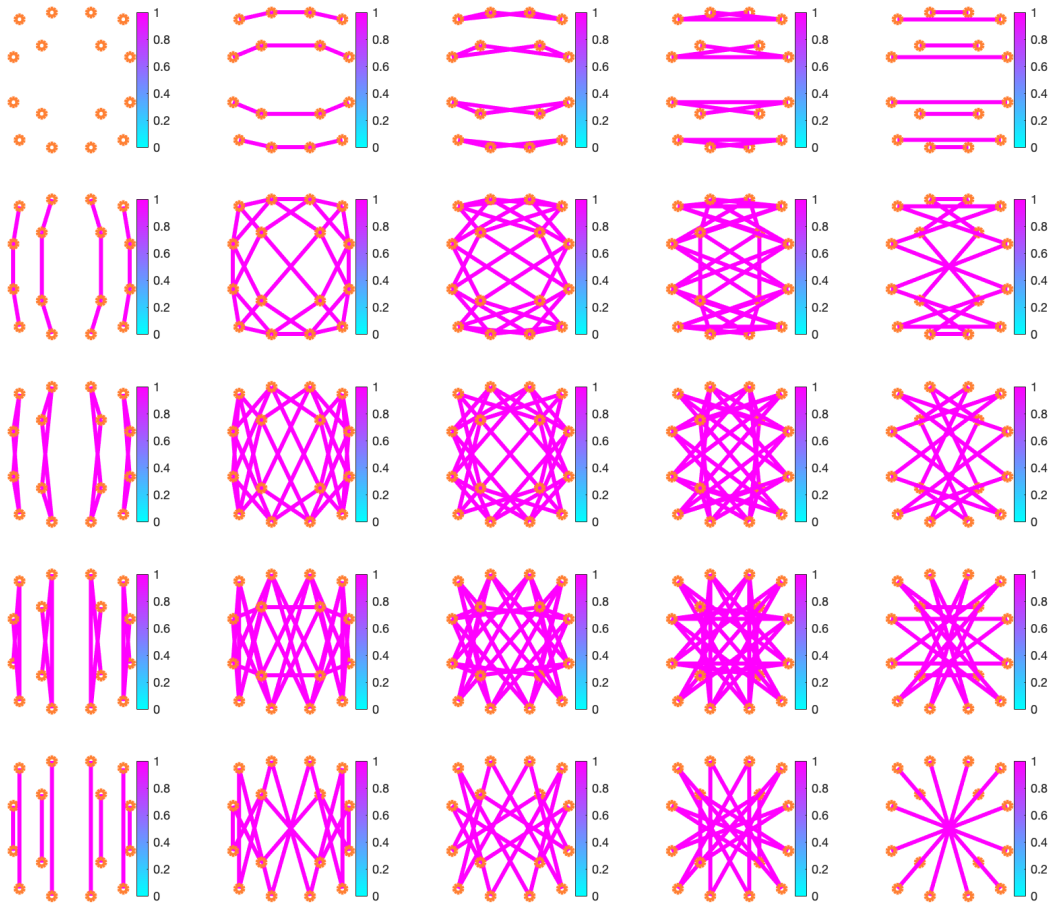


Figure 5.4: Graphs associated to sparse operators of 2D 4×4 DCT. For visualization, coordinates are slightly shifted to prevent some edges from overlapping. Self-loops are not shown in the graphs.

Proposition 2 (Sparse 2D DTT operators)**.** *Let* $\mathbf{\Phi} = \mathbf{\Phi}_r \otimes \mathbf{\Phi}_c$ *with* $\mathbf{\Phi}_r$ *and* $\mathbf{\Phi}_c$ *being orthogonal transforms among the 16 DTTs, and let* $\mathcal{Z}_r$ *and* $\mathcal{Z}_c$ *be the set of sparse operators associated to* $\mathbf{\Phi}_r$ *and* $\mathbf{\Phi}_c$*, respectively. Denote the eigenpairs associated to the operators of* $\mathcal{Z}_r$ *and* $\mathcal{Z}_c$ *as* $(\lambda_{r,j}, \boldsymbol{\phi}_{r,j})$ *and* $(\lambda_{c,k}, \boldsymbol{\phi}_{c,k})$ *with* $j = 1, \ldots, N_1$ *and* $k = 1, \ldots, N_2$*. Then,*

$$\mathcal{Z} = \{\mathbf{Z}_r \otimes \mathbf{Z}_c,\ \mathbf{Z}_r \in \mathcal{Z}_r,\ \mathbf{Z}_c \in \mathcal{Z}_c\}$$

is a set of sparse operators corresponding to $\mathbf{\Phi}_r \otimes \mathbf{\Phi}_c$*, with associated eigenpairs* $(\lambda_{r,j}\lambda_{c,k}, \boldsymbol{\phi}_{r,j} \otimes \boldsymbol{\phi}_{c,k})$*.*

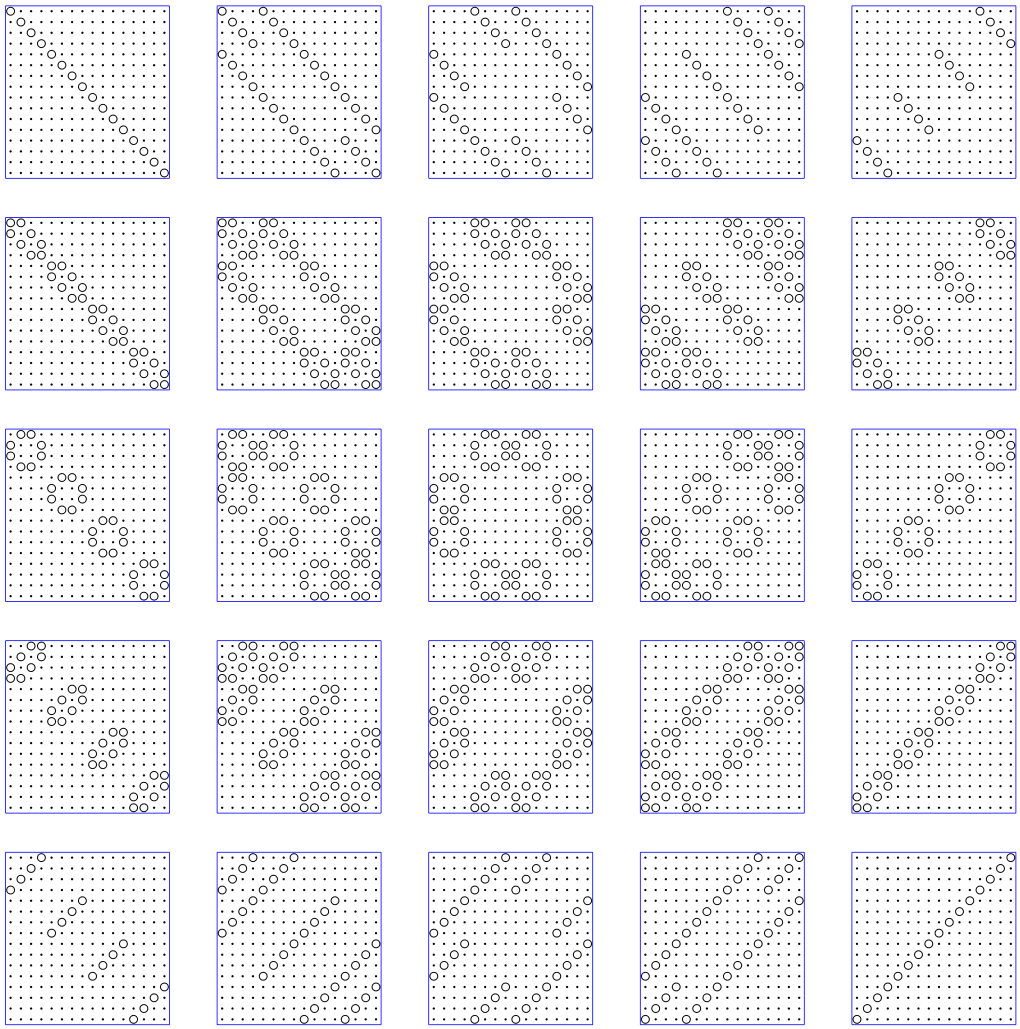


Figure 5.5: Sparse operators associated to 2D 4×4 DCT. Symbols $\cdot$ and $\bigcirc$ represent 0 and 1, respectively.

Proof: Let $\mathbf{Z}_r^{(1)}, \ldots, \mathbf{Z}_r^{(M_1)}$ be sparse operators in $\mathcal{Z}_r$ with associated eigenvalues contained in vectors $\boldsymbol{\lambda}_r^{(1)}, \ldots, \boldsymbol{\lambda}_r^{(M_1)}$, respectively. Also let $\mathbf{Z}_c^{(1)}, \ldots, \mathbf{Z}_c^{(M_1)}$ be those in $\mathcal{Z}_c$ with eigenvalues in $\boldsymbol{\lambda}_c^{(1)}, \ldots, \boldsymbol{\lambda}_c^{(M_2)}$, respectively. We note that

$$\begin{aligned}
\mathbf{Z}_r^{(m_1)} &= \boldsymbol{\Phi}_r \cdot \mathrm{diag}(\boldsymbol{\lambda}_r^{(m_1)}) \cdot \boldsymbol{\Phi}_r^\top, \quad m_1 = 1, \ldots, M_1, \\
\mathbf{Z}_c^{(m_2)} &= \boldsymbol{\Phi}_c \cdot \mathrm{diag}(\boldsymbol{\lambda}_c^{(m_2)}) \cdot \boldsymbol{\Phi}_c^\top, \quad m_2 = 1, \ldots, M_2.
\end{aligned}$$

Applying a well-known Kronecker product identity [145], we obtain

$$\mathbf{Z}_r^{(m_1)} \otimes \mathbf{Z}_c^{(m_2)} = \mathbf{\Phi} \cdot \mathrm{diag}(\boldsymbol{\lambda}_r^{(m_1)} \otimes \boldsymbol{\lambda}_c^{(m_2)}) \cdot \mathbf{\Phi}^\top. \qquad \square$$

In Proposition 2, we allow $\mathbf{\Phi}_c$ and $\mathbf{\Phi}_r$ to be the same. An example is shown in Figures 5.4 and 5.5, where $\mathbf{\Phi}_c = \mathbf{\Phi}_r$ is the length-4 DCT-II, and $\mathbf{\Phi}$ is the 4×4 2D DCT.

5.2.4 Remarks on Graph Operators of Arbitrary GFTs

Obtaining multiple sparse operators $\mathbf{Z}^{(k)}$ for a fixed GFT $\mathbf{\Phi} \in \mathbb{R}^{N \times N}$ is a challenging problem in general. Let the graph Laplacian associated to $\mathbf{\Phi}$ be $\mathbf{L}$, and λ_j be the eigenvalue of $\mathbf{L}$ associated to eigenvector $\boldsymbol{\phi}_j$. It can be seen that, if the graph does not have any self-loops, the Laplacian of the complement graph [86]

$$\mathbf{L}^c := N w_{\max} \mathbf{I} - w_{\max} \mathbf{1}\mathbf{1}^\top - \mathbf{L},$$

has eigenpairs $(0, \boldsymbol{\phi}_1)$ and $(n - \lambda_j, \boldsymbol{\phi}_j)$ for $j = 2, \ldots, N$. However, $\mathbf{L}^c$ will be a dense matrix when $\mathbf{L}$ is sparse, and thus it may not provide an efficient MPGF design.

Some additional remarks on the retrieval of sparse graph operators are presented as follows. More details on those remarks can be found in Appendix C.

Characterization of Sparse Laplacians with a Common GFT

Extending a key result in [92], we can characterize the set of all graph Laplacians (i.e., that satisfy (1.4) with non-negative edge and self-loop weights) sharing a given GFT $\mathbf{\Phi}$ by a convex polyhedral cone. In particular, those graph Laplacians that are the most sparse among all correspond to the edges of a polyhedral cone (i.e., where the faces of the cone meet each other). However, the enumeration of edges is in general an NP-hard problem since the number of polyhedron vertices or edges can be a combinatorial number of N.

Construction of Sparse Operators from Symmetric Graphs

If a graph with Laplacian $\mathbf{L}$ satisfies the φ-symmetry property in Definition 2, then we can construct a sparse operator in addition to $\mathbf{L}$. Recall that for an involution φ on graph vertices, a graph is φ-symmetric if $w_{i,j} = w_{\varphi(i),\varphi(j)}$ for all $i, j \in \mathcal{V}$. For such a graph, a sparse operator can be constructed as follows:

Lemma 7. *Given a φ-symmetric graph $\mathcal{G}$ with Laplacian $\mathbf{L}$, we can construct a graph $\overline{\mathcal{G}_\varphi}$ by connecting nodes i and j with edge weight 1 for all node pairs (i, j) with $\varphi(i) = j, i \neq j$). In this way, the Laplacian $\overline{\mathbf{L}_\varphi}$ of $\overline{\mathcal{G}_\varphi}$ commutes with $\mathbf{L}$.*

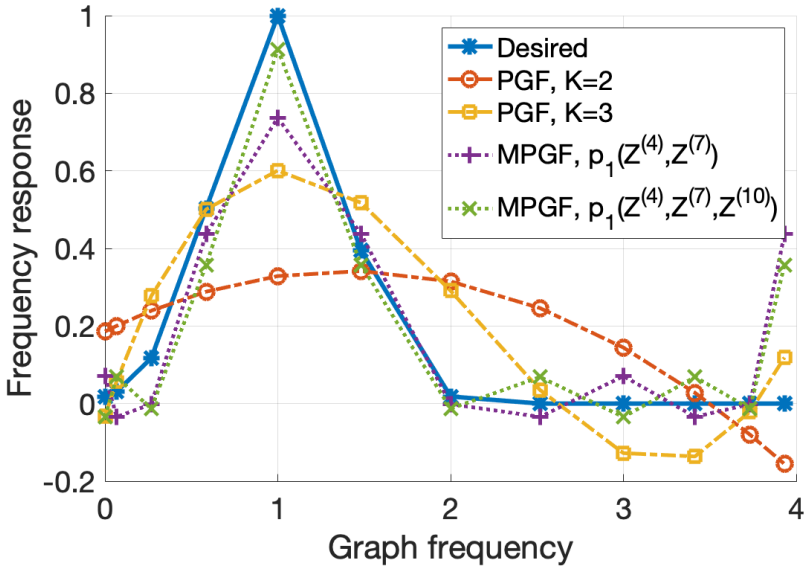


Figure 5.6: An example for PGF and MPGF fitting results on a length 12 line graph. The desired frequency response is $h^*(\lambda) = \exp(-4(\lambda-1)^2)$. The PGF and MPGF filters have been optimized based on (1.6) and (5.10).

The proof of this theorem and a illustrative example are presented in Appendix D.

5.3 New Graph Filter Designs with Sparse Operators

In this section, we introduce some filter design approaches based on sparse operators for DTTs. The least squares design method will be summarized in Section 5.3.1. We also propose a minimax filter design in Section 5.3.2 for both PGF and MPGF. Then, in Section 5.3.3 we show that weighted energy in graph frequency domain can also be efficiently approximated using multiple graph operators.

5.3.1 Least Squares (LS) Graph Filter

For an arbitrary graph filter $\mathbf{H}^*$, its frequency response $\mathbf{h}^* = (h^*(\lambda_1), \ldots, h^*(\lambda_N))^\top$, can be approximated with a filter $\mathbf{H}_{\mathcal{Z},K}$ in (5.1) by designing a set of coefficients $\mathbf{g}$ as in

(5.2) or (5.3). Let $h(\lambda_j)$ be the frequency response of corresponding to $\mathbf{H}_{\mathcal{Z},K}$, then one way to obtain $\mathbf{g}$ is through a least squares solution:

$$\begin{aligned}\mathbf{g}^* &= \underset{\mathbf{g}}{\operatorname{argmin}} \quad \sum_{j=1}^{N} \left(h^*(\lambda_j) - h(\lambda_j)\right)^2 \\ &= \underset{\mathbf{g}}{\operatorname{argmin}} \quad \sum_{j=1}^{N} \left(h^*(\lambda_j) - p_K(\lambda_j^{(1)}, \ldots, \lambda_j^{(M)})\right)^2 \\ &= \underset{\mathbf{g}}{\operatorname{argmin}} \quad \left\|\mathbf{h}^* - \mathbf{\Pi}_K(\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(M)}) \cdot \mathbf{g}\right\|^2, \end{aligned} \tag{5.10}$$

where $\mathbf{\Pi}_K$ for $K = 1$ and $K = 2$ are

$$\mathbf{\Pi}_1(\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(M)}) = \begin{pmatrix} 1 & \lambda_1^{(1)} & \ldots & \lambda_1^{(M)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \lambda_N^{(1)} & \ldots & \lambda_N^{(M)} \end{pmatrix},$$

$$\mathbf{\Pi}_2(\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(M)}) = \begin{pmatrix} 1 & \lambda_1^{(1)} & \ldots & \lambda_1^{(M)} & \lambda_1^{(1)}\lambda_1^{(1)} & \lambda_1^{(1)}\lambda_1^{(2)} & \ldots & \lambda_1^{(M)}\lambda_1^{(M)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & \lambda_N^{(1)} & \ldots & \lambda_N^{(M)} & \lambda_N^{(1)}\lambda_N^{(1)} & \lambda_N^{(1)}\lambda_N^{(2)} & \ldots & \lambda_N^{(M)}\lambda_N^{(M)} \end{pmatrix}.$$

This formulation can be generalized to a weighted least squares problem, where we allow different weights for different graph frequencies. This enables us to approximate the filter in particular frequencies with higher accuracy. In this case, we consider

$$\mathbf{g}^* = \underset{\mathbf{g}}{\operatorname{argmin}} \quad \sum_{j=1}^{N} \rho_i^2 \left(h^*(\lambda_j) - h(\lambda_j)\right)^2 = \underset{\mathbf{g}}{\operatorname{argmin}} \quad \|\operatorname{diag}(\boldsymbol{\rho})(\mathbf{h}^* - \mathbf{\Pi}_K \cdot \mathbf{g})\|^2, \tag{5.11}$$

where $\rho_i \geq 0$ is the weight corresponding to λ_i. Note that when $\boldsymbol{\rho} = \mathbf{1}$, the problem (5.11) reduces to (5.10).

When $\mathbf{g}$ is more sparse, (i.e., has a smaller ℓ_0 norm), fewer terms will be involved in the polynomial p_K, leading to a lower complexity for the filtering operation. This ℓ_0-constrained problem can be viewed as a sparse representation of $\operatorname{diag}(\boldsymbol{\rho})\mathbf{h}^*$ in an overcomplete dictionary $\operatorname{diag}(\boldsymbol{\rho})\mathbf{\Pi}_K$. Well-known methods for this problem include the orthogonal matching pursuit (OMP) algorithm [93], and the optimization with a sparsity-promoting ℓ_1 constraint:

$$\underset{\mathbf{g}}{\text{minimize}} \quad \|\operatorname{diag}(\boldsymbol{\rho})(\mathbf{h}^* - \mathbf{\Pi}_K \cdot \mathbf{g})\|^2 \quad \text{subject to} \quad \|\mathbf{g}\|_1 \leq \tau, \tag{5.12}$$

where τ is a pre-chosen threshold. In fact, this formulation can be viewed as an extension of its PGF counterpart [115] to an MPGF setting. Note that (5.12) is a ℓ_1-constrained least squares problem (a.k.a., the LASSO problem), where efficient solvers are available [126].

Compared to conventional PGF $\mathbf{H}$ in (1.6), the implementation with $\mathbf{H}_{\mathcal{Z},K}$ has several advantages. First, when $K = 1$, the MPGF (5.2) is a linear combination of different sparse operators, which is amenable to parallelization. This is contrast to high degree PGFs based on (1.6) that require applying the graph operator repeatedly. Second, $\mathbf{H}_{\mathcal{Z},K}$ is a generalization of $\mathbf{H}$ and provides more degrees of freedom, which may lead to a more accurate approximation with a lower order polynomial. Note that, while the eigenvalues of $\mathbf{Z}^k$ for $k = 1, 2, \ldots$ are typically all increasing (if $\mathbf{Z} = \mathbf{L}$) or decreasing (if, for instance, $\mathbf{Z} = 2\mathbf{I} - \mathbf{L}$), those of different $\mathbf{Z}^{(m)}$'s have more diverse distributions (i.e., increasing, decreasing, or non-monotnoic). Thus, it is more likely that MPGFs can efficiently approximate filters with non-monotonic frequency responses. For example, we demonstrate in Figure 5.6 the resulting PGF and MPGF for a bandpass filter. We can see that, for $K = 2$ and $K = 3$, a degree-1 MPGF with K operators gives a higher approximation accuracy than a degree-K PGF, while they have a similar complexity.

5.3.2 Minimax Graph Filter

The minimax approach is a popular filter design method in classical signal processing. The goal is to design an length-K FIR filter whose frequency response $G(e^{j\omega})$ approximates the desired frequency response $H(e^{j\omega})$ in a way that the maximum error within some range of frequency is minimized. A standard design method is the Parks-McClellan algorithm, which is a variation of the Remez exchange algorithm [83]. Here, we explore minimax design criteria for graph filters. We denote $h^*(\lambda)$ the desired frequency response, and $g(\lambda)$ the polynomial filter that approximates $h^*(\lambda)$.

Polynomial Graph Filter

Let $g(\lambda)$ be the PGF with degree K given by (1.6). Since graph frequencies $\lambda_1, \ldots, \lambda_N$ are discrete, we only need to minimize the maximum error between h^* and g at frequencies $\lambda_1, \ldots, \lambda_N$. In particular, we would like to solve polynomial coefficients g_i:

$$\underset{\mathbf{b}}{\text{minimize}} \quad \underbrace{\max_i \rho_i \left| h^*(\lambda_i) - \sum_{j=0}^{K} g_j \lambda_i^j \right|}_{\|\text{diag}(\boldsymbol{\rho})(\mathbf{h}^* - \boldsymbol{\Psi}\mathbf{g})\|_\infty}$$

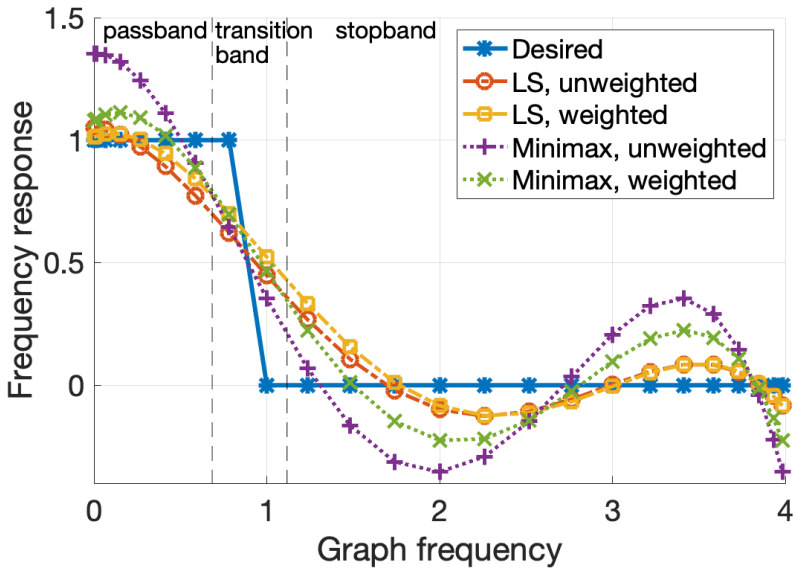


Figure 5.7: Example illustrating the frequency responses of degree $K = 4$ PGF with least squares (LS) and minimax criteria, with weighted or unweighted settings. The filters are defined on a length 24 line graph. In the weighted setting, weights ρ_i are chosen to be 2, 0, and 1 for passband, transition band, and stopband, respectively.

where $\mathbf{\Psi}$ is the matrix in (1.8), ρ_i is the weight associated to λ_i and $\|\cdot\|_\infty$ represents the infinity norm. Note that, when $K \geq N-1$ and $\mathbf{\Psi}$ is full row rank, then $\mathbf{h}^* = \mathbf{\Psi}\mathbf{g}$ can be achieved with $\mathbf{g} = \mathbf{\Psi}^\dagger \mathbf{h}^*$. Otherwise, we reduce this problem by setting $\epsilon = \|\text{diag}(\boldsymbol{\rho})\,(\mathbf{h}^* - \mathbf{\Psi}\mathbf{g})\,\|_\infty$:

$$\underset{\mathbf{g},\,\epsilon}{\text{minimize}} \quad \epsilon \quad \text{subject to} \quad -\epsilon\mathbf{1} \preceq \text{diag}(\boldsymbol{\rho})\,(\mathbf{h}^* - \mathbf{\Psi}\mathbf{g}) \preceq \epsilon\mathbf{1}, \tag{5.13}$$

whose solution can be efficiently obtained with a linear programming solver.

Multivariate Polynomial Graph Filter

Now we consider $g(\lambda)$ a graph filter with M graph operators with degree K, as in (5.1). In this case, we can simply extend the problem (5.13) to

$$\underset{\mathbf{g},\,\epsilon}{\text{minimize}} \quad \epsilon \quad \text{subject to} \quad -\epsilon\mathbf{1} \preceq \text{diag}(\boldsymbol{\rho})\,(\mathbf{h}^* - \mathbf{\Pi}_K\mathbf{g}) \preceq \epsilon\mathbf{1}, \tag{5.14}$$

where a ℓ_1 or ℓ_0 norm constraint on $\mathbf{g}$ can also be considered.

To summarize, we show in Table 5.2 the objective functions of least squares and minimax designs with PGF and MPGF, where weights on different graph frequencies

	PGF	MPGF
Least squares	$\min_{\mathbf{g}} \quad \|\|\text{diag}(\boldsymbol{\rho})(\mathbf{h}-\boldsymbol{\Psi}\mathbf{g})\|\|^2$	(5.11)
Minimax	(5.13)	(5.14)

Table 5.2: Least squares and minimax design approaches of for PGF and MPGF, with weights ρ_i on different graph frequencies.

are considered. Note that the least squares PGF design shown in Table 5.2 is a simple extension of the unweighted design (1.8) in [105].

Using an ideal low-pass filter as the desired filter, we show a toy example with degree-4 PGF in Figure 5.7. When different weights ρ_i are used for passband, transition band, and stopband, approximation accuracies differ for different graph frequencies. By comparing LS and minimax results in a weighted setting, we also see that the minimax criterion yields a smaller maximum error within the passband (see the last frequency bin in passband) and stopband (see the first frequency bin in stopband).

5.3.3 Weighted GFT Domain Energy Evaluation

Let $\mathbf{x}$ be a signal and $\boldsymbol{\Phi}$ be a GFT to be applied, we consider a weighted sum of squared GFT coefficients:

$$\mathcal{C}_{\boldsymbol{\Phi}}(\mathbf{x};\mathbf{q}) = \sum_{i=1}^{N} q_i(\boldsymbol{\phi}_i^\top \mathbf{x})^2, \tag{5.15}$$

where arbitrary weights $\mathbf{q} = (q_1, \ldots, q_N)^\top$ can be considered. Denote the graph Laplacian associated to $\boldsymbol{\Phi}$ as $\mathbf{L} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\top$, then $\mathcal{C}_{\boldsymbol{\Phi}}(\mathbf{x};\mathbf{q})$ has a similar form to the Laplacian quadratic form (1.5), since

$$\mathbf{x}^\top \mathbf{L}\mathbf{x} = \sum_{l=1}^{N} \lambda_l(\boldsymbol{\phi}_l^\top \mathbf{x})^2. \tag{5.16}$$

Note that computation of $\mathbf{x}^\top\mathbf{L}\mathbf{x}$ using (1.5) can be done in the vertex domain, and does not require the GFT coefficients. This provides a low complexity implementation than (5.16), especially when the graph is sparse (i.e., few edges and self-loops).

Similar to vertex domain Laplacian quadratic form computation (1.5), we note that $\mathcal{C}_{\boldsymbol{\Phi}}(\mathbf{x};\mathbf{q})$ can also be realized as a quadratic form:

$$\mathcal{C}_{\boldsymbol{\Phi}}(\mathbf{x};\mathbf{q}) = \sum_{i=1}^{N} q_i(\boldsymbol{\phi}_i^\top \mathbf{x})^2 = \mathbf{x}^\top \underbrace{\left(\boldsymbol{\Phi}\cdot\text{diag}(\mathbf{q})\cdot\boldsymbol{\Phi}^\top\right)}_{\mathbf{H}_{\mathbf{q}}}\mathbf{x}, \tag{5.17}$$

where $\mathbf{H_q}$ can be viewed as a graph filter with frequency response $h_{\mathbf{q}}(\lambda_i) = q_i$. Thus, we can approximate $\mathbf{H_q}$ with a sparse filter $\mathbf{H}_{\hat{\mathbf{q}}}$ such that $\mathbf{x}^\top \mathbf{H}_{\hat{\mathbf{q}}} \mathbf{x}$ approximates $\mathcal{C}_{\mathbf{\Phi}}(\mathbf{x}; \mathbf{q})$. For example, if we consider a polynomial with degree 1 as in (5.2), we have

$$\mathbf{x}^\top \underbrace{\left[g_0 \mathbf{I} + \sum_{m=1}^{M} g_m \mathbf{Z}^{(m)} \right]}_{\mathbf{H}_{\hat{\mathbf{q}}}} \mathbf{x} = \sum_{i=1}^{N} \underbrace{\left(g_0 + \sum_{m=1}^{M} g_m \lambda_i^{(m)} \right)}_{\hat{q}_i} (\boldsymbol{\phi}_i^\top \mathbf{x})^2. \tag{5.18}$$

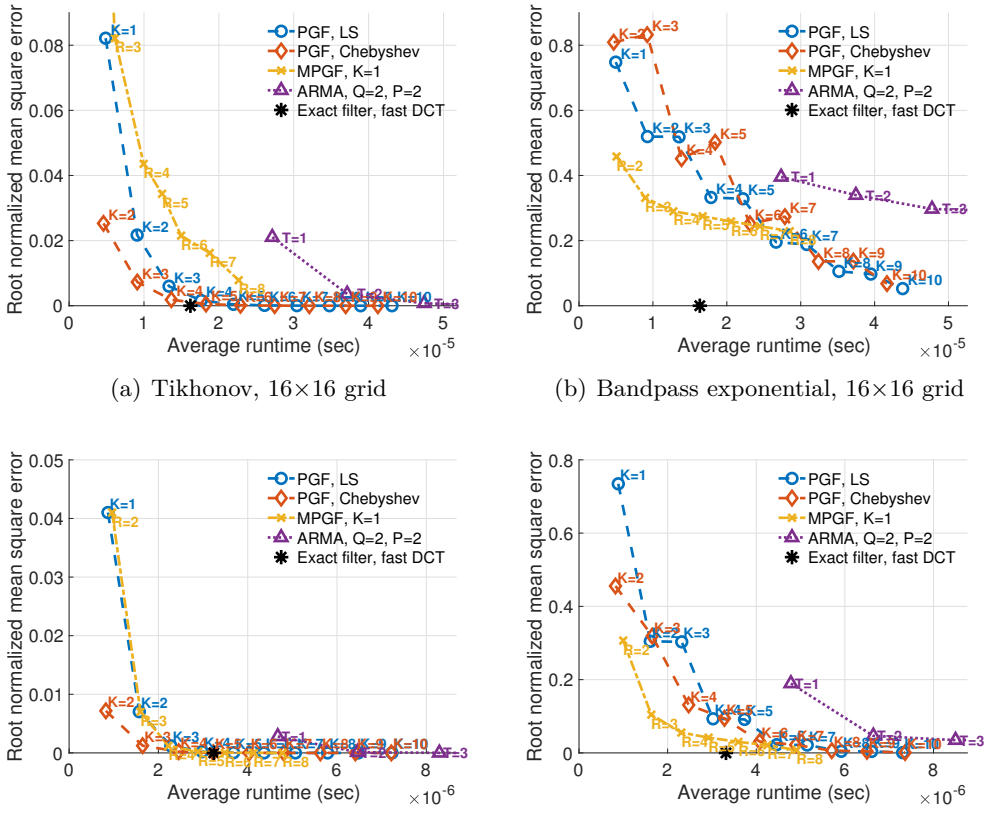
The left hand side can be computed efficiently if there are only a few nonzero g_m, making $\mathbf{H}_{\hat{\mathbf{q}}}$ sparse. The right hand side can be viewed as a proxy of (5.15) if g_m's are chosen such that $\hat{q}_i \approx q_i$. Such coefficients g_m can be obtained by solving (5.12) with $\mathbf{h}^* = \mathbf{q}$.

5.3.4 Complexity Analysis

For a graph with N nodes and E edges, it has been shown in [16] that a degree-K PGF has $\mathcal{O}(KE)$ complexity. For an MPGF with R terms, we denote E' the maximum sparsity of the operator among all operators involved. Each term of MPGF requires at most $\mathcal{O}(KE')$ operations, so the overall complexity of an MPGF is $\mathcal{O}(KRE')$. We note that for DTT filters, the sparsity of all operators we have introduced is at most $2N$. Thus, complexities of PGF and MPGF can be reduced to $\mathcal{O}(KN)$ and $\mathcal{O}(KRN)$, respectively. We note that $\mathcal{O}(KRN)$ is not a tight upper bound for the complexity if many terms of the MPGF have lower degrees than K. In addition, the polynomial degree required by an MPGF to reach a similar accuracy as a PGF can achieve may be lower. Thus, an MPGF does not necessarily have higher complexity than a PGF that bring a similar approximation accuracy. Indeed, MPGF implementation may be further optimized by parallelizing the computation associated to different graph operators.

5.4 Experiments

We consider two experiments to validate the filter design approaches. In Section 5.4.1, we evaluate the complexity of PGF and MPGF for DCT-II, and compare the trade-off between complexity and filter approximation accuracy as compared to conventional implementations in the DCT domain. In Section 5.4.2 we implement DTT filters in a state-of-the-art video encoder–AV1, where we obtain a computational speedup in transform type search.



(a) Tikhonov, 16×16 grid

(b) Bandpass exponential, 16×16 grid

(c) Tikhonov, length-64 line graph

(d) Bandpass exponential, length-64 line graph

Figure 5.8: Runtime vs approximation error for (a)(c) Tikhonov DCT filter, (b)(d) bandpass exponential DCT filter. Those filters are defined based on two different graphs: (a)(b) 16×16 grid, (c)(d) length-64 line graph. Different PGF degrees K, MPGF operators involved R, and ARMA iteration numbers T, are labelled in the figures.

5.4.1 Filter Approximation Accuracy with Respect to Complexity

In the first experiment, we implement several DCT filters on a 16×16 grid, and a length-64 line graph. Those filters are implemented in C in order to fairly evaluate computational complexity under an environment close to hardware.

Comparison among filter implementations.

First, the following filters are considered:

- *Tikhonov filter:* given $\mathbf{z} = \mathbf{x} + \mathbf{n}$, a noisy observation of signal $\mathbf{x}$, the denoising problem can be formulated as a regularized least squares problem:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{x} - \mathbf{z}\|^2 + \mu \mathbf{x}^\top \mathbf{L} \mathbf{x}.$$

  The solution is given by $\hat{\mathbf{x}} = \mathbf{H}_t \mathbf{x}$, where $\mathbf{H}_t = (\mathbf{I} + \mu \mathbf{L})^{-1}$ is known as the Tikhonov graph filter with frequency response $h_t(\lambda) = 1/(1 + \mu\lambda)$. Applications of the Tikhonov filter in graph signal processing include signal denoising [114], classification [78], and inter-predicted video coding [143].

- *Bandpass exponential filter:* bandpass graph filters are key components in M-channel graph filter banks [122, 123]. Here, we consider the frequency response

$$h_{\exp}(\lambda) = \exp(-\gamma(\lambda - \lambda_{pb})^2),$$

  where γ is a decaying factor and λ_{pb} is the central frequency of the passband.

For the choice of parameters, we use $\mu = 0.25$, $\gamma = 1$, and $\lambda_{pb} = 0.5\lambda_{max}$ in this experiment, to characterize a smooth Tikhonov filter and a bandpass filter with a symmetric frequency response, respectively. The following filter implementations are compared:

- *Polynomial DCT filter:* given the desired frequency response, two design methods for PGF coefficients are considered. The first method is a least squares design (PGF, LS) [105] with iterative implementation described in Section 1.2.2. The second method is the Chebyshev polynomial approaches (PGF, Chebyshev) [115], where PGF is implemented based on recurrence relations of Chebyshev polynomials.

- *Multivariate polynomial DCT filter:* we consider all sparse graph operators (289 operators for the 16×16 grid and 65 operators for the length-64 line graph). Then, we obtain the least squares filter (5.10) with an ℓ_0 constraint and $K = 1$ using orthogonal matching pursuit, with R being 2 to 8.

- *Autoregressive moving average (ARMA) graph filter [51]*: we consider an IIR graph filter in rational polynomial form, i.e.,

$$\mathbf{H}_{\text{ARMA}} = \left(\sum_{p=0}^{P} a_p \mathbf{Z}^p \right)^{-1} \left(\sum_{q=0}^{Q} b_q \mathbf{Z}^q \right).$$

We choose polynomial degrees as $Q = P = 2$ and consider different numbers of iterations T. The graph filter implementation is based on the conjugate gradient approach described in [67], whose complexity is $\mathcal{O}((PT + Q)E)$.

- *Exact filter with fast DCT:* the filter operation is performed by a cascade of a forward DCT, a frequency masking with h, and an inverse DCT, where the forward and inverse DCTs are implemented using well-known fast algorithms [9]. For 4×4 or 16×16 grids, 2D separable DCTs are implemented, where a fast 1D DCT is applied to all rows and columns.

In LS designs, uniform weights $\boldsymbol{\rho} = \mathbf{1}$ are used. For each graph we consider, 20000 random input signals are generated. The complexity for each graph filter method is then evaluated as an average runtime over all 20000 trials. We measure the error between approximate and exact frequency responses with the root normalized mean square error $\|\mathbf{h}_{\text{approx}} - \mathbf{h}\|/\|\mathbf{h}\|$.

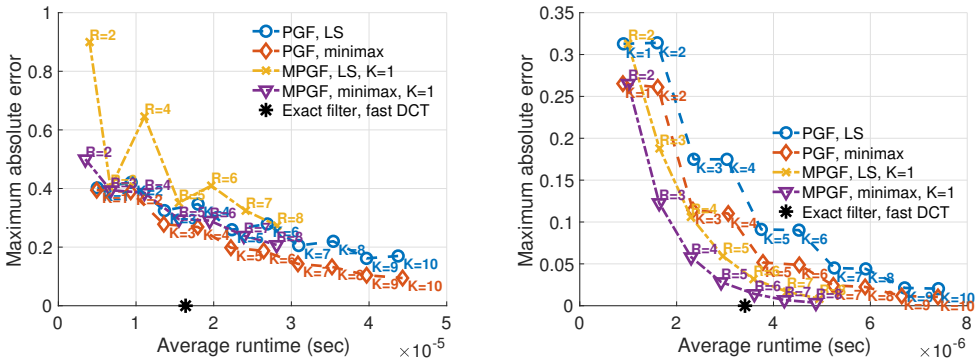
We show in Figure 5.8 the resulting runtimes and errors, where a point closer to the origin correspond to a better trade-off between complexity and approximation accuracy. We observe in Figure 5.8(a)(c) that low degree PGFs accurately approximate the Tikhonov filter, whose frequency response is closer to a linear function of λ. In Figure 5.8(b)(d), for bandpass exponential filter on the length-64 line graph, MPGF achieves a higher accuracy with lower complexity than PGF and ARMA graph filters. As discussed in Section 5.3.4, the complexity of PGF and MPGF grows linearly with the graph size, while the fast DCT algorithm has $\mathcal{O}(N \log N)$ complexity. Thus, PGF and MPGF would achieve a better speed performance with respect to exact filter when the graph size is larger. Note that in this experiment, a fast algorithm with $\mathcal{O}(N \log N)$ complexity for the GFT (DCT-II) is available. However, this is not always true for arbitrary graph size N, nor for other types of DTTs, where fast exact graph filter may not be available.

Evaluation of minimax designs.

Next, we consider an ideal low-pass filter:

$$h_{LP}(\lambda) = \begin{cases} 1, & 0 \leq \lambda \leq \lambda_c \\ 0, & \text{otherwise} \end{cases}$$

where $\lambda_c = 0.5\lambda_{max}$ is the cut-off frequency. The weight ρ_i is chosen to be 0 in the transition band $0.4 \leq \lambda_i \leq 0.6$, and 1 in passband and stopband. Figure 5.9 shows the resulting runtimes and approximation errors, which are measured with the maximum absolute error between approximate and desired frequency responses in passband and



(a) Ideal low-pass, 16×16 grid

(b) Ideal low-pass, length-64 line graph

Figure 5.9: Runtime vs maximum absolute error for various designs of ideal low-pass filter on (a) 16×16 grid, and (b) length-64 line graph.

stopband: $\max_i \rho_i |h_{\text{approx}}(\lambda_i) - h(\lambda_i)|$. We can see in Figure 5.9 that, when K or R increases, the maximum absolute error steadily decreases in PGF and MPGF designs with minimax criteria. In contrast, PGF and MPGF designs with LS criterion may lead to non-monotonic behavior in terms of the maximum absolute error as in Figure 5.9(a). In fact, under the LS criterion, using more sparse operators will reduce the least squares error, but does not always decrease the maximum absolute error.

Based on the results in Figures 5.8 and 5.9, we provide some remarks on the choice of DTT filter implementation:

- If the desired frequency response is close to a linear function of λ, e.g., Tikhonov filters with a small μ or graph diffusion processes [116], then a low-order PGF would be sufficiently accurate, and has the lowest complexity.

- If the graph size is small, or transform length allows a fast DTT algorithm, or when separable DTTs are available (e.g., on a 16×16 grid), DTT filter with fast DTT implementation would be favorable.

- For a sufficiently large length (e.g., $N = 64$) and a frequency response that is non-smooth (e.g., ideal low-pass filter) or non-monotonic (e.g., bandpass filter), an MPGF design may fit the desired filter with a reasonable speed performance. In particular, we note that $\mathbf{Z}^{(2)}$ is a bandpass filter with passband center $\lambda_{pb} = \lambda_{max}/2$. Thus, MPGF using $\mathbf{Z}^{(2)}$ would provide an efficiency improvement for bandpass filters with λ_{pb} close to $\lambda_{max}/2$.

- When robustness of the frequency response in the maximum absolute error sense is an important concern, a design based on minimax criterion would be preferable.

5.4.2 Transform Type Selection in Video Coding

In the second experiment, we consider the quadratic form (5.15) as a transform type cost, and apply the method described in Section 5.3.3 to speed up transform type selection in the AV1 codec [10]. In transform coding [41], (5.15) can be used as a proxy of the bitrate cost for block-wise transform type optimization [33,50]. In particular, we denote $\mathbf{x}$ an image or video block, and $\mathbf{\Phi}$ the orthogonal transform applied to $\mathbf{x}$. Lower bitrate cost can be achieved if $\mathbf{\Phi}$ gives a high energy compaction in the low frequencies, i.e., the energy of $\mathbf{\Phi}^\top \mathbf{x}$ is concentrated in the first few entries. Thus, the proxy of cost (5.15) can be defined with positive and increasing $\mathbf{q}$ ($0 < q_1 < \cdots < q_N$) to penalize large and high frequency coefficients, thus favoring transforms having more energy in the low frequencies.

AV1 includes four 1D transforms: 1) $\mathbf{U}$: DCT, 2) $\mathbf{V}$: ADST, 3) $\mathbf{JV}$: FLIPADST, which has flipped ADST functions, and 4) $\mathbf{I}$: IDTX (identity transform), where no transform will be applied. For small inter predicted blocks, all 2D combinations of 1D transforms are used. Namely, there are 16 2D transforms candidates, $(\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}})$ with $\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}} \in \{\mathbf{U}, \mathbf{V}, \mathbf{JV}, \mathbf{I}\}$, which makes the encoder computationally expensive. Recent work on encoder complexity reduction includes [65,74,119], which apply heuristic and data-driven techniques to prune transform types during the search.

To speed up transform type selection in AV1, for 1D pixel block $\mathbf{x} \in \mathbb{R}^N$, we choose the following increasing weights for (5.15)[1]:

$$q_i = \delta_i = 2 - 2\cos\left(\frac{(i - \frac{1}{2})\pi}{N}\right). \tag{5.19}$$

Then, different transform type costs would be given by (5.15) with different $\mathbf{\Phi}$, i.e., $\mathcal{C}_{\mathbf{T}}(\mathbf{x}; \mathbf{q})$ with $\mathbf{T} \in \{\mathbf{U}, \mathbf{V}, \mathbf{JV}, \mathbf{I}\}$. This choice allows efficient computation of exact $\mathcal{C}_{\mathbf{V}}(\mathbf{x}; \mathbf{q})$ and $\mathcal{C}_{\mathbf{JV}}(\mathbf{x}; \mathbf{q})$ through their corresponding sparse Laplacian matrices:

$$\mathcal{C}_{\mathbf{V}}(\mathbf{x}; \mathbf{q}) = \mathbf{x}^\top \mathbf{L}_{\text{A}} \mathbf{x}, \quad \mathcal{C}_{\mathbf{JV}}(\mathbf{x}; \mathbf{q}) = \mathbf{x}^\top \mathbf{J}\mathbf{L}_{\text{A}}\mathbf{J}\mathbf{x},$$

where $\mathbf{J}\mathbf{L}_{\text{A}}\mathbf{J}$ is the left-right and up-down flipped version of $\mathbf{L}_{\text{A}}$. For the approximation of DCT cost $\mathcal{C}_{\mathbf{U}}(\mathbf{x}; \mathbf{q})$, we obtain $R = 3$ nonzeros polynomial coefficients g_m with degree

[1] As (5.15) serves as a proxy of the actual bitrate cost, we leave out the search of optimal weights. Weights are chosen to be increasing functions because transform coefficients associated to a higher frequency typically requires more bits to encode. The weights in (5.19) are used because of their computational for $\mathcal{Q}_V = \mathbf{x}_i^\top \mathbf{L}_{\text{A}} \mathbf{x}_i$. In fact, we have observed experimentally that different choices among several increasing weights produce similar coding results.

Table 5.3: Encoding time and quality loss (in BD rate) of different transform pruning methods. The baseline is AV1 with a full transform search (no pruning). A smaller loss is better.

Method	Encoding time	Quality loss
PRUNE_LAPLACIAN [74]	91.71%	0.32%
PRUNE_OPERATOR	89.05%	0.31%
PRUNE_2D_FAST [119]	86.78%	0.05%

$L = 1$ as in (5.18) using an exhaustive search. As a result, costs for all 1D transforms can be computed in the pixel domain as follows

$$\begin{aligned}
\mathcal{Q}_{\mathbf{U}} &= \mathbf{x}_i^\top \left(g_0 \mathbf{I} + \sum_{m=1}^{M} g_m \mathbf{Z}_{\text{DCT-II}}^{(m)} \right) \mathbf{x}_i \\
\mathcal{Q}_{\mathbf{V}} &= \mathcal{C}_{\mathbf{V}}(\mathbf{x}_i; \mathbf{q}) = \mathbf{x}_i^\top \mathbf{L}_{\text{A}} \mathbf{x}_i \\
\mathcal{Q}_{\mathbf{JV}} &= \mathcal{C}_{\mathbf{JV}}(\mathbf{x}_i; \mathbf{q}) = \mathbf{x}_i^\top \mathbf{J}\mathbf{L}_{\text{A}}\mathbf{J} \mathbf{x}_i \\
\mathcal{Q}_{\mathbf{I}} &= \mathcal{C}_{\mathbf{I}}(\mathbf{x}_i; \mathbf{q}) = \sum_j w_j \mathbf{x}_i(j)^2,
\end{aligned} \tag{5.20}$$

where M is the number of DCT operators and g_m has only $R = 3$ non-zero elements.

Extending an experiment PRUNE_LAPLACIAN in [74], we implemented a new experiment named PRUNE_OPERATORS in AV1[2]. We implement the integer versions of the transform cost evaluation (5.20) for transform lengths 4, 8, 16, and 32. Within each 2D block, we take an average over all columns or rows, to obtain column and row costs $\mathcal{Q}_{\mathbf{T}}^{(\text{col})}$ and $\mathcal{Q}_{\mathbf{T}}^{(\text{row})}$ with $\mathbf{T} \in \{\mathbf{U}, \mathbf{V}, \mathbf{JV}, \mathbf{I}\}$. Those costs are aggregated into 16 2D transform costs by summing the associated column and row costs. For example, the cost associated to vertical ADST and horizontal DCT is given by

$$\mathcal{Q}_{(\mathbf{V},\mathbf{U})} = \mathcal{Q}_{\mathbf{V}}^{(\text{col})} + \mathcal{Q}_{\mathbf{U}}^{(\text{row})}.$$

Finally, we design a pruning criteria, where each 2D column (or row) transform will be pruned if its associated cost is relatively large compared to the others.

C1. For $\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}} \in \{\mathbf{U}, \mathbf{V}, \mathbf{JV}\}$, prune $(\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}})$ if

$$\mathcal{Q}_{(\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}})} > \tau_1 \left(\mathcal{Q}_{\mathbf{U}}^{(\text{col})} + \mathcal{Q}_{\mathbf{V}}^{(\text{col})} + \mathcal{Q}_{\mathbf{JV}}^{(\text{col})} + \mathcal{Q}_{\mathbf{U}}^{(\text{row})} + \mathcal{Q}_{\mathbf{V}}^{(\text{row})} + \mathcal{Q}_{\mathbf{JV}}^{(\text{row})} \right).$$

C2. For $\mathbf{T}_{\text{col}} = \mathbf{I}$ or $\mathbf{T}_{\text{row}} = \mathbf{I}$, prune $(\mathbf{T}_{\text{col}}, \mathbf{T}_{\text{row}})$ if

[2]The experiment has been implemented on a version in July 2020. Available: https://aomedia-review.googlesource.com/c/aom/+/113461

Table 5.4: Encoding time and quality loss (in BD rate) of PRUNE_OPERATORS versus PRUNE_2D_FAST. Smaller or negative loss is better.

Sequence	Encoding time	Quality loss
akiyo	102.10%	0.00%
bowing	97.22%	-0.14%
bus	103.92%	-0.17%
city	102.36%	0.18%
crew	103.65%	0.07%
foreman	104.29%	0.07%
harbour	106.49%	-0.06%
ice	105.22%	0.30%
mobile	103.27%	0.23%
news	103.29%	-0.09%
pamphlet	97.75%	0.21%
paris	105.54%	0.21%
soccer	104.53%	0.22%
students	100.71%	0.03%
waterfall	102.34%	0.23%
Overall	102.61%	0.26%

$$\mathcal{Q}_{(\mathbf{T}_{\text{col}},\mathbf{T}_{\text{row}})} > \tau_2 \left(\mathcal{Q}_{\mathbf{U}}^{(\text{col})} + \mathcal{Q}_{\mathbf{V}}^{(\text{col})} + \mathcal{Q}_{\mathbf{JV}}^{(\text{col})} + \mathcal{Q}_{\mathbf{I}}^{(\text{col})} + \mathcal{Q}_{\mathbf{U}}^{(\text{row})} + \mathcal{Q}_{\mathbf{V}}^{(\text{row})} + \mathcal{Q}_{\mathbf{JV}}^{(\text{row})} + \mathcal{Q}_{\mathbf{I}}^{(\text{row})} \right),$$

where threshold parameters are chosen as $\tau_1 = 0.34$,$\tau_2 = 0.33$. Note that the number of 1D transforms being pruned can be different for different blocks. The pruning rules C1 do not depend on $\mathcal{Q}_{\mathbf{I}}$ because IDTX tends to have a larger bitrate cost with a significantly lower computational complexity than the other transforms. Thus, more aggressive pruning criteria C1 is applied to **U**, **V**, and **JV** to reduce more encoding time.

This pruning scheme is evaluated using 15 benchmark test sequences: `akiyo`, `bowing`, `bus`, `city`, `crew`, `foreman`, `harbour`, `ice`, `mobile`, `news`, `pamphlet`, `paris`, `soccer`, `students`, and `waterfall`. The results are shown in Table 5.3, where the speed improvement is measured in the percentage of encoding time compared to the scheme without any pruning. Each number in the table is an average over several target bitrate levels: 300, 600, 1000, 1500, 2000, 2500, and 3000 kbps. Note that the proposed method yields a smaller quality loss with shorter encoding time than in our previous work [74]. Our method does not outperform the state-of-the-art methods PRUNE_2D_FAST in terms of the average BD rate, but shows a gain in particular video sequences such as `bowing` (as shown in Table 5.4. Note that in [119], for each supported block size ($N \times N$, $N \times 2N$ and $2N \times N$, with $N \in \{4, 8, 16\}$), a specific neural network is required to obtain the scores, involving more than 5000 parameters to be learned in total. In contrast, our

approach only requires the weights $\mathbf{q}$ to be determined for each transform length, requiring $4 + 8 + 16 + 32 = 60$ parameters. With or without optimized weights, our model is more interpretable than the neural-network-based model, as has a significantly smaller number of parameters, whose meaning can be readily explained.

5.5 Summary

In this chapter, we explored discrete trigonometric transform (DTT) filtering approaches using sparse graph operators. First, we introduced fundamental graph operators associated to 8 DCTs and 8 DSTs by exploiting trigonometric properties of their transform bases. We also showed that these sparse operators can be extended to 2D separable transforms involving 1D DTTs. Considering a weighted setting for frequency response approximation, we proposed least squares and minimax approaches for both polynomial graph filter (PGF) and multivariate polynomial graph filter (MPGF) designs. We demonstrated through an experiment that PGF and MPGF designs would provide a speedup compared to traditional DTT filter implemented in transform domain. We also used MPGF to design a speedup technique for transform type selection in a video encoder, where a significant complexity reduction can be obtained.

Chapter 6

Irregularity-Aware GFT for Image and Video Coding

Mean square error (MSE) is commonly used as quality metric in many image and video coding standards. However, as it is well-known that MSE does not always reflect perceptual quality, it is important to incorporate a perceptually-driven metric into the coding optimization process. Based on such a metric, it would be possible to formulate a bit allocation problem with the goal of spending more bits on image regions that are perceptually more sensitive to quantization error. In the literature, this problem is typically addressed by designing perceptual quantization strategies. For example, JPEG quantization tables can be designed based on human visual system (HVS) [133], while JPEG-2000 adopts a visual masking technique [141] that exploits self-contrast masking and neighborhood masking, leading to adaptive quantization of wavelet coefficients without any overhead. Quantization parameter (QP) adjustment is a popular approach in video codecs such as HEVC [120], in which QP is changed per block or per coding unit.

In this chapter, we propose a novel approach based on designing transforms with the goal of optimizing a weighted mean square error (WMSE), which allows us to adapt the perceptual quality pixel-wise instead of block-wise. We make use irregularity-aware graph Fourier transforms (IAGFTs) [36], generalized GFTs where orthogonality is defined with respect to an inner product such that distance between a signal and a noisy version corresponds to a WMSE instead of the MSE. This leads to a generalized Parseval's Theorem, in which the quantization error energy in the IAGFT transform domain is the same as the pixel domain WMSE. Based on the IAGFT, we design an image coding framework, where perceptual quality is characterized by choosing suitable weights for the WMSE. Under this framework, the overall perceptual quality of an image can be enhanced by weighting different pixels differently based on their perceptual importance, while the quantization step size is fixed for the entire image. We consider a noise model, under which the WMSE weights are chosen to maximize the structural similarity (SSIM) [135]. Besides WMSE weights driven by SSIM, other perceptual weights may also be used

Work in this chapter has been published in [76].

to optimize other metrics. We demonstrate experimentally the benefits of our framework by modifying a JPEG encoder to incorporate these novel transforms, showing coding gains in terms of multi-scale SSIM [137].

In Section 6.1 we give a summary of IAGFT. In Section 6.2, we propose a weight design for IAGFT that favors improved SSIM. Some properties of IAGFT basis are discussed in Section 6.3. In Section 6.4, we demonstrate of perceptually driven IAGFT through experimental results. Finally we conclude this chapter in Section 6.5.

6.1 Irregularity-aware graph Fourier transform (IAGFT)

The IAGFT [36] is a generalization of the GFT, where the graph Fourier modes (i.e., GFT basis functions) are determined not only by the signal variation operator $\mathbf{L}$, but also by a positive definite matrix $\mathbf{Q}$ that leads to a $\mathbf{Q}-$inner product [36]: $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}} = \mathbf{x}^\top \mathbf{Q} \mathbf{y}$, and therefore to a new definition of orthogonality: $\mathbf{x}$ is orthogonal to $\mathbf{y}$ if and only if $\mathbf{x}^\top \mathbf{Q} \mathbf{y} = 0$. Typically, $\mathbf{Q}$ is chosen to be diagonal, so that the energy of signal $\mathbf{x}$ is a weighted sum of its squared components: $\|\mathbf{x}\|^2_{\mathbf{Q}} = \sum_{i \in \mathcal{V}} q_i |x_i|^2$, with $\mathbf{Q} = \mathrm{diag}(q_1, \ldots, q_n)$. The notion of generalized energy leads to a generalized GFT, i.e., the IAGFT:

Definition 4 (Generalized graph Fourier modes)**.** *Given the Hilbert space defined by the $\mathbf{Q}-$inner product and a graph variation operator $\mathbf{L}$, the set of $(\mathbf{L}, \mathbf{Q})-$graph Fourier modes is defined as the solution $\{\mathbf{u}_k\}_k$ to the following sequence of minimization problems: for increasing $K \in \{1, \ldots, N\}$,*

$$\underset{\mathbf{u}_K}{\textit{minimize}} \quad \mathbf{u}_K^\top \mathbf{L} \mathbf{u}_K \qquad \textit{subject to} \quad \mathbf{U}_K^\top \mathbf{Q} \mathbf{U}_K = \mathbf{I}, \tag{6.1}$$

where $\mathbf{U}_K = (\mathbf{u}_1, \ldots, \mathbf{u}_K)$.

Definition 5 (Irregularity-aware GFT)**.** *Let $\mathbf{U}$ be the matrix of $(\mathbf{L}, \mathbf{Q})-$graph Fourier modes, the $(\mathbf{L}, \mathbf{Q})-$GFT is $\mathbf{F} = \mathbf{U}^\top \mathbf{Q}$ and its inverse is $\mathbf{F}^{-1} = \mathbf{U}$.*

In fact, (6.1) can be written as a generalized Rayleigh quotient minimization, whose solution can be obtained efficiently through the generalized eigenvalue problem. Note that when $\mathbf{Q} = \mathbf{I}$, $\mathbf{F}$ reduces to conventional GFT as in Section 1.2.1. One key property of the IAGFT is the *generalized Parseval's theorem*:

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{Q}} = \langle \hat{\mathbf{x}}, \hat{\mathbf{y}} \rangle_{\mathbf{I}}, \tag{6.2}$$

with $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ being the $(\mathbf{L}, \mathbf{Q})-$GFT of $\mathbf{x}$ and $\mathbf{y}$, respectively. Depending on the application, various choices of $\mathbf{Q}$ may be used. Examples include diagonal matrices of node degrees and Voronoi cell areas (refer to [36] for details).

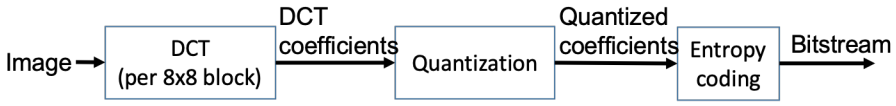


Figure 6.1: Flow diagrams of JPEG encoder.

6.2 Perceptual coding with Weighted MSE

We focus on the weighted mean square error (WMSE) as an image quality metric, where different pixels are associated with different weights. First, in Section 6.2.1, we design a transform coding scheme that optimizes the WMSE, and analyze its error in pixel domain. We focus on the choice of perceptual quality inspired weights in Section 6.2.2.

6.2.1 Transform Coding with IAGFT

We define the WMSE with weights $\mathbf{q} \in \mathbb{R}^n$ (or, in short, $\mathbf{q}$-MSE) between a distorted signal $\mathbf{z} \in \mathbb{R}^n$ and its reference signal $\mathbf{x} \in \mathbb{R}^n$ as

$$\mathrm{WMSE}(\mathbf{z}, \mathbf{x}, \mathbf{q}) := \frac{1}{n} \sum_{i=1}^{n} q_i (z_i - x_i)^2 = \frac{1}{n} \langle \mathbf{z} - \mathbf{x}, \mathbf{z} - \mathbf{x} \rangle_{\mathbf{Q}}, \tag{6.3}$$

where $\mathbf{Q} = \mathrm{diag}(\mathbf{q})$. When $\mathbf{q} = \mathbf{1}$, i.e., $\mathbf{Q} = \mathbf{I}$, the WMSE reduces to the conventional MSE. We note that the right hand side of (6.3) is a $\mathbf{Q}$-inner product, so the generalized Parseval's Theorem gives

$$\mathrm{WMSE}(\mathbf{z}, \mathbf{x}, \mathbf{q}) = \frac{1}{n} \langle \hat{\mathbf{z}} - \hat{\mathbf{x}}, \hat{\mathbf{z}} - \hat{\mathbf{x}} \rangle_{\mathbf{I}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{z}_i - \hat{x}_i)^2.$$

This means that *minimizing the* $\mathbf{q}$*-MSE is equivalent to minimizing the* ℓ_2 *error energy in the IAGFT domain.* Based on this fact, we propose an image coding scheme that integrates IAGFT into the JPEG framework. The diagram is shown in Figure 6.1, where the values in $\mathbf{Q}$ are quantized and transmitted as signaling overhead for the decoder to uniquely reconstruct the image. Further details for implementation will be described in Section 6.4.

Next we provide a pixel domain error analysis under uniform quantization noise assumption. Let $\boldsymbol{\varepsilon}_p$ and $\boldsymbol{\varepsilon}_t$ be the vectors of errors within a block in the pixel and IAGFT domain, respectively. Then, the variance of the i-th element in $\boldsymbol{\varepsilon}_p$ is

$$\mathbb{E}\left[\varepsilon_p(i)^2\right] = \mathbb{E}\left[(\mathbf{e}_i^\top \mathbf{U} \boldsymbol{\varepsilon}_t)^2\right] = \mathrm{tr}\left(\mathbf{U}^\top \mathbf{e}_i \mathbf{e}_i^\top \mathbf{U} \cdot \mathbb{E}\left[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_t^\top\right]\right),$$

where $\mathbf{e}_i$ is the i-th standard basis. Denote the quantization step size for the j-th transform coefficient as Δ_j and model the quantization noise with uniform distribution $\varepsilon_t(i) \sim \text{Unif}(-\Delta_i/2, \Delta_i/2)$. Thus, we have $\mathbb{E}\left[\boldsymbol{\varepsilon}_t \boldsymbol{\varepsilon}_t^\top\right] = \text{diag}(\Delta_1^2, \ldots, \Delta_n^2)/12$. When a uniform quantizer with $\Delta_i = \Delta$ is used for all i,

$$\mathbb{E}\left[\varepsilon_p(i)^2\right] = \frac{\Delta^2}{12}\text{tr}\left(\mathbf{U}^\top \mathbf{e}_i \mathbf{e}_i^\top \mathbf{U}\right) = \frac{\Delta^2}{12}\text{tr}\left(\mathbf{e}_i \mathbf{e}_i^\top \mathbf{Q}^{-1}\right) = \frac{\Delta^2}{12 q_i}, \tag{6.4}$$

where we have used $\mathbf{U}\mathbf{U}^\top = \mathbf{Q}^{-1}$, which follows from the fact that $\mathbf{U}^\top(\mathbf{Q}\mathbf{U}) = \mathbf{I} = (\mathbf{Q}\mathbf{U})\mathbf{U}^\top$. With (6.4), we know that the expected WMSE for this block is

$$\mathbb{E}\left[\text{WMSE}(\mathbf{z}, \mathbf{x}, \mathbf{q})\right] = \frac{1}{n}\sum_{i=1}^{n} q_i \left(\mathbb{E}\left[\varepsilon_p(i)^2\right]\right) = \frac{\Delta^2}{12},$$

which only depends on the quantization step.

Note that the scheme shown in Figure 6.1 can be viewed as a bit allocation method. When $q_j = 2q_i$ for pixels i and j within a block, the quantization error of IAGFT coefficients tends to contribute more error to pixel j than to pixel i in the pixel domain. Implicitly, this indicates that more bits are spent to accurately encode pixel j. On the other hand, if $\mathbf{Q}_\ell = 2\mathbf{Q}_k$ for blocks k and ℓ, we can show that the IAGFT coefficients will satisfy $\mathbf{x}_\ell = \sqrt{2}\mathbf{x}_k$, meaning that the encoder tends to use more bits for block ℓ than for block k.

6.2.2 SSIM-Driven Weights for WMSE

In this work, we adopt the structural similarity (SSIM) as the target metric for perceptual quality, and design a WMSE to optimize it[1]. SSIM is one of the most popular image quality assessment metrics, with many experiments demonstrating its better alignment with perceptual visual quality as compared to MSE [135]. For a distorted image $\mathbf{z}$ and its reference image $\mathbf{x}$, the definition of SSIM is

$$\begin{aligned} \text{SSIM}(\mathbf{x}, \mathbf{z}) &= \frac{1}{n}\sum_{i=1}^{n} \text{SSIM}(x_i, z_i), \\ \text{SSIM}(x_i, z_i) &= \frac{2\mu_{x_i}\mu_{z_i} + c_1}{\mu_{x_i}^2 + \mu_{z_i}^2 + c_1} \cdot \frac{2\sigma_{x_i z_i} + c_2}{\sigma_{x_i}^2 + \sigma_{z_i}^2 + c_2}, \end{aligned} \tag{6.5}$$

[1] In fact, our proposed method can be applied for any arbitrary WMSE. The application of this method based on other metrics such as Video Multimethod Assessment Fusion (VMAF) is considered for future work.

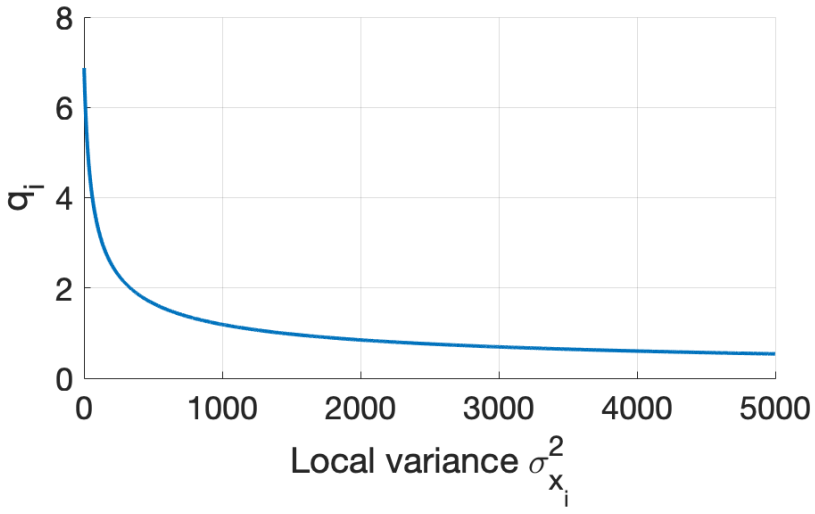


Figure 6.2: Values of q_i with respect to local variance, with $\Delta = 8$.

where μ_{x_i} and $\sigma^2_{x_i}$ are local mean and variance around pixel i and the summation is taken over all n pixels in the image.

We denote $\mathbf{z} = \mathbf{x} + \boldsymbol{\varepsilon}_p$, and assume that $\mathbf{x}$ and $\boldsymbol{\varepsilon}_t$ are independent. Based on the statistics of $\boldsymbol{\varepsilon}_p$ derived in Section 6.2.1, we have $\mu_{z_i} = \mu_{x_i}$, $\sigma^2_{x_i z_i} = \sigma^2_{x_i}$, and $\sigma^2_{z_i} = \sigma^2_{x_i} + \Delta^2/12q_i$. Thus, the local SSIM in (6.5) reduces to

$$\mathrm{SSIM}(x_i, z_i) = \frac{2\sigma^2_{x_i} + c_2}{2\sigma^2_{x_i} + c_2 + \Delta^2/(12q_i)} = \frac{q_i}{q_i + \gamma_i},.$$

where $\gamma_i = \Delta^2/12(2\sigma^2_{x_i} + c_2)$. To obtain $\mathbf{q}$ that maximizes the SSIM, we introduce an upper bound for $\sum_i q_i$ as a proxy of the bitrate constraint, and solve

$$\underset{\mathbf{q}}{\text{maximize}} \quad \frac{1}{n}\sum_{i=1}^{n} \frac{q_i}{q_i + \gamma_i} \qquad \text{subject to} \quad \sum_{i=1}^{n} q_i \le n. \tag{6.6}$$

It can be shown that this problem is convex in $\mathbf{q}$. Using the Lagrangian cost function and Karush-Kuhn-Tucker (KKT) conditions, we can obtain a closed-form solution:

$$q_i = \frac{(n + \sum_{i=1}^{n} \gamma_i)\sqrt{\gamma_i}}{\sum_{i=1}^{n} \sqrt{\gamma_i}} - \gamma_i. \tag{6.7}$$

While $\mathbf{q}$ is a high dimensional vector (with dimension n, the number of pixels in the image), (6.7) provides an efficient way to obtain the optimal solution, which only depends on the quantization step and local variance. The computation of all q_i can be

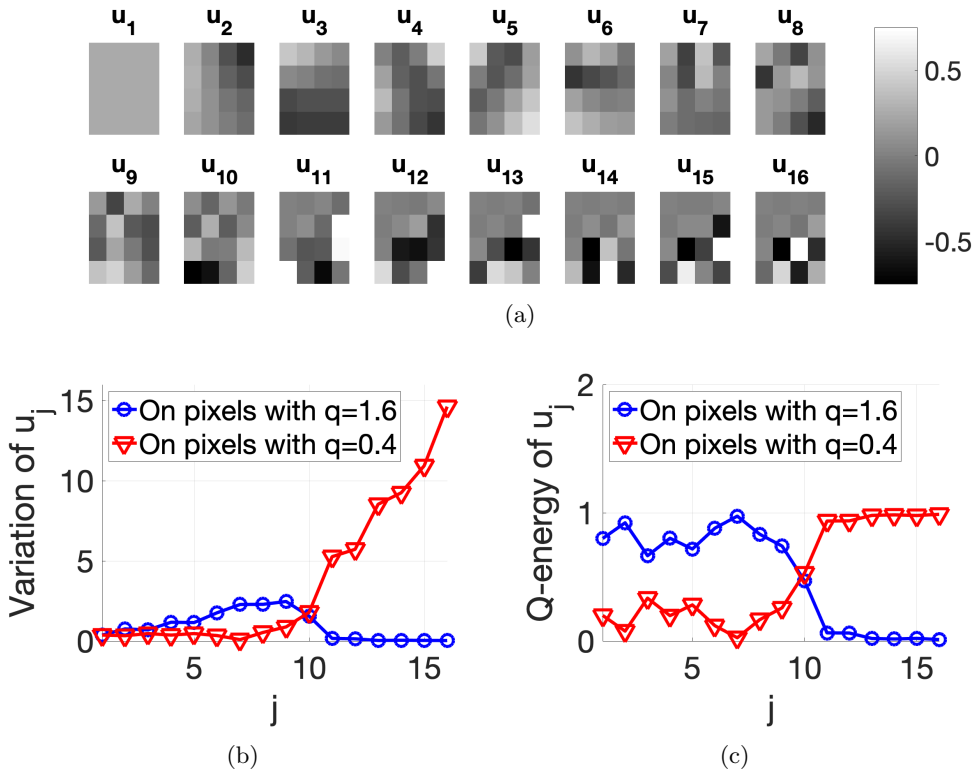


Figure 6.3: (a) Basis, (b) $\mathbf{Q}_0$-energy and (c) variations of $\mathbf{Q}_0$-IAGFT modes. We use j to denote indices of basis functions. In (b) and (c), two curves represent quantities for pixels with $q_i = 1.6$ and with $q_i = 0.4$, respectively.

carried out in $\mathcal{O}(n)$ time. Figure 6.2 shows the resulting of q_i with different local variance values and a fixed quantization step size. The fact that q_i decreases with respect to local variance means that larger weights are used for pixels in uniform or smooth regions of the image, which in turn results in higher quality in those regions.

6.3 IAGFT Transform Bases

In this section, we provide some remarks on IAGFT basis functions. First, we consider the case with $\mathbf{Q} = k\mathbf{I}$, where the IAGFT coefficients are $\sqrt{k}$ times the DCT coefficients. In this case, when we apply a $k\mathbf{I}$-IAGFT followed by a uniform quantization with step size Δ, it is equivalent to applying a DCT followed by a quantization with step size $\sqrt{k}\Delta$. Therefore, this special case reduces to a block-wise quantization step adjustment scheme, as in related work such as [48]. This means that our scheme can be viewed as a generalization of quantization step adjustment method, and it can adapt the quality per pixel, which is finer than a per block adaptation.

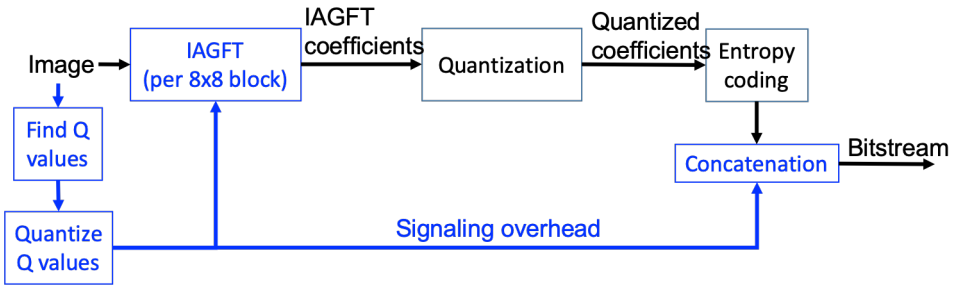


Figure 6.4: Flow diagrams of our proposed scheme. Blocks highlighted in blue are new components.

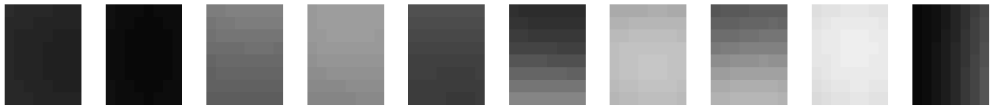


Figure 6.5: Vector quantization codewords of q_i for 8×8 blocks.

As a second example, we show the 2D $\mathbf{Q}_0$-IAGFT basis in Figure 6.3(a), where $\mathbf{Q}_0$ is the diagonal matrix associated to a 4×4 block with WMSE weights:

$$\begin{pmatrix} 1.6 & 1.6 & 1.6 & 1.6 \\ 1.6 & 1.6 & 1.6 & 0.4 \\ 1.6 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.4 \end{pmatrix}.$$

Note that the pixels in the top left corner have larger weights, while the weights sum to 16 as in the $\mathbf{I}$-IAGFT (i.e., DCT). In Figure 6.3(b)(c) we show the $\mathbf{Q}_0$-energy and variation of each basis function, within top-left regions ($q_i = 1.6$) and within bottom-right regions ($q_i = 0.4$). By definition of IAGFT (Definition 5), functions $\mathbf{u}_j$ with increasing j would correspond to lower to higher variations $\mathbf{u}_j^\top \mathbf{L} \mathbf{u}_j$, while having the same $\mathbf{Q}_0$-energy $\mathbf{u}_j^\top \mathbf{Q}_0 \mathbf{u}_j$. In Figure 6.3(b) we observe that those basis functions corresponding to low to medium frequencies (i.e., $\mathbf{u}_1$ to $\mathbf{u}_9$) have increasing variations for pixels in the top left region. In fact, the $\mathbf{Q}_0$-energy of $\mathbf{u}_1$ to $\mathbf{u}_9$ are highly concentrated to pixels with a larger q. On the other hand, other basis functions associated to higher frequencies ($\mathbf{u}_{11}$ to $\mathbf{u}_{16}$) are localized to the lower right corner, where $q_i = 0.4$. This means that in the pixel domain, the energy in the area with large q_i will be mostly captured by low frequency IAGFT coefficients. As a result, when the importance of pixel i is characterized with q_i, those IAGFT basis function with lower frequencies would tend to represent information on more important pixels.

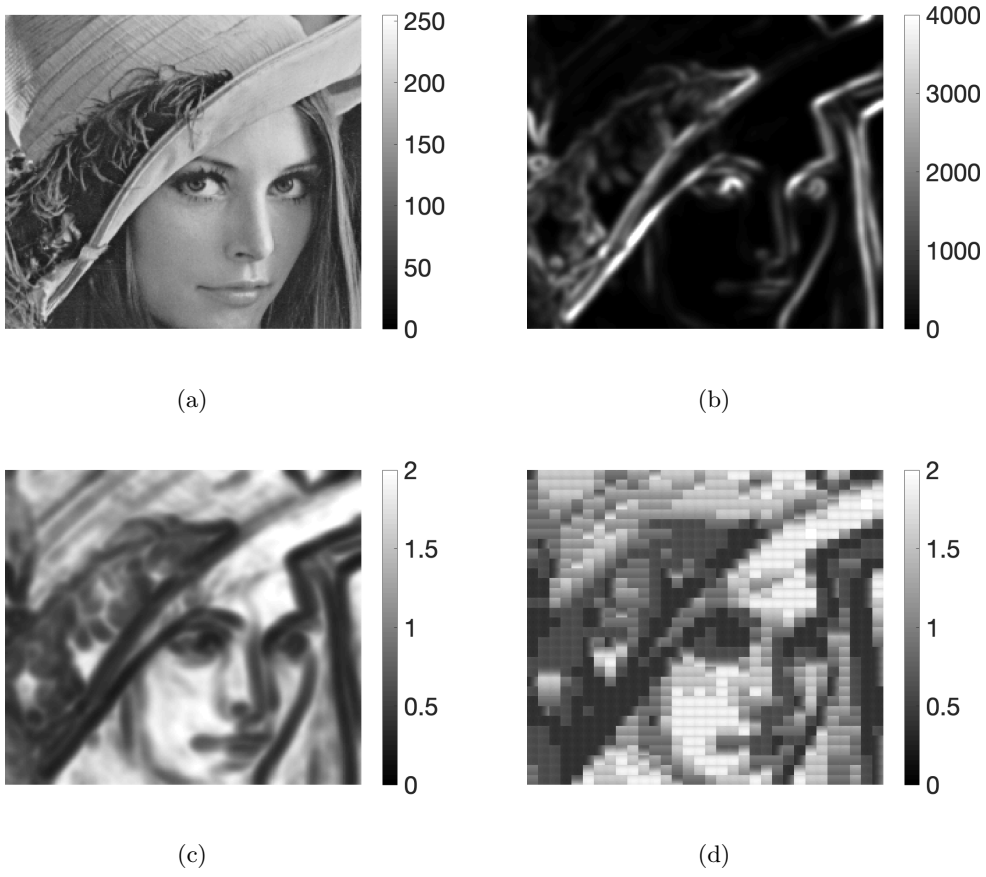
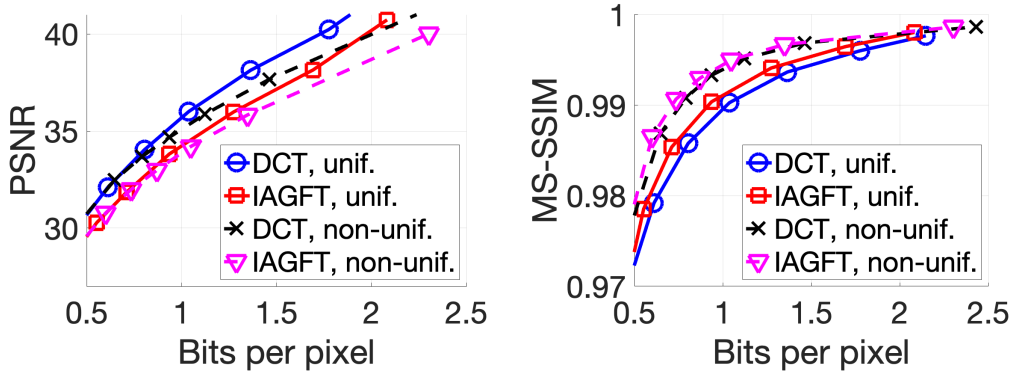


Figure 6.6: An example: (a) original image, (b) local variance map (c) q_i map, and (d) quantized q_i map with vector quantization.

6.4 Experiments

6.4.1 Image Coding Results on JPEG

To demonstrate the effectiveness of the proposed framework, we apply the coding scheme illustrated in Figure 6.4 in JPEG. Note that the non-uniform quantization table in JPEG standard was designed based on perceptual criteria for DCT coefficients. We propose a similar non-uniform quantization for IAGFTs as follows. For an IAGFT with basis functions $\mathbf{u}_k$, we find the unique representations in DCT domain, denoted as $\mathbf{u}_k = \sum_{i=1}^{n} \phi_{ki}\mathbf{v}_i$ with $\mathbf{v}_i$ being the i-th DCT basis vector. Then, we choose quantization step associated to $\mathbf{u}_k$ as a weighted mean: $\overline{\Delta}_k = \sum_{i=1}^{n} |\phi_{ki}|\Delta_i$, where Δ_i is the quantization step associated with $\mathbf{v}_i$. In this way, when $\mathbf{u}_k$ has low frequency in DCT domain, a small quantization step size will be used, and vice versa. The Laplacian of a uniform grid graph is used as variation operator to define the IAGFT. The weights $\mathbf{q}$ for WMSE are first



(a) (b)

Figure 6.7: RD curves for `Airplane` image in (a) PSNR and (b) MS-SSIM.

obtained from (6.7), then quantized using entropy-constrained vector quantization (VQ) [35] with 10 codewords trained from the 8×8 blocks of q_i values in `house` image. The resulting codewords are shown in Figure 6.5, and signaling overhead for each codeword is based on the selection frequency during VQ training. In particular, the number of bits for each codeword is $\lceil -p \log_2(p) \rceil$, where p is the empirical probability obtained for this codeword during the training procedure. For each 8×8 block of testing images, we quantize the corresponding $\mathbf{q}$ to the closest codeword, and apply the associated 8×8 non-separable IAGFT. We assume that transform bases and quantization tables for the IAGFTs represented by the codewords are embedded in the codec, so we do not require eigen-decomposition or side information for bases and quantizers. For illustration, Figure 6.6(b) shows the local variance map obtained as in SSIM formula (6.5), and Figure 6.6(c)(d) show that resulting q_i obtained from (6.7) and the quantized q_i with VQ, respectively.

The RD performances in terms of PSNR and multi-scale SSIM (MS-SSIM) [137] are shown in Figure 6.7. We observe that the proposed scheme leads to a loss in PSNR, while giving a compression gain in MS-SSIM. In Table 6.1, we show the BD rate performance in PSNR, SSIM, and MS-SSIM for several benchmark images. By comparing the "NU, DCT" and "NU, IAGFT" rows, we note that with non-uniform quantization, the IAGFT-based scheme yields a MS-SSIM gain compared to the DCT-based scheme (i.e., standard JPEG), even though (6.6) was derived under a uniform quantization assumption. One probable reason for a higher gain in MS-SSIM than in SSIM is that q_i's have been smoothed by the vector quantizer. Thus, the quantized q_i's are no longer optimal for SSIM. Alternatively, such a smoothing procedure may potentially favor the MS-SSIM metric, which takes into account the quality of smoothed and downscaled versions of

		Mandrill	Lena	Airplane	Sailboat
Uniform, IAGFT	PSNR	14.78%	17.64%	20.40%	17.69%
	SSIM	1.28%	2.18%	3.81%	1.64%
	MS-SSIM	-2.09%	-2.25%	-8.18%	-7.70%
Non-uniform, DCT	PSNR	8.42%	5.14%	8.90%	3.82%
	SSIM	-6.23%	-5.22%	-9.00%	-10.64%
	MS-SSIM	-27.82%	-17.05%	-17.76%	-30.27%
Non-uniform, IAGFT	PSNR	22.64%	17.96%	26.14%	14.36%
	SSIM	-2.05%	-1.99%	-2.82%	-6.19%
	MS-SSIM	-28.15%	-17.74%	-21.28%	-31.74%

Table 6.1: Bit rate comparison with respect to DCT-based scheme with uniform quantization table (i.e., "Uniform, DCT"). Negative numbers correspond to compression gains. Uniform and Non-uniform refer to uniform and non-uniform quantization tables, respectively.

the original output image. We also note that for this experiment, the side information accounts for 6% to 8% of the bit rates. Thus, further optimization for signaling overhead may lead to a coding gain in SSIM.

6.4.2 Subjective Comparison–An Example

Here, we show an example of output images using DCT-based and IAGFT-based schemes in Figure 6.8. Those image patches are chosen to have similar bitrates, but the IAGFT encoded image has a better quality in SSIM and MS-SSIM. We can compare these images in two different regions circled in blue (smooth region) and yellow (texture region) boxes. In the smooth region, the q_i values are larger, meaning that the IAGFT scheme spends more bits on this region and produces smaller error. Indeed, less blocking artifacts are observed in the blue region in the IAGFT encoded image compared to the same region in the DCT encoded image. On the contrary, q_i values in the texture region are smaller, so IAGFT produces a larger error in terms of the MSE, compared to the same region in the DCT encoded image. Perceptually, the reconstruction error in texture region would be less visible as that in a smooth region. Such a difference in perceptual sensitivity to distortion demonstrates how different weights q_i in different pixels lead to an improvement in human perceptual quality.

6.5 Conclusion

In this chapter, we consider weighted mean square error (WMSE) as an image quality metric, as a generalization of the widely used MSE. By selecting proper weights, WMSE offers a high flexibility and enables us to better characterize human perceptual quality

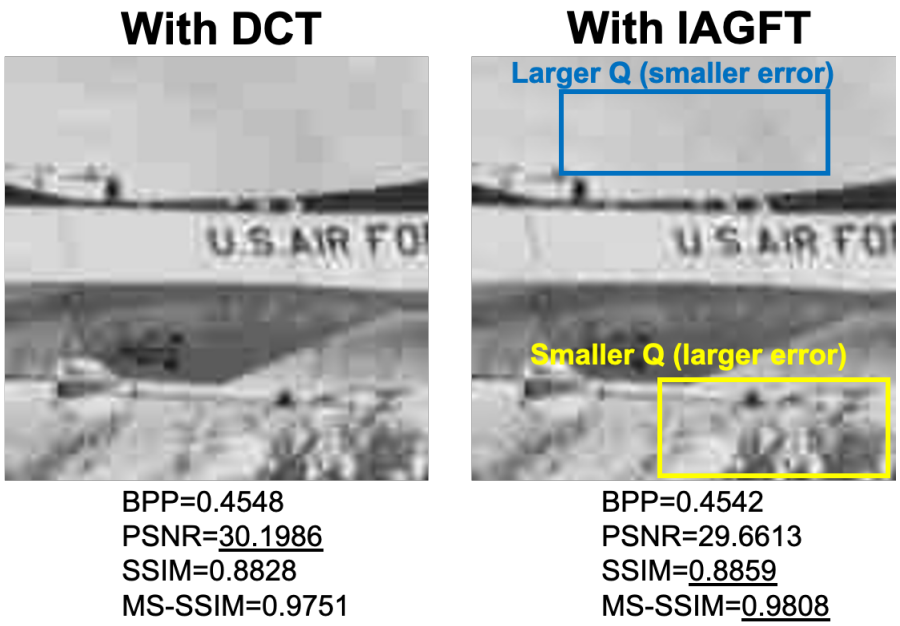


Figure 6.8: Encoded image patches of (a) DCT and (b) IAGFT.

than the MSE. In order to optimize WMSE-based image quality, we proposed a novel image coding scheme using irregularity-aware graph Fourier transform (IAGFT). Based on the generalized Parseval's theorem, we have shown the optimality in terms of WMSE when uniform quantization is performed in the IAGFT domain. We then design weights to maximize the structural similarity (SSIM), where the weights are determined by the local image variances and quantization step size. When integrated into JPEG standard, our method with the associated SSIM-driven WMSE can provide a compression gain in MS-SSIM. In the future, we will extend this method to schemes with non-uniform quantization and consider the integration into existing video codecs.

Chapter 7

Conclusion and Future Work

7.1 Summary of this Thesis

The primary goal of this thesis was to provide efficient graph-based approaches, including speedup techniques for the graph Fourier transform and efficient graph filtering. Among possible applications, we focused primarily on image and video compression, where graph Fourier transform (GFT) and its variations (such as lapped GFT and irregularity aware GFT) can be applied in transform coding.

Firstly, in Chapter 2, we explored fast GFTs with Haar units as components. In particular, algebraic conditions for a graph such that the Haar units emerge in its GFT were derived. Those conditions indicate that butterfly stages of Haar units arise when the graph have symmetry or bipartition properties. We then defined a notion of graph symmetry, such that a graph with this symmetry property is guaranteed to have a fast GFT algorithm. We presented an approach to derive divide-and-conquer fast GFT algorithm, and show a number of examples with potential fields of application. Our fast GFT approach is suited for graphs with regular or nearly regular topologies, uniformly weighted graphs, or graph that are symmetric by construction. Experimental results showed that the GFT run time can be reduced significantly using the proposed fast GFTs.

In Chapter 3 we applied efficient graph-based transforms to video coding in different aspects. We formulated a general convex problem for solving this graph learning problem. Under this framework, we considered the so-called symmetric line graph transforms (SLGT), which are 1D transforms with a butterfly stage. Then, we applied the learning method to non-separable transforms, i.e., GFTs on grid graphs, and then propose an approximate fast GFT solution when the butterfly stages are not known ahead of time. Experimental results showed that, by learning parameters of the symmetric line graph, we obtained useful SLGTs that give a compression gain for inter residual blocks. In addition, we can also obtain non-separable GFTs to approximate KLTs, which, when applied to intra residual block with some certain modes, can improve the compression efficiency.

In addition to block-based GFTs, we have also extended classical lapped transforms (transforms on overlapping blocks) to a graph-based setting in Chapter 4. Perfect reconstruction and orthogonality properties were re-examined for lapped transforms defined on graphs. With the focus on line graphs, we proposed the design of lapped graph Fourier transform, which leads to a nearly optimal transform coding gain, and can be applied to piecewise smooth image compression. We showed experimentally that the proposed lapped transform outperforms the DCT and the conventional lapped orthogonal transform.

We investigated in Chapter 5 DTT (DCT and DST) filters in a graph-based perspective. We showed that all types of DCTs and DSTs are associated to a family of sparse graph operators, which are DTT filters by themselves. Thus, those operators allow us to design efficient approximation of DTT filters with arbitrary frequency responses using 1) polynomial graph filter (PGF) and 2) multivariate polynomial graph filter (MPGF). We introduced design approaches of both PGF and MPGF with least squares and minimax criteria, to obtain efficient DTT filters. With experiments implemented in C, the derived DCT and DST filters yields a speedup in popular graph filters–Tikhonov filter, bandpass exponential filter, and ideal low-pass filter, when graph size is sufficiently large. Our method can also be applied to rate-distortion optimization for transform type search, yield a significant speed up in the AV1 codec.

Finally, in Chapter 6, we studied the application of irregularity aware GFT (IAGFT) for perceptual coding. In particular, we pointed out the fact the IAGFT has orthogonality property with respect to a weighted mean square error (WMSE), which can be used to characterize perceptual image quality. Given a certain WMSE, the generalized Parseval's theorem implies that minimum WMSE distortion can be achieved when uniform quantization is applied in the IAGFT domain. Based on these properties, we designed WMSE weights to align with the well-known structural similarity (SSIM) index for perceptual image quality. For validation, a coding scheme is built on top of the JPEG encoder, where IAGFT with the SSIM-driven weights is applied instead of the traditional DCT. Our preliminary results showed a promising coding gain in MS-SSIM.

7.2 Future Work

For future work, we consider the following issues or extensions:

1. *Lapped GFTs for more general graphs and applications.* In Section 4.2.2, we have demonstrated some results on piecewise smooth images. In fact, the definition of the lapped transforms on graphs can be applied to general graph topologies. Thus, we would like to explore a wider range of applications, and with different

types of graphs. For example, image filtering can be an application because conventional graph-based filtering cannot be easily applied without partitioning the image, especially when the number of pixels is large.

2. *Extensions for IAGFT in perceptual video coding.* With the generalized Parseval's theorem, we can design a variety of useful transforms based on different WMSEs. Choices of weights that lead to perceptual quality enhancement, or other video coding benefits, would be an interesting problem. For example, larger weights can be chosen for pixels that are used to predict many other pixels, to achieve higher prediction accuracy. Another extension would be exploring efficient methods to reduce signaling overhead. For example, weights for the WMSE may be derived from information that are available to both encoder and decoder, so that no extra bits will be required.

Appendix A

Proofs of Theorems and Lemmas

A.1 Proof of Theorem 2

In the following, we will repeatedly use (2.15) and (2.16). Also recall that in (2.11),

$$\mathbf{L}^{+} = \begin{pmatrix} \mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J} & \sqrt{2}\mathbf{L}_{XZ} \\ \sqrt{2}\mathbf{L}_{XZ}^{\top} & \mathbf{L}_{ZZ} \end{pmatrix}, \tag{A.1}$$

$$\mathbf{L}^{-} = \mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY}. \tag{A.2}$$

From the block partition structure (2.6), we also have

$$(\mathbf{L}_{XX})_{i,j} = l_{i,j}, \quad (\mathbf{L}_{XZ})_{i,j} = l_{i,p+j}, \quad (\mathbf{L}_{XY})_{i,j} = l_{i,n-p+j}, \quad (\mathbf{L}_{ZZ})_{i,j} = l_{p+i,p+j},$$
$$(\mathbf{L}_{YY})_{i,j} = l_{n-p+i,n-p+j}, \quad (\mathbf{J}\mathbf{L}_{XY})_{i,j} = l_{p+1-i,n-p+j}, \quad (\mathbf{L}_{XY}\mathbf{J})_{i,j} = l_{i,n+1-j}.$$

Such changes of indices will be used in the derivations below.

- **Edges of $\mathcal{G}^{+}$.** With $i, j \in \mathcal{V}^{+} = \{1, \ldots, n-p\}$ and $i \neq j$, we discuss three different cases separately, all based on (A.1). If $i, j \in \mathcal{V}_X = \{1, \ldots, p\}$, then

$$w_{i,j}^{+} = -\left(\mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J}\right)_{i,j} = -(l_{i,j} + l_{i,n+1-j}) = w_{i,j} + w_{i,n+1-j}.$$

  If $i \in \mathcal{V}_X = \{1, \ldots, p\}$ and $j \in \mathcal{V}_Z = \{p+1, \ldots, n-p\}$, then

$$w_{i,j}^{+} = -\left(\sqrt{2}\mathbf{L}_{XZ}\right)_{i,j-p} = -\sqrt{2}l_{i,j} = \sqrt{2}w_{i,j},$$

  and the same holds for $i \in \mathcal{V}_Z$ and $j \in \mathcal{V}_X$. If $i, j \in \mathcal{V}_Z = \{p+1, \ldots, n-p\}$, then

$$w_{i,j}^{+} = -\left(\mathbf{L}_{ZZ}\right)_{i-p,j-p} = -l_{i,j} = w_{i,j}.$$

- **Self-loops of $\mathcal{G}^+$.** To express s_i^+, we discuss the cases with $i \in \mathcal{V}_X$ and $i \in \mathcal{V}_Z$ separately. If $i \in \mathcal{V}_X = \{1, \dots, p\}$, then from (A.1) we have

$$\begin{aligned} s_i^+ &= \sum_{j=1}^{p} (\mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J})_{i,j} + \sum_{j=1}^{n-2p} \left(\sqrt{2}\mathbf{L}_{XZ}\right)_{i,j} = \sum_{j=1}^{p} (l_{i,j} + l_{i,n+1-j}) + \sqrt{2} \sum_{j=1}^{n-2p} l_{i,p+j} \\ &= \underbrace{s_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{i,j}}_{l_{i,i}} + \underbrace{\left(- \sum_{\substack{j=1 \\ j\neq i}}^{p} w_{i,j} \right)}_{l_{i,j} \text{ with } i\neq j} - \sum_{j=1}^{p} w_{i,n+1-j} - \sqrt{2} \sum_{j=1}^{n-2p} w_{i,p+j} \\ &= s_i - (\sqrt{2}-1) \sum_{j=p+1}^{n-p} w_{i,j}. \end{aligned}$$

  If $i \in \mathcal{V}_Z = \{p+1, \dots, n-p\}$, then (A.1) gives

$$\begin{aligned} s_i^+ &= \sum_{j=1}^{n-2p} (\mathbf{L}_{ZZ})_{i-p,j} + \sum_{j=1}^{p} \left(\sqrt{2}\mathbf{L}_{XZ}\right)_{i-p,j} = \sum_{j=1}^{n-2p} l_{i,p+j} + \sqrt{2} \sum_{j=1}^{p} l_{j,i} \\ &= \underbrace{s_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{i,j}}_{l_{i,i}} + \underbrace{\left(- \sum_{\substack{j=p+1 \\ j\neq i}}^{n-p} w_{i,j} \right)}_{l_{i,j} \text{ with } i\neq j} - \sqrt{2} \sum_{j=1}^{p} w_{i,j} \\ &= s_i + \sum_{j=1}^{p} w_{i,j} + \sum_{j=n-p+1}^{n} w_{i,j} - \sqrt{2} \sum_{j=1}^{p} w_{i,j} = s_i + (2-\sqrt{2}) \sum_{j=1}^{p} w_{i,j} \end{aligned}$$

- **Edges of $\mathcal{G}^-$.** With $i, j \in \mathcal{V}_Y = \{n-p+1, \dots, n\}$ and $i \neq j$, from (A.2) we have

$$w_{i,j}^- = -\left(\mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY}\right)_{i-n+p,j-n+p} = -(l_{i,j} - l_{n+1-i,j}) = w_{i,j} - w_{n+1-i,j}.$$

- **Self-loops of $\mathcal{G}^-$.** Here, we have $i \in \mathcal{V}_Y = \{n-p+1, \dots, n\}$, so, by (A.2),

$$\begin{aligned} s_i^- &= \sum_{j=1}^{p} \left(\mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY}\right)_{i-n+p,j} = \sum_{j=1}^{p} (l_{i,n-p+j} - l_{n+1-i,n-p+j}) \\ &= \underbrace{s_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{i,j}}_{l_{i,i}} + \underbrace{\left(- \sum_{\substack{j=n-p+1 \\ j\neq i}}^{n} w_{i,j} \right)}_{l_{i,j} \text{ with } i\neq j} + \sum_{j=1}^{p} w_{i,p+1-j} = s_i + 2\sum_{j=1}^{p} w_{i,j} + \sum_{j=p+1}^{n-p} w_{i,j}. \end{aligned}$$

The results derived above apply to the case when $\phi = (n, n-1, \ldots, 1)$. For any arbitrary ϕ, we can simply modify the sub-indices based on ϕ. Thus, the results above can be written as in Lemma 2.

A.2 Derivations for Sparse Operators of DST-IV, DST-VII, and DCT-V

A.2.1 Sparse DST-IV Operators

Recall the definition of DST-IV functions as in (1.11):

$$\phi_j(k) = v_j(k) = \sqrt{\frac{2}{N}} \sin \frac{(j-\frac{1}{2})(k-\frac{1}{2})\pi}{N}$$

As in Section 5.2.1, we can obtain

$$\begin{aligned} v_j(p-\ell) + v_j(p+\ell) &= \sqrt{\frac{2}{N}} \left[\sin \frac{(j-\frac{1}{2})(p-\ell-\frac{1}{2})\pi}{N} + \sin \frac{(j-\frac{1}{2})(p-\ell-\frac{1}{2})\pi}{N} \right] \\ &= 2\sqrt{\frac{2}{N}} \sin \frac{(j-\frac{1}{2})(p-\ell-\frac{1}{2})\pi}{N} \cos \frac{\ell(j-\frac{1}{2})\pi}{N} \\ &= \left(2\cos \frac{\ell(j-\frac{1}{2})\pi}{N} \right) v_j(p), \end{aligned}$$

where we have applied the trigonometric identity

$$\sin\alpha + \sin\beta = 2\sin\left(\frac{\alpha+\beta}{2}\right)\cos\left(\frac{\alpha-\beta}{2}\right). \tag{A.3}$$

By the left and right boundary condition of DST-IV, we have

$$v_j(p-\ell) = -v_j(-p+\ell+1), \quad v_j(p+\ell) = v_j(-p-\ell+2N+1).$$

which gives the following result:

Proposition 3. *For $\ell = 1, \ldots, N-1$, we define $\mathbf{Z}^{(\ell)}_{DST\text{-}IV}$ as a $N \times N$ matrix, whose p-th row has only two non-zero elements specified as follows:*

$$\left(\mathbf{Z}^{(\ell)}_{DST\text{-}IV}\right)_{p,q_1} = \begin{cases} 1 \text{ with } q_1 = p-\ell, & \text{if } p-\ell \geq 1 \\ -1 \text{ with } q_1 = -p+\ell+1, & \text{otherwise} \end{cases},$$

$$\left(\mathbf{Z}^{(\ell)}_{DST\text{-}IV}\right)_{p,q_2} = 1, \quad q_2 = \begin{cases} p+\ell, & \text{if } p+\ell \leq N \\ -p-\ell+2N+1, & \text{otherwise} \end{cases}$$

The corresponding eigenvalues are $\lambda_j = 2\cos\frac{\ell(j-\frac{1}{2})\pi}{N}$.

A.2.2 Sparse DST-VII Operators

Now, we consider the basis function of DST-VII:

$$\phi_j(k) = \frac{2}{\sqrt{2N+1}} \sin\frac{\left(j-\frac{1}{2}\right)k\pi}{N+\frac{1}{2}}.$$

Then, by (A.3) we have

$$\begin{aligned}
\phi_j(p-\ell)+\phi_j(p+\ell) &= \frac{2}{\sqrt{2N+1}}\left[\sin\frac{\left(j-\frac{1}{2}\right)(p-\ell)\pi}{N+\frac{1}{2}} + \sin\frac{\left(j-\frac{1}{2}\right)(p+\ell)\pi}{N+\frac{1}{2}}\right] \\
&= \frac{2}{\sqrt{2N+1}} 2\sin\frac{\left(j-\frac{1}{2}\right)p\pi}{N+\frac{1}{2}}\cos\frac{\ell\left(j-\frac{1}{2}\right)\pi}{N+\frac{1}{2}} \\
&= \left(2\cos\frac{\ell\left(j-\frac{1}{2}\right)\pi}{N+\frac{1}{2}}\right)\phi_j(p) \qquad \text{(A.4)}
\end{aligned}$$

The left boundary condition (i.e., $\phi_j(k) = -\phi_j(-k)$) of DST-VII corresponds to $\phi_j(p-\ell) = -\phi_j(-p+\ell)$. Together with the right boundary condition $\phi_j(p+\ell) = \phi_j(-p-\ell+2N+1)$, we have the following proposition.

Proposition 4. *For* $\ell = 1, \ldots, N-1$, *we define* $\mathbf{Z}^{(\ell)}_{DST\text{-}VII}$ *as a* $N \times N$ *matrix, whose* p*-th row has at most two non-zero elements specified as follows:*

$$\left(\mathbf{Z}^{(\ell)}_{DST\text{-}VII}\right)_{p,q_1} = \begin{cases} 1 \text{ with } q_1 = p-\ell, & \text{if } p > \ell \\ -1 \text{ with } q_1 = -p+\ell, & \text{if } p < \ell \end{cases},$$

$$\left(\mathbf{Z}^{(\ell)}_{DST\text{-}VII}\right)_{p,q_2} = 1, \quad q_2 = \begin{cases} p+\ell, & \text{if } p+\ell \le N \\ -p-\ell+2N+1, & \text{otherwise} \end{cases}$$

The corresponding eigenvalues are $\lambda_j = 2\cos\frac{\ell\left(j-\frac{1}{2}\right)\pi}{N+\frac{1}{2}}$.

In Proposition 4, note that the ℓ-th row has only one nonzero element because when $p = \ell$, $\phi_j(p-\ell) = 0$, and (A.4) reduces to

$$\phi_j(p+\ell) = \left(2\cos\frac{\ell\left(j-\frac{1}{2}\right)\pi}{N+\frac{1}{2}}\right)\phi_j(p).$$

A.2.3 Sparse DCT-V Operators

Here, ϕ_j are defined as DCT-V basis functions

$$\phi_j(k) = \frac{2}{\sqrt{2N-1}} c_j c_k \sin \frac{(j-1)(k-1)\pi}{N-\frac{1}{2}}.$$

Note that $c_k = 1/\sqrt{2}$ for $k = 1$ and 1 otherwise. For the trigonometric identity (5.7) to be applied, we introduce a scaling factor such that $b_k c_k = 1$ for all k:

$$b_k = \begin{cases} \sqrt{2}, & k = 1 \\ 1, & \text{otherwise} \end{cases}.$$

In this way, by (5.7) we have

$$\begin{aligned}
&b_{p-\ell} \cdot \phi_j(p-\ell) + b_{p+\ell} \cdot \phi_j(p+\ell) \\
&= \frac{2}{\sqrt{2N-1}} c_j \left[\cos \frac{(j-1)(p-\ell-1)\pi}{N-\frac{1}{2}} + \cos \frac{(j-1)(p+\ell-1)\pi}{N-\frac{1}{2}} \right] \\
&= \frac{2}{\sqrt{2N-1}} c_j 2 \cos \frac{(j-1)(p-1)\pi}{N-\frac{1}{2}} \cos \frac{\ell(j-1)\pi}{N-\frac{1}{2}} \\
&= \left(2 \cos \frac{\ell(j-1)\pi}{N-\frac{1}{2}} \right) b_p \phi_j(p),
\end{aligned}$$

so this eigenvalue equation can be written as

$$\frac{b_{p-\ell}}{b_p} \cdot \phi_j(p-\ell) + \frac{b_{p+\ell}}{b_p} \cdot \phi_j(p+\ell) = \left(2 \cos \frac{\ell(j-1)\pi}{N-\frac{1}{2}} \right) \phi_j(p). \tag{A.5}$$

The left boundary condition of DCT-V corresponds to $\phi_j(p-\ell) = \phi_j(-p+\ell+2)$, and the right boundary condition gives $\phi_j(p+\ell) = \phi_j(-p-\ell+2N+1)$. Thus, (A.5) yields the following proposition:

Proposition 5. *For $\ell = 1, \ldots, N-1$, we define $\mathbf{Z}^{(\ell)}_{DCT\text{-}V}$ as a $N \times N$ matrix, whose p-th row has at most two non-zero elements specified as follows:*

$$\left(\mathbf{Z}^{(\ell)}_{DCT\text{-}V}\right)_{p,q_1} = \begin{cases} \sqrt{2} \text{ with } q_1 = 1, & \text{if } p-\ell = 1 \\ 1 \text{ with } q_1 = p-\ell, & \text{if } p-\ell > 1 \\ 1 \text{ with } q_1 = -p+\ell+2, & \text{if } p-\ell \le 0, p \ne 1 \end{cases},$$

$$\left(\mathbf{Z}^{(\ell)}_{DCT\text{-}V}\right)_{p,q_2} = \begin{cases} \sqrt{2} \text{ with } q_2 = 1, & p = 1 \\ 1 \text{ with } q_2 = p+\ell, & p \ne 1, p+\ell \le N \\ 1 \text{ with } q_2 = -p-\ell+2N+1, & \text{otherwise} \end{cases}$$

The corresponding eigenvalues are $\lambda_j = 2\cos\frac{\ell(j-1)\pi}{N-\frac{1}{2}}$.

The values of $\sqrt{2}$ in Proposition 5 arise from $b_{p-\ell}/b_p$ and $b_{p+\ell}/b_p$ in the LHS of (A.5). In particular, when $p = \ell + 1$ we have $b_{p-\ell}/b_p = \sqrt{2}$ and $b_{p+\ell}/b_p = 1$, so (A.5) gives

$$\sqrt{2}\phi_j(p-\ell) + \phi_j(p+\ell) = \left(2\cos\frac{\ell(j-1)\pi}{N-\frac{1}{2}}\right)\phi_j(p).$$

When $p = 1$, $b_{p-\ell}/b_p = b_{p+\ell}/b_p = 1/\sqrt{2}$. In addition, by the left boundary condition, $\phi_j(p-\ell) = \phi_j(p+\ell)$, so (A.5) reduces to

$$\sqrt{2}\phi_j(p+\ell) = \left(2\cos\frac{\ell(j-1)\pi}{N-\frac{1}{2}}\right)\phi_j(p), \quad \text{for } p = 1.$$

meaning that the first row of $\mathbf{Z}^{(\ell)}_{\text{DCT-V}}$ has one nonzero element only.

A.3 Proof of Theorem 3

Denote the objective function of the CGL estimation problem (3.4) as $\mathcal{J}(\mathbf{L})$. It have been shown in [25] that

$$\mathcal{J}(\mathbf{L}) := -\log|\mathbf{L}|_\dagger + \text{trace}(\mathbf{LS}) + \alpha\|\mathbf{L}\|_{1,\text{off}} = -\log|\mathbf{L}|_\dagger + \text{trace}(\mathbf{LK}), \tag{A.6}$$

where $\mathbf{K} = \mathbf{S} + \alpha(\mathbf{I} - \mathbf{1}\mathbf{1}^\top)$. The derivation from the function in (3.4) to (A.6) is based on the facts that $\|\mathbf{L}\|_{1,\text{off}} = \text{trace}(\mathbf{L}(\mathbf{I} - \mathbf{1}\mathbf{1}^\top))$ when $\mathbf{L}$ is a CGL.

First, we consider the oriented incidence matrix of a graph: $\mathbf{\Xi} = (\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_m) \in \mathbb{R}^{n\times m}$. Each of its columns $\boldsymbol{\xi}_j$ has two nonzero elements, 1 and -1, representing an edge ε_j connecting nodes s_j and t_j:

$$\boldsymbol{\xi}_j = \mathbf{e}_{s_j} - \mathbf{e}_{t_j} \text{ for } \varepsilon_j = (s_j, t_j) \in \mathcal{E},$$

where $\mathbf{e}_i$ is the i-th vector of the standard basis, that is, $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0$, $j \neq i$. We can express $\mathbf{L}$ in terms of the edge weights and $\mathbf{\Xi}$:

$$\mathbf{L} = \sum_{j=1}^{m} w_{s_j,t_j}\boldsymbol{\xi}_j\boldsymbol{\xi}_j^\top = \mathbf{\Xi}\cdot\text{diag}(w_{s_1,t_1}, \ldots, w_{s_m,t_m})\cdot\mathbf{\Xi}^\top, \tag{A.7}$$

where $\text{diag}(w_{s_1,t_1}, \ldots, w_{s_m,t_m})$ is the $m \times m$ diagonal matrix formed by w_{s_i,t_i}. Note that, as ε_j is undirected, either $\boldsymbol{\xi}_j = \mathbf{e}_{s_j} - \mathbf{e}_{t_j}$ or $\boldsymbol{\xi}_j = \mathbf{e}_{t_j} - \mathbf{e}_{s_j}$ is allowed ((A.7) holds either way, and the choice does not affect the following results). For simplicity, we define

$u_j := w_{s_j,t_j}$ to represent weights with a single subscript, and also define a weight vector as $\mathbf{u} = (u_1, \ldots, u_m)^\top$ for compact notation.

Next, we note that $|\mathbf{L}|_\dagger$ can be simplified using Kirchhoff's matrix tree theorem:

Theorem 4 (Matrix Tree Theorem [56])**.** *For a CGL* $\mathbf{L}$ *of a connected graph* $\mathcal{G}$*, we have*

$$|\mathbf{L}|_\dagger = n \sum_{\mathcal{T} \subseteq \mathcal{P}} \prod_{\varepsilon \in \mathcal{T}} w_\varepsilon,$$

where $\mathcal{P}$ *is the set of all spanning trees of* $\mathcal{G}$ *and* w_ε *is the weight of edge* ε*.*

By the assumption that $\mathcal{G}$ is a tree, the only spanning tree of $\mathcal{G}$ is itself. Thus, Theorem 4 implies that

$$|\mathbf{L}|_\dagger = n \prod_{j=1}^{m} u_j. \tag{A.8}$$

Now, we can reduce (A.6) to

$$\begin{aligned}
\mathcal{J}(\mathbf{u}) &= -\log\left(n \prod_{j=1}^{m} u_j\right) + \operatorname{trace}\left(\sum_{j=1}^{m} u_j \boldsymbol{\xi}_j \boldsymbol{\xi}_j^\top \mathbf{K}\right) \\
&= -\log(n) - \sum_{j=1}^{m} \log(u_j) + \sum_{j=1}^{m} u_j (\boldsymbol{\xi}_j^\top \mathbf{K} \boldsymbol{\xi}_j),
\end{aligned}$$

which is a convex function in $\mathbf{u}$. By setting its derivative with respect to u_j to zero, we obtain the optimal weights $u_j = 1/\left(\boldsymbol{\xi}_j^\top \mathbf{K} \boldsymbol{\xi}_j\right)$. Note that, with the expression $\mathbf{K} = \mathbf{S} + \alpha(\mathbf{I} - \mathbf{1}\mathbf{1}^\top)$ and $\mathbf{S} = \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\top / N$, the weights can be expressed in terms of $\mathbf{x}_i$:

$$\begin{aligned}
w_{s,t} &= \left[(\mathbf{e}_s - \mathbf{e}_t)^\top \mathbf{K} (\mathbf{e}_s - \mathbf{e}_t)\right]^{-1} = (k_{s,s} + k_{t,t} - 2k_{s,t})^{-1} \\
&= \left[\underbrace{\frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{x}_i(s) - \mathbf{x}_i(t)\right)^2}_{:=\delta_{s,t}} + 2\alpha\right]^{-1}, \quad \text{for } (s,t) \in \mathcal{E},
\end{aligned}$$

It is clear that this value is always positive given $\alpha > 0$, so $\mathbf{u} \geq \mathbf{0}$ is satisfied, meaning that the derived form is indeed the closed form solution of (3.4). Note that the quantity $\delta_{s,t}$ is the mean-square-difference between the s-th and the t-th elements over all realizations $\mathbf{x}_i$. □

Appendix B

Search of Involutions in General Graphs

In this appendix, we introduce two algorithms for searching valid involutions in a general graph. The purpose of these algorithms is to identify symmetry properties of a given graph that lead to fast graph Fourier transform (GFT) implementations discussed in Chapter 2. As in Section 1.2, we denote a graph as $\mathcal{G}$ with vertex set $\mathcal{V}$ of n nodes, edge set $\mathcal{E}$, and weighted adjacency matrix $\mathbf{W}$. Weighted self-loops are allowed in the graphs. Degrees of graph nodes are denoted as d_1, d_2, ..., d_n. Recall Definition 1 that a permutation on the set $\mathcal{V}$ is called an involution if $\phi(\phi(i)) = i$ for all $i \in \mathcal{V}$. For $\mathcal{V} = \{1, \ldots, n\}$, we denote an involution $\phi : \mathcal{V} \to \mathcal{V}$ as $\phi = (\phi(1), \phi(2), \ldots, \phi(n))$. As defined in Definition 2, a graph $\mathcal{G}$ is called ϕ-symmetric if $w_{i,j} = w_{\phi(i),\phi(j)}$ for all $i \in \mathcal{V}$, $j \in \mathcal{V}$. We say ϕ is a *valid* involution for graph $\mathcal{G}$ if $\mathcal{G}$ is ϕ-symmetric.

Essentially, finding all valid involutions of a graph is a combinatorial problem, since the number of all involutions $T(n)$, also known as the telephone number (2.19), grows asymptotically faster than polynomials in n [58] ($T(n)$ with n from 1 to 12 are shown in Table B). An exhaustive search of valid involutions among $T(n)$ of possible candidates, as described as in Algorithm 2, has an exponential (or higher) complexity.

However, given the knowledge of the graph structure and associated properties (e.g., when the graph is a tree), we can exploit useful strategies to prune the brute force algorithm, or even design an algorithm of polynomial complexity. In what follows, we will present two methods in Sections B.1 and B.2, respectively. The first method takes advantage of the degree list, and prune the exhaustive search based on the degrees of the nodes. The intuition behind it is that two nodes that are paired by an involution ϕ in a ϕ-symmetric graph must have the same degree. This provides a criterion for pruning involutions that are not valid during the search. The second approach is designed particularly for trees, which are special cases where $\mathcal{O}(n \log n)$ complexity (or $\mathcal{O}(n)$ complexity, when a $\mathcal{O}(n)$ sort is used) can be achieved. In this method, we consider a property that involutions of a tree graph can be characterized by pair(s) of identical

Work in this appendix is included in a technical report that can be found in [69].

Algorithm 2 Exhaustive search of graph symmetry

procedure EXHAUSTIVE($\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$)
  $\Pi \leftarrow \emptyset$
  $\mathcal{T} \leftarrow$ set of all involutions on $\mathcal{V}$
  while $\phi \in \mathcal{T}$ **do** ▷ Loop over all involutions
    if $w_{i,j} = w_{\phi(i),\phi(j)}$ for all $i, j \in \mathcal{V}$ **then**
      $\Pi \leftarrow \Pi \cup \{\phi\}$
    end if
  end while
  return Π ▷ Set of all valid involutions
end procedure

Table B.1: Telephone numbers $T(n)$ with n from 1 to 12

n	1	2	3	4	5	6	7	8	9	10	11	12
$T(n)$	1	2	4	10	26	76	232	764	2620	9496	35696	140152

tree branches whose roots are either common or adjacency. This property enables us to search involutions by comparing out-going branches at each tree node, leading to a linear complexity.

B.1 Pruning based on Degree Lists

From Definition 2, we obtain a necessary condition for nodes i and $\phi(i)$ being paired: nodes i and $\phi(i)$ have the same *weighted degree* (sum of weights on edges adjacent to a node). This is given by

$$d_i = \sum_{(i,j)\in\mathcal{E}} w_{i,j} = \sum_{(i,j)\in\mathcal{E}} w_{\phi(i),\phi(j)} = \sum_{(\phi(i),\phi(j))\in\mathcal{E}} w_{\phi(i),\phi(j)} = d_{\phi(i)}. \tag{B.1}$$

In addition, we can also consider the *unweighted degree* (number of neighbors of a node). We denote this number of neighbors for node i as f_i. Similar to (B.1), we have

$$f_i = \sum_{(i,j)\in\mathcal{E}} \mathbb{1}(w_{i,j}) = \sum_{(i,j)\in\mathcal{E}} \mathbb{1}(w_{\phi(i),\phi(j)}) = \sum_{(\phi(i),\phi(j))\in\mathcal{E}} \mathbb{1}(w_{\phi(i),\phi(j)}) = f_{\phi(i)}, \tag{B.2}$$

where $\mathbb{1}$ is the indicator function with $\mathbb{1}(w) = 1$ if $w \neq 0$ and $\mathbb{1}(w) = 0$ otherwise.

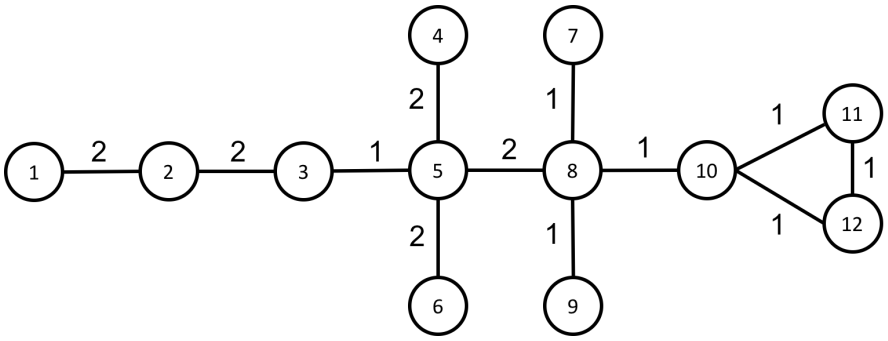
Based on these two conditions (B.1), (B.2), we obtain a useful pruning rule for the exhaustive search: *only search those involutions in which all pairs of nodes have the same degrees and numbers of neighbors.* To apply this criterion, we first partition the vertex set $\mathcal{V}$ into subsets $\{\mathcal{V}_{d,f}\}$, each of which contains nodes that share the common weighted

Algorithm 3 Truncated search of graph symmetry based on degree lists

procedure TRUNCATED($\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$)
  Compute the weighted degree list $\mathbf{d} = (d_1, \ldots, d_n)^\top$ with $d_i = \sum_{j=1}^n w_{i,j}$
  Compute the unweighted degree list $\mathbf{f} = (f_1, \ldots, f_n)^\top$ with $f_i = \sum_{j=1}^n \mathbb{1}(w_{i,j})$
  $\mathcal{S} \leftarrow$ GetTruncatedList($\mathbf{d}, \mathbf{f}$)
  $\Pi \leftarrow \emptyset$
  while $\phi \in \mathcal{S}$ **do** ▷ Loop over all involutions that are not pruned
    if $w_{i,j} = w_{\phi(i),\phi(j)}$ for all $i, j \in \mathcal{V}$ **then**
      $\Pi \leftarrow \Pi \cup \{\phi\}$
    end if
  end while
  return Π ▷ Set of all valid involutions
end procedure
procedure GETTRUNCATEDLIST($\mathbf{d}, \mathbf{f}$)
  $\mathcal{V}_{d,f} \leftarrow \emptyset$ for all d and f
  for $i = 1, \ldots, n$ **do**
    $\mathcal{V}_{d_i,f_i} \leftarrow \mathcal{V}_{d_i,f_i} \cup \{i\}$
  end for
  for $\mathcal{V}_{d,f} \neq \emptyset$ **do** ▷ Loop over all $\mathcal{V}_{d,f}$
    $\mathcal{S}_{d,f} \leftarrow$set of all involutions on $\mathcal{V}_{d,f}$
  end for
  $\mathcal{S} \leftarrow \bigotimes_{d,f} \mathcal{S}_{d,f}$ ▷ Direct product of all involution sets
  return $\mathcal{S}$ ▷ Set of all valid involutions after pruning
end procedure

and unweighted degrees (d and f). Then, for each vertex subset $\mathcal{V}_{d,f}$, we list all $T(|\mathcal{V}_{d,f}|)$ possible involutions on it. Finally, we search all combinations of the listed involutions across all vertex subsets. Note that, the number of such involutions is $\Pi_{d,f} T(|\mathcal{V}_{d,f}|)$, which is typically significantly smaller than $T(n)$ for graphs that are not regular (node degrees are not all equal).

An example is shown in Figure B.1. We note that the graph nodes have weighted degrees $d \in \{1, 2, 3, 4, 5, 7\}$ and unweighted degrees $f \in \{1, 2, 3, 4\}$. As in Figure B.1(c), there are 7 combinations of (d, f), so the vertex set is partitioned into 7 subsets with sizes 2, 3, 2, 1, 1, 1, 1, and 1. For the node in each class with size 1, all valid involutions must map it to itself, which reduces the dimension of the search. On the other hand, those classes $\mathcal{V}_{1,1}$, $\mathcal{V}_{2,1}$, and $\mathcal{V}_{2,2}$ have 2, 3, and 2 elements, respectively, so there are $T(2) = 2$, $T(3) = 4$, and $T(2) = 2$ candidates of involutions to be searched. As a result, the number of total involutions to be searched is reduced from $T(12) = 140152$ to $T(2)^2 T(3) = 2 \times 2 \times 4 = 16$.



(a) Graph

d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
2	4	3	2	7	2	1	5	1	3	2	2

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
1	2	2	1	4	1	1	4	1	3	2	2

(b) Degree lists

Involutions on $\mathcal{V}_{1,1} = \{7, 9\}$
$(d = 1, f = 1)$

$\phi(7)$	$\phi(9)$
7	9
9	7

Involutions on $\mathcal{V}_{2,1} = \{1, 4, 6\}$
$(d = 2, f = 1)$

$\phi(1)$	$\phi(4)$	$\phi(6)$
1	4	6
1	6	4
4	1	6
6	4	1

Involutions on $\mathcal{V}_{2,2} = \{11, 12\}$
$(d = 2, f = 2)$

$\phi(11)$	$\phi(12)$
11	12
12	11

Involutions on $\mathcal{V}_{3,2} = \{3\}$
$(d = 3, f = 2)$

$\phi(3)$
3

Involutions on $\mathcal{V}_{3,3} = \{10\}$
$(d = 3, f = 3)$

$\phi(10)$
10

Involutions on $\mathcal{V}_{4,2} = \{2\}$
$(d = 4, f = 2)$

$\phi(2)$
2

Involutions on $\mathcal{V}_{5,4} = \{8\}$
$(d = 5, f = 4)$

$\phi(8)$
8

Involutions on $\mathcal{V}_{7,4} = \{5\}$
$(d = 7, f = 4)$

$\phi(5)$
5

(c) Tables of involutions

Figure B.1: An example of pruning based on degree lists. (a) An example graph, (b) its weighted and unweighted degree lists, and (d) its vertex partitions and involutions on them.

To summarize, we show the procedures based on this pruning criterion in Algorithm 3, whose MATLAB implementation can be found in [69].

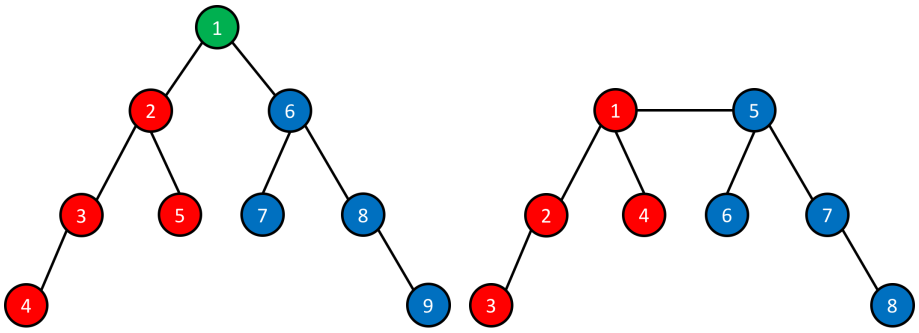


Figure B.2: Symmetry of tree characterized by identical branches. (a) Two branches with a common root. (b) Two branches with adjacent roots. Their associated involutions are $\phi_a = (1, 6, 8, 9, 7, 2, 5, 3, 4)$ and $\phi_b = (5, 7, 8, 6, 1, 4, 2, 3)$.

B.1.1 Complexity Analysis

Let the number of nodes be n and the number of edges be m. We note that, for a given involution ϕ, checking whether it is a valid involution (lines 5 and 6 in Algorithm 2) requires $\mathcal{O}(m)$ time. This means that Algorithm 2 takes an overall complexity of $\mathcal{O}(mT(n))$. To analyze the complexity of Algorithm 3, we break it into several parts:

- Retrieval of the degree lists (lines 2 and 3). This can be done by accumulating all edge weights, so the complexity is $\mathcal{O}(m)$ of each of $\mathbf{d}$ and $\mathbf{f}$.
- Obtaining the truncated list of involutions (lines 10-17). The algorithm requires scanning through all nodes (lines 12-13), visiting all possible involutions for each $\mathcal{V}_{d,f}$ (lines 14-15), and applying the direct product of all sets of involutions (line 16). Let the number of partitions be q and the sizes of partitions be k_1, k_2, ..., k_q. Then, these three procedures take $\mathcal{O}(n)$, $\mathcal{O}(\sum_{i=1}^{q} T(k_i))$, and $\mathcal{O}(\Pi_{i=1}^{q} T(k_i))$, respectively.
- Searching over the truncated set of involutions (lines 6-8). This takes $\mathcal{O}(m\Pi_{i=1}^{q} T(k_i))$.

Combining all the complexities above, we obtain an overall complexity of $\mathcal{O}(m\Pi_{i=1}^{q} T(k_i))$. This reduces the complexity of Algorithm 2 by a factor of $T(n)/\Pi_{i=1}^{q} T(k_i)$, where $n = \sum_{i=1}^{q} k_i$.

B.2 Searching of Identical Tree Branches

In this section, a polynomial time ($\mathcal{O}(n \log n)$, or $\mathcal{O}(n)$ with optimization) algorithm is provided for the search of involution on trees. This algorithm arises from the fact that a symmetry in a tree can be uniquely characterizedby identical branches of the tree whose roots are common (as in Figure B.2(a)) or adjacent (as in Figure B.2(b)). In what follows, we refer to such branches as *adjacent identical branches* (AIB). The main algorithm is based on the fact that AIBs can be identified efficiently through a bottom-up traversal in a tree.

This approach is motivated by related work in [13, 30], where the so-called *subtree repeats* can be found in linear time. The subtree repeat problem and our problem are similar, but different in two aspects. First, we do not allow the subtrees to overlap. Second, from the condition of graph symmetry, we require that the roots of the target subtrees to be adjacent. Thus, the algorithm we design can be regarded as a simplified version of the forward/non-overlapping stage in [30], and has the same time and space complexity.

First, we show that if a tree $\mathcal{G}$ is ϕ-symmetry, then this symmetry can be characterized by a pair, or several pairs of adjacent identical branches in $\mathcal{G}$:

Lemma 8. *Let ϕ be a non-trivial involution (ϕ ($\phi(i) \neq i$ for some i) and let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ be a ϕ-symmetric tree. Then, the tree nodes can be partitioned into $\mathcal{V} = \mathcal{V}_X \cup \mathcal{V}_Y \cup \mathcal{V}_Z$ such that $\phi(i) \in \mathcal{V}_Y$ if $i \in \mathcal{V}_X$ and $\phi(j) = j$ if $j \in \mathcal{V}_Z$, and there is at most one edge $(k, l) \in \mathcal{E}$ with $k \in \mathcal{V}_X$, $l \in \mathcal{V}_Y$.*

From this lemma, we know that involution ϕ is associated to a pair of branches (sub-trees) whose vertex sets are $\mathcal{V}_X$ and $\mathcal{V}_Y$, respectively.

Proof. We define $\mathcal{V}_Z = \{i \in \mathcal{V} | \phi(i) = i\} \neq \mathcal{V}$, and discuss two cases: $\mathcal{V}_Z \neq \emptyset$ and $\mathcal{V}_Z = \emptyset$.

If $\mathcal{V}_Z$ is non-empty, since ϕ is non-trivial, there must be a (possibly non-unique) node $k_1 \in \mathcal{V}_Z$ connected to two nodes that are not in $\mathcal{V}_Z$, denoted as i_1 and $j_1 = \phi(i_1)$. First, we start with $\mathcal{V}_X = \{i_1\}$ and $\mathcal{V}_Y = \{j_1\}$. Then, we apply a depth-first search (DFS) to traverse the branch rooted by i_1 toward its other neighbors than k_1, and include those nodes being visited into $\mathcal{V}_X$. Note that, from the ϕ-symmetry, every step of the DFS corresponds to a traversal step from j_1 towards its other neighbors than k_1. In this way, we can obtain an identical branch rooted by j_1, and include all nodes in this branch into $\mathcal{V}_Y$. If there exists another node $k_2 \in \mathcal{V}_Z$ other than k_1 connected to two nodes i_2 and $j_2 = \phi(i_2)$, we apply the same traversal procedure again and append $\mathcal{V}_X$ and $\mathcal{V}_Y$ in the same way. This procedure can be repeated until all nodes are classified into $\mathcal{V}_X$, $\mathcal{V}_Y$, and

$\mathcal{V}_Z$. The resulting $\mathcal{V}_X$ and $\mathcal{V}_Y$ will end up containing one or several identical branches of the original tree.

If $\mathcal{V}_Z$ is empty, then by connectivity of the tree graph, there must be an edge connecting two nodes paired by ϕ, i and $\phi(i)$. Then, we apply a DFS to traverse the branch rooted by i toward its other neighbor than $\phi(i)$, and include all visited nodes into $\mathcal{V}_X$. Similarly, the ϕ-symmetry also gives an identical branch rooted by $\phi(i)$. We include nodes in this branch into $\mathcal{V}_Y$. The connectivity of the graph implies that all nodes are included in these two indentical branches. □

Note that the converse of this theorem is straightforward: when there are two identical branches whose roots are common or adjacent, we can identify an involution ϕ that pairs the nodes of the branches such that the graph is ϕ-symmetric.

B.2.1 Algorithm

Since tree symmetry can be characterized by adjacent identical branches, we can build branch descriptors for all branches in the tree, and compare branches that are joined or adjacent. Note that, to apply this method, the tree needs to be rooted, i.e., has a root so that there is a hierarchy with different levels of nodes.

The reasonable choice of root would be a node in the *center* of the tree. The center of a graph is the set of vertices v where the greatest distance $d(u, v)$ to other vertices v is minimal. In fact, if $(z_1, \ldots, z_L)$ is a longest path of the graph (whose length is called the *diameter* of the graph), then the center node(s) of this path must belong to the center of the graph. An important consequence of this property is given as follows.

Lemma 9. *A node in the center of a tree cannot be an internal node of a branch that has an adjacent identical branch.*

Proof. If otherwise, joining these two branches leads to a path whose length is greater than the diameter, yielding a contradiction. □

To find the center of a tree, we can apply breadth-first search (BFS) twice to obtain the longest path in the tree, and identify the nodes in the center of the longest path. An algorithm for obtaining the center is shown in Algorithm 4. Then, with Lemma 9, we design a bottom-up branch matching algorithm on a tree rooted with its center, which is summarized in Algorithm 5.

As a demonstration of this algorithm, an example with graph in Figure B.3 is provided in Table B.2, where we show the iterations in reverse topological order of nodes (from bottom to top of the tree). Note that in iteration 5, an involution associated to the two AIBs is $\phi_1 = (1, 2, 3, 5, 4, 6, 7, \ldots, 13)$ (pairing of nodes 4 and 5), while the involution

Algorithm 4 Finding the center of a tree

procedure TreeCenter($\mathcal{G}$) ▷ $\mathcal{G}$ is a tree
  Pick any node $v_1 \in \mathcal{V}$
  Run BFS on $\mathcal{G}$ from v_1, and obtain the furthest node v_2
  Run BFS on $\mathcal{G}$ from v_2, and obtain the furthest node v_3
  Retrieve the v_2-v_3 path $(z_1 = v_2, z_2, \ldots, z_L = v_3)$ from the previous BFS solution
  ▷ This path is the diameter of the tree
  if $L = 2n + 1$ is odd **then**
    return $\{z_n\}$ ▷ The tree has one center
  end if
  if $L = 2n$ is even **then**
    return $\{z_n, z_{n+1}\}$ ▷ The tree has two centers
  end if
end procedure

Algorithm 5 Finding valid involutions in a tree

procedure FindInvolutionInTree($\mathcal{G}$)
  Take a node $v \in \mathcal{C}$=TreeCenter($\mathcal{G}$)
  Run BFS from v, and obtain the node list with topogolical order $(u_1, \ldots, u_n)$
  $\mathcal{I} \leftarrow \emptyset$ ▷ List of valid involutions
  for $i = n, \ldots, 1$ **do** ▷ Build branch descriptor bottom-up
    if u_i has no childs **then**
      $\mathcal{S}_i \leftarrow$'' ▷ Empty descriptor
    end if
    if u_i has childs **then**
      $(t_{i,1}, \ldots, t_{i,c_i}) \leftarrow$list of childs, sorted w.r.t. their descriptors.
      If any two branch descriptors are equal, append $\mathcal{I}$ with the involution characterized by the two identical branches
      $\mathcal{S}_i \leftarrow$'$(w_{u_i,t_{i,1}}\mathcal{S}_{t_{i,1}})(w_{u_i,t_{i,2}}\mathcal{S}_{t_{i,2}})\ldots(w_{u_i,t_{i,c_i}}\mathcal{S}_{t_{i,c_i}})$' ▷ Build descriptor by concatenating the weights and the descriptors of the sorted child branches
    end if
  end for
  if $|\mathcal{C}| = 2$ **then** ▷ The center has two nodes
    if Two branches rooted by the two center nodes are identical **then**
      Append $\mathcal{I}$ with the involution characterized by these two branches
    end if
  end if
  return $\mathcal{I}$
end procedure

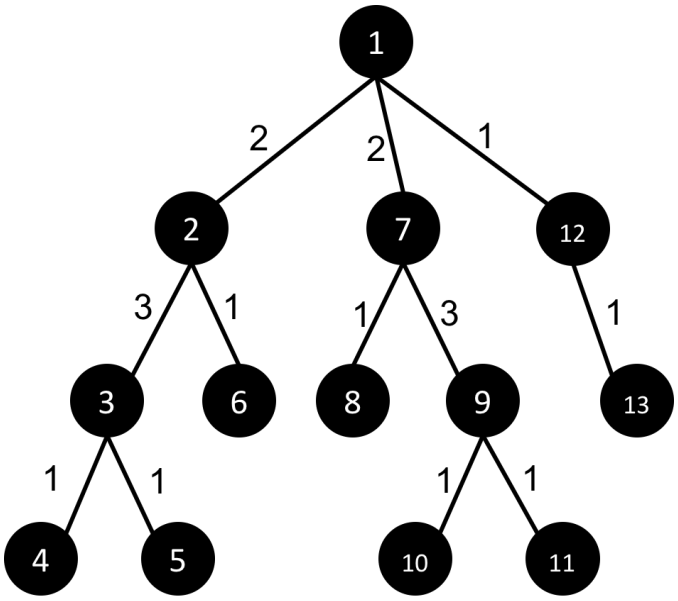


Figure B.3: An example graph for demonstration of Algorithm 5. The center of this graph is $\{1\}$. Note that a topological order given by BFS from node 1 would be $(1, 2, 7, 12, 3, 6, 8, 9, 13, 4, 5, 10, 11)$.

found in iteration 8 is $\phi_2 = (1, \ldots, 8, 9, 11, 10, 12, 13)$ (pairing of nodes 10 and 11). In the last iteration, the involution corresponding to the largest AIBs is

$$\phi_3 = (1, 7, 9, 10, 11, 8, 2, 6, 3, 4, 5, 12, 13),$$

that pairs nodes 2, 3, 4, 5, and 6 with nodes 7, 9, 10, 11, and 8, respectively. Note that, when each descriptor is built, the sorting of branch descriptors from the lower level will fix the order, so two AIBs must have identical branch descriptors.

B.2.2 Complexity Analysis

The algorithm consists of the following steps:

- Finding the center (line 2). This requires two BFS's and a traversal of a path. For a tree, the number of edges is $m = n - 1$, so the overall complexity of these operations on a tree is $\mathcal{O}(m + n) = \mathcal{O}(n)$.
- Finding a topological order using a BFS (line 3). The complexity of this step is $\mathcal{O}(n)$.
- Building descriptors for all nodes (line 5-11). Here, it take n iterations to loop over all tree nodes. Let p_i be the number of childs of node u_i. In iteration i, sorting the

Table B.2: Demonstration of Algorithm 5.

Iteration	Node	Branch descriptor	Comment
1	4	$\mathcal{S}_4 =$‘’	
2	5	$\mathcal{S}_5 =$‘’	
3	10	$\mathcal{S}_{10} =$‘’	
4	11	$\mathcal{S}_{11} =$‘’	
5	3	$\mathcal{S}_3 = (w_{3,4}\mathcal{S}_4)(w_{3,5}\mathcal{S}_5)$ =‘(1)(1)’	AIB found
6	6	$\mathcal{S}_6 =$‘’	
7	8	$\mathcal{S}_8 =$‘’	
8	9	$\mathcal{S}_9 = (w_{9,11}\mathcal{S}_{11})(w_{9,10}\mathcal{S}_{10})$ =‘(1)(1)’	AIB found
9	13	$\mathcal{S}_{13} =$‘’	
10	2	$\mathcal{S}_2 = (w_{2,6}\mathcal{S}_6)(w_{2,3}\mathcal{S}_3)$ =‘(1)(3((1)(1)))’	
11	7	$\mathcal{S}_7 = (w_{7,8}\mathcal{S}_8)(w_{7,9}\mathcal{S}_9)$ =‘(1)(3((1)(1)))’	
12	12	$\mathcal{S}_{12} =$‘(1)’	
13	1	$\mathcal{S}_1 = (w_{1,12}\mathcal{S}_{12})(w_{1,2}\mathcal{S}_2)(w_{1,7}\mathcal{S}_7)$ =‘(1(1))(2(1)(3(1)(1)))(2(1)(3(1)(1)))’	AIB found

descriptors takes $\mathcal{O}(p_i \log(p_i))$, and concatenating them takes $\mathcal{O}(p_i)$. The overall complexity of this loop is

$$\begin{aligned}
\sum_{i=1}^{n} p_i \log(p_i) &= (n-1)\left[\sum_{i=1}^{n} \frac{p_i}{n-1}\log(p_i)\right] \\
&= (n-1)\left[\sum_{i=1}^{n} \frac{p_i}{n-1}\log\left(\frac{p_i}{n-1}\right) + \sum_{i=1}^{n}\frac{p_i}{n-1}\log(n-1)\right] \\
&= (n-1)\left[-H_p + \frac{\sum_{i=1}^{n} p_i}{n-1}\log(n-1)\right] \\
&\leq (n-1)\log(n-1) \qquad \text{(B.3)}
\end{aligned}$$

where the equation $\sum_{i=1}^{n} p_i = n-1$ has been applied, and $H_p \geq 0$ because it is an entropy quantity. From (B.3), we obtain a complexity of $\mathcal{O}(n \log n)$.

- Check complete symmetry (line 13-14). This requires a comparison of two descriptors, and takes $\mathcal{O}(n)$ time.

In fact, as mentioned in [30], sorting of descriptors can be implemented by radix sort, which requires only $\mathcal{O}(p_i)$, due to the finite range of strings to be sorted. As a result, the overall complexity is $\mathcal{O}(n \log n)$ with an $\mathcal{O}(n \log n)$ sort, or $\mathcal{O}(n)$ with a $\mathcal{O}(n)$ sort.

B.3 Conclusion

In this appendix, we discuss two methods for searching a valid involution ϕ in a graph for graph symmetry. The first method is based on the fact that two symmetric nodes must have the same weighted and unweighted degrees. The second approach takes advantage of the sparsity of tree graphs, in which case the graph symmetry can be characterized by adjacent identical branches. We show the algorithms of both methods as well as examples for illustration. The complexities of both methods are analyzed. The MATLAB implementations can be found in the companion github repository [69].

Appendix C

Characterization of Sparse Laplacians with a Common GFT

This section includes remarks on the retrieval of sparse operators for general GFTs beyond DCT and DST.

The characterization of all Laplacians that share a common GFT has been studied in the context of graph topology identification and graph diffusion process inference [18,92,108]. In particular, it has been shown in [92] that the set of *normalized* Laplacian matrices having a fixed GFT can be characterized by a convex polytope. Following a similar proof, we briefly present the counterpart result for *unnormalized* generalized Laplacians, where self-loops are allowed:

Theorem 5. *The set of Laplacian matrices with a fixed GFT can be characterized by a convex polyhedral cone in the space of eigenvalues* $(\lambda_1, \ldots, \lambda_N)$.

Proof: For a given GFT $\mathbf{\Phi}$, let the eigenvalues λ_j of the Laplacian be variables. By definition of the Laplacian (1.4), we can see that

$$\mathbf{L} = \mathbf{\Phi} \cdot \mathrm{diag}(\lambda_1, \ldots, \lambda_N) \cdot \mathbf{\Phi}^\top = \sum_{k=1}^{N} \lambda_k \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top \tag{C.1}$$

is a valid Laplacian matrix if $l_{ij} \leq 0$ (non-negative edge weights), $l_{ii} \geq \sum_{j=1, j\neq i}^{N} l_{ij}$ (non-negative self-loop weights), and $\lambda_k \geq 0$ for all k (non-negative graph frequencies). With the expression (C.1) we have $l_{ij} = \sum_{k=1}^{N} \lambda_k \phi_k(i)\phi_k(j)$, and thus the Laplacian conditions can be expressed in terms of λ_j's:

$$\begin{aligned}
&\sum_{k=1}^{N} \lambda_k \phi_k(i)\phi_k(j) \leq 0, \quad \text{for } i \neq j, \\
&\sum_{k=1}^{N} \lambda_k \phi_k(i)^2 \geq \sum_{\substack{j=1 \\ j\neq i}}^{N} \lambda_k \phi_k(i)\phi_k(j), \quad \text{for } i = 1, \ldots, N, \\
&\lambda_k \geq 0, \quad k = 1, \ldots, N.
\end{aligned} \tag{C.2}$$

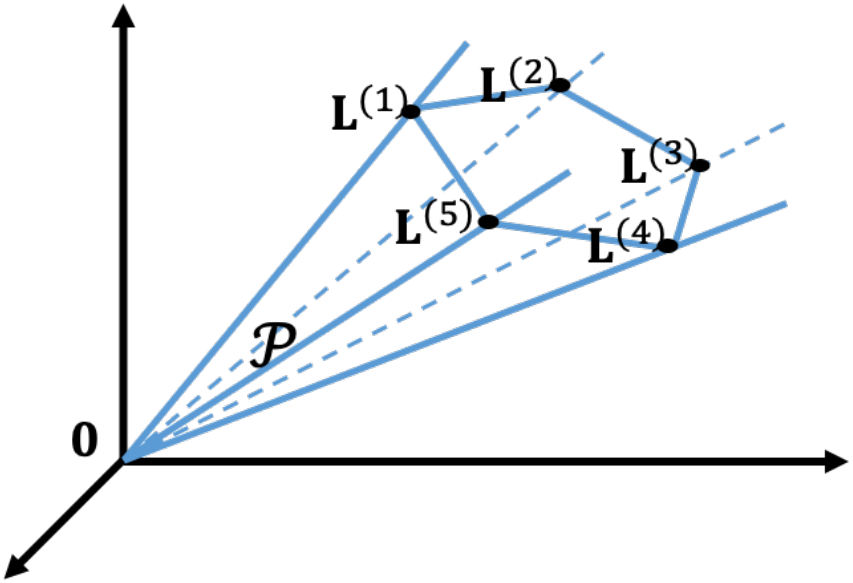


Figure C.1: An illustrative example of a polyhedral cone in $\mathbb{R}^3$ with a vertex at $\mathbf{0}$ and 5 edges. Any element of the cone can be represented as $\sum_{m=1}^{5} a_m \mathbf{L}^{(m)}$ with non-negative a_m.

These constraints on $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_N)^\top$ are all linear, so the feasible set for $\boldsymbol{\lambda} \in \mathbb{R}^N$ is a convex polyhedron. We denote this polyhedron by $\mathcal{P}$, and highlight some properties as follows:

- $\mathcal{P}$ is non-empty: it is clear that $\boldsymbol{\lambda} = \mathbf{1}$ gives $\mathbf{L} = \mathbf{I}$, which is a trivial, but valid, Laplacian.
- $\boldsymbol{\lambda} = \mathbf{0}$ is the only vertex of $\mathcal{P}$: when $\lambda_j = 0$ for all j, equality is met for all constraints in (C.2). This means that all hyperplanes that define $\mathcal{P}$ intersect at a common point $\mathbf{0}$, which further implies that $\mathcal{P}$ does not have other vertices than $\mathbf{0}$.

From those facts above, we conclude that $\mathcal{P}$ is a non-empty convex polyhedral cone. □

For illustration purpose, we can visualize the structure of a 3-dimensional polyhedral cone with 5 edges in Figure C.1. Notably, any element in $\mathcal{P}$ can be expressed by a conical combination (linear combination with non-negative coefficients) of elements on the edges of $\mathcal{P}$, as illustrated in Figure C.1. In particular, let $\mathcal{P}$ have M edges, and let $\mathbf{L}^{(1)}, \ldots, \mathbf{L}^{(M)}$ be points on different edges, then any element $\mathbf{Q} \in \mathcal{P}$ can be represented as

$$\mathbf{Q} = \sum_{m=1}^{M} a_m \mathbf{L}^{(m)}, \quad a_m \geq 0.$$

The fact that Laplacians have non-positive off-diagonal entries implies that the $\mathbf{L}^{(m)}$'s are the most sparse Laplacians. This can be seen by noting that a conical combination of two Laplacians must have more non-zero off-diagonal elements than the two individual Laplacians do.

Since sparse Laplacians are characterized by edges of a polyhedral cone, we can choose sparse filters in (5.1) as those Laplacians: $\mathcal{Z} = \{\mathbf{L}^{(k)}\}_k$. The retrieval of those matrices would require an algorithm that enumerates the vertices and edges given the description of a polyhedron. A popular algorithm for this problem is the so-called reverse search [3], which has a complexity $\mathcal{O}(rdv)$, where r is the number of linear constraints in $\mathbb{R}^d$, and v is the number of target vertices. In (C.2), $d = N$ and $m = (N^2 + 3N)/2$, so the complexity reduces to $\mathcal{O}(N^3 v)$. In practice, the vertex enumeration problem is in general an NP-hard problem since the number of vertices v can be a combinatorial number: $\binom{r}{d}$. For the purpose of efficient graph filter design, a truncated version of the algorithm [3] may be applied to obtain a few instead of all vertices. The study of such a truncated algorithm will be left for our future work.

Appendix D

Construction of Sparse Operators in Symmetric Graphs

This section shows a construction of sparse operator for graphs with certain symmetry properties. In Chapter 2, we highlighted that a GFT has a butterfly stage for fast implementation if the associated graph demonstrates a symmetry property based on involution. Here, we show how a sparse operator can be constructed for a φ-symmetric graph $\mathcal{G}$.

Lemma 10. *Given a φ-symmetric graph $\mathcal{G}$ with Laplacian $\mathbf{L}$, we can construct a graph $\overline{\mathcal{G}_\varphi}$ by connecting nodes i and j with edge weight 1 for all node pairs (i,j) with $\varphi(i) = j, i \neq j$. In this way, the Laplacian $\overline{\mathbf{L}_\varphi}$ of $\overline{\mathcal{G}_\varphi}$ commutes with $\mathbf{L}$.*

Proof: We note that, for $i \in \mathcal{V}$, we either have $\varphi(i) = j \neq i$ with $\varphi(j) = i$ or $\varphi(i) = i$. Without loss of generality, we order the graph vertices such that $\varphi(i) = N + 1 - i$ for $i = 1, \ldots, k$ and $\varphi(i) = i$ for $i = k+1, \ldots, N+1-k$. With this vertex order, we express $\mathbf{L}$ in terms of block matrix components,

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} & \mathbf{L}_{13} \\ \mathbf{L}_{12}^\top & \mathbf{L}_{22} & \mathbf{L}_{23} \\ \mathbf{L}_{13}^\top & \mathbf{L}_{23}^\top & \mathbf{L}_{33} \end{pmatrix},$$

where $\mathbf{L}_{11}, \mathbf{L}_{33} \in \mathbb{R}^{k \times k}$ and $\mathbf{L}_{22} \in \mathbb{R}^{(N-2k)\times(N-2k)}$. By φ-symmetry, the block components of $\mathbf{L}$ satisfy Lemma 4:

$$\mathbf{L}_{13} = \mathbf{J}\mathbf{L}_{13}^\top\mathbf{J}, \quad \mathbf{L}_{33} = \mathbf{J}\mathbf{L}_{11}\mathbf{J}, \quad \mathbf{L}_{23} = \mathbf{L}_{12}^\top\mathbf{J}. \tag{D.1}$$

We can also see that the Laplacian constructed from Lemma 10, with the same node ordering defined as above, is

$$\overline{\mathbf{L}_\varphi} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{J} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\mathbf{J} & \mathbf{0} & \mathbf{I} \end{pmatrix}.$$

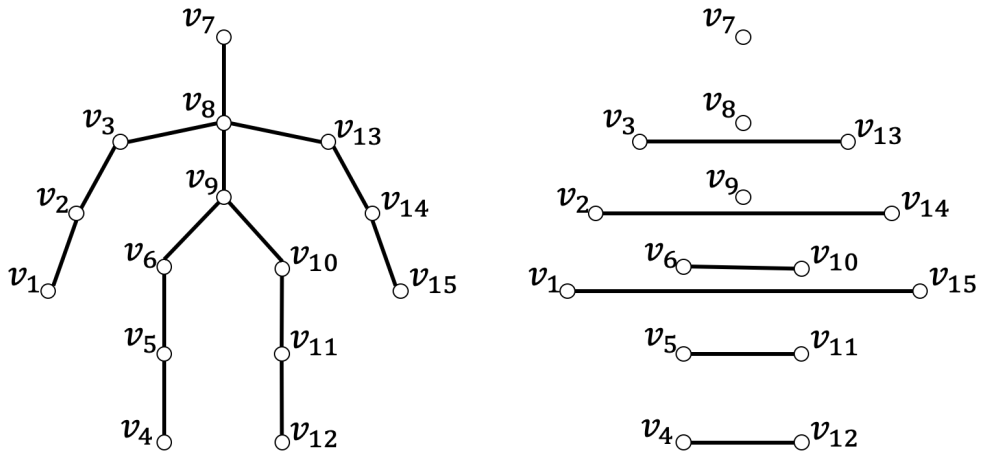


Figure D.1: An illustrative example for graph operator construction based on graph symmetry. (a) The 15-node human skeletal graph $\mathcal{G}$. (b) The graph $\overline{\mathcal{G}_\varphi}$ associated to an alternative sparse operator by construction. All edge weights are 1.

Then, using (D.1), we can easily verify that

$$\mathbf{L}\overline{\mathbf{L}_\varphi} = \begin{pmatrix} \mathbf{L}_{11} - \mathbf{L}_{13}\mathbf{J} & \mathbf{0} & -\mathbf{L}_{11}\mathbf{J} + \mathbf{L}_{13} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{L}_{13}^\top - \mathbf{J}\mathbf{L}_{11} & \mathbf{0} & -\mathbf{L}_{13}^\top\mathbf{J} + \mathbf{J}\mathbf{L}_{11}\mathbf{J} \end{pmatrix} = \overline{\mathbf{L}_\varphi}\mathbf{L},$$

which concludes the proof. □

We demonstrate an example for the construction of $\overline{\mathcal{G}_\varphi}$, in Figure D.1. Figure D.1(a) shows a 15-node human skeletal graph $\mathcal{G}$ [54]. A left-to-right symmetry can be observed in $\mathcal{G}$, which induces an involution φ with $\varphi(i) = i$ for $i = 7, 8, 9$ and $\varphi(i) = 16 - i$ otherwise. With the construction in Lemma 10, we obtain a graph $\overline{\mathcal{G}_\varphi}$ as in Figure D.1(b) by connecting all pairs of symmetric nodes in Figure D.1(a). We denote $\mathbf{Z}^{(1)} = \mathbf{L}$ and $\mathbf{Z}^{(2)} = \overline{\mathbf{L}_\varphi}$ the Laplacians of $\mathcal{G}$ and $\overline{\mathcal{G}_\varphi}$, respectively, and $\mathbf{\Psi} = (\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_{15})$ the GFT matrix of $\mathbf{L}$ with basis functions in increasing order of eigenvalues. Thus, we have

$$\mathbf{Z}^{(2)} = \mathbf{\Psi} \cdot \mathrm{diag}(\boldsymbol{\lambda}^{(2)}) \cdot \mathbf{\Psi}^\top, \quad \boldsymbol{\lambda}^{(2)} = (0, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0)^\top.$$

Since $\mathbf{Z}^{(2)}$ has only two distinct eigenvalues with high multiplicities, every polynomial of $\mathbf{Z}^{(2)}$ also has two distinct eigenvalues only, which poses a limitation for graph filter design. However, an MPGF with both $\mathbf{Z}^{(1)}$ and $\mathbf{Z}^{(2)}$ still provides more degrees of freedom compared to a PGF with a single operator.

Reference List

[1] A. Anis, A. Gadde, and A. Ortega, "Efficient sampling set selection for bandlimited graph signals using graph spectral proxies," *IEEE Transactions on Signal Processing*, vol. 64, no. 14, pp. 3775–3789, July 2016.

[2] A. Arrufat, P. Philippe, and O. Déforges, "Non-separable mode dependent transforms for intra coding in HEVC," in *IEEE Visual Communications and Image Processing Conference*, Dec. 2014, pp. 61–64.

[3] D. Avis and F. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete & Computational Geometry*, vol. 8, pp. 295–313, Sep. 1992.

[4] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," *Proceedings of the ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting*, 01 2001.

[5] A. Cantoni and P. Butler, "Eigenvalues and eigenvectors of symmetric centrosymmetric matrices," *Linear Algebra and its Applications*, vol. 13, no. 3, pp. 275–288, 1976.

[6] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett, and J. Kovačević, "Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring," *IEEE Transactions on Signal Processing*, vol. 62, no. 11, pp. 2879–2893, June 2014.

[7] S. Chen, A. Sandryhaila, J. M. F. Moura, and J. Kovačević, "Signal denoising on graphs via graph filtering," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2014, pp. 872–876.

[8] W.-H. Chen and S. C. Fralick, "Image enhancement using cosine transform filtering," in *Image Sci. Math. Symp.*, Nov 1976.

[9] W.-H. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Transactions on Communications*, vol. 25, no. 9, pp. 1004–1009, Sep. 1977.

[10] Y. Chen, D. Mukherjee, J. Han, A. Grange, Y. Xu, S. Parker, C. Chen, H. Su, U. Joshi, C.-H. Chiang, and et al., "An overview of coding tools in AV1: the first video codec from the alliance for open media," *APSIPA Transactions on Signal and Information Processing*, vol. 9, p. e6, 2020.

[11] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng, "Graph spectral image processing," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 907–930, May 2018.

[12] B. Chitprasert and K. R. Rao, "Discrete cosine transform filtering," *Signal Processing*, vol. 19, no. 3, pp. 233–245, 1990.

[13] M. Christou, M. Crochemore, T. Flouri, C. S. Iliopoulos, J. Janoušek, B. Melichar, and S. P. Pissis, "Computing all subtree repeats in ordered trees," *Information Processing Letters*, vol. 112, no. 24, pp. 958–962, 2012.

[14] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Soc., 1997, vol. 92.

[15] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[16] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2320–2333, May 2019.

[17] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, May 1998.

[18] Y. De Castro, T. Espinasse, and P. Rochet, "Reconstructing undirected graphs from eigenspaces," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 1679–1702, Jan. 2017.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, pp. 3844–3852.

[20] J. A. Deri and J. M. F. Moura, "Spectral projector-based graph Fourier transforms," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 785–795, Sep. 2017.

[21] X. Dong, A. Ortega, P. Frossard, and P. Vandergheynst, "Inference of mobility patterns via spectral graph wavelets," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 3118–3122.

[22] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[23] F. Dorfler and F. Bullo, "Kron reduction of graphs with applications to electrical networks," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 60, no. 1, pp. 150–163, 2013.

[24] H. E. Egilmez, Y.-H. Chao, and A. Ortega, "Graph-based transforms for video coding," *arXiv preprint arXiv:1909.00952*, 2020. [Online]. Available: https://arxiv.org/abs/1909.00952

[25] H. E. Egilmez, E. Pavez, and A. Ortega, "Graph learning from data under Laplacian and structural constraints," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 825–841, 2017.

[26] H. E. Egilmez, A. Said, Y. H. Chao, and A. Ortega, "Graph-based transforms for inter predicted video coding," *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 3992–3996, Sep. 2015.

[27] N. Emirov, C. Cheng, J. Jiang, and Q. Sun, "Polynomial graph filter of multiple shifts and distributed implementation of inverse filtering," *arXiv:2003.11152*, March 2020. [Online]. Available: http://arxiv.org/abs/2003.11152

[28] P. Erdös and A. Rényi, "Asymmetric graphs," *Acta Mathematica Hungarica*, vol. 14, no. 3-4, pp. 295–315, Sep. 1963.

[29] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Transactions on Signal Processing*, vol. 40, no. 9, pp. 2174–2193, Sep. 1992.

[30] T. Flouri, K. Kobert, S. P. Pissis, and A. Stamatakis, "An optimal algorithm for computing all subtree repeats in trees," in *Combinatorial Algorithms*, T. Lecroq and L. Mouchard, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 269–282.

[31] G. Fracastoro and E. Magli, "Steerable discrete Fourier transform," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 319–323, March 2017.

[32] G. Fracastoro, D. Thanou, and P. Frossard, "Graph transform learning for image compression," in *Picture Coding Symposium (PCS)*, Dec. 2016, pp. 1–5.

[33] G. Fracastoro, D. Thanou, and P. Frossard, "Graph transform optimization with application to image compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 419–432, 2020.

[34] A. Gavili and X. Zhang, "On the shift operator, graph frequency, and optimal filtering in graph signal processing," *IEEE Transactions on Signal Processing*, vol. 65, no. 23, pp. 6303–6318, Dec. 2017.

[35] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[36] B. Girault, A. Ortega, and S. S. Narayanan, "Irregularity-aware graph Fourier transforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5746–5761, Nov. 2018.

[37] A. Gnutti, F. Guerrini, R. Leonardi, and A. Ortega, "Symmetry-based graph Fourier transforms for image representation," in *IEEE International Conference on Image Processing (ICIP)*, Oct. 2018, pp. 2575–2579.

[38] C. Godsil and G. Royle, *Algebraic Graph Theory*, ser. Graduate Texts in Mathematics. volume 207 of Graduate Texts in Mathematics. Springer, 2001, vol. 207.

[39] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[40] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[41] V. K. Goyal, "Theoretical foundation of transform coding," *IEEE Signal Processing Magazine*, pp. 9–21, Sep. 2001.

[42] M. Grant and S. Boyd, "CVX: Matlab Software for Disciplined Convex Programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[43] H. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 10, pp. 1455–1461, 1987.

[44] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[45] J. Han, A. Saxena, V. Melkote, and K. Rose, "Jointly optimized spatial prediction and block transform for video and image coding," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 1874–1884, Apr. 2012.

[46] J. Han, Y. Xu, and D. Mukherjee, "A butterfly structured design of the hybrid transform coding scheme," in *Picture Coding Symposium*, 2013, pp. 1–4.

[47] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171–1220, 2008.

[48] I. Höntsch and L. J. Karam, "Adaptive image coding with perceptual distortion control," *IEEE Transactions on Image Processing*, vol. 11, no. 3, pp. 213–222, March 2002.

[49] W. Hu, G. Cheung, and A. Ortega, "Intra-prediction and generalized graph Fourier transform for image coding," *Signal Processing Letters, IEEE*, vol. 22, no. 11, pp. 1913–1917, Nov. 2015.

[50] W. Hu, G. Cheung, A. Ortega, and O. C. Au, "Multiresolution graph Fourier transform for compression of piecewise smooth images," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 419–433, Jan. 2015.

[51] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 274–288, Jan. 2017.

[52] D. Jakobson, S. D. Miller, I. Rivin, and Z. Rudnick, *Eigenvalue Spacings for Regular Graphs*. New York, NY: Springer New York, 1999, pp. 317–327.

[53] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall Professional Technical Reference, 1990.

[54] J.-Y. Kao, A. Ortega, and S. S. Narayanan, "Graph-based approach for motion capture data representation and analysis," in *IEEE International Conference on Image Processing (ICIP)*, Oct. 2014, pp. 2061–2065.

[55] T. N. Kipf and M. Welling, "Semi-Supervised classification with graph convolutional networks," *arXiv:1609.02907 [cs, stat]*, Feb. 2017, arXiv: 1609.02907. [Online]. Available: http://arxiv.org/abs/1609.02907

[56] G. Kirchhoff, "Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer StrÃűme gefÃijhrt wird," *Annalen der Physik*, vol. 148, no. 12, pp. 497–508, 1847. [Online]. Available: http://dx.doi.org/10.1002/andp.18471481202

[57] H. Kitajima, "A symmetric cosine transform," *IEEE Transactions on Computers*, vol. C-29, no. 4, pp. 317–323, Apr. 1980.

[58] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[59] C. W. Kok, "Fast algorithm for computing discrete cosine transform," *IEEE Transactions on Signal Processing*, vol. 45, no. 3, pp. 757–760, Mar. 1997.

[60] L. Le Magoarou and R. Gribonval, "Are there approximate fast Fourier transforms on graphs?" in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 4811–4815.

[61] L. Le Magoarou and R. Gribonval, "Flexible multilayer sparse approximations of matrices and applications," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 4, pp. 688–700, 2016.

[62] L. Le Magoarou, R. Gribonval, and N. Tremblay, "Approximate fast graph Fourier transforms via multilayer sparse approximations," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 2, pp. 407–420, June 2018.

[63] L. Le Magoarou, N. Tremblay, and R. Gribonval, "Analyzing the approximation error of the fast graph Fourier transform," in *Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 45–49.

[64] B. Li, O. G. Guleryuz, J. Ehmann, and A. Vosoughi, "Layered-givens transforms: Tunable complexity, high-performance approximation of optimal non-separable transforms," in *IEEE International Conference on Image Processing (ICIP)*, Sep. 2017, pp. 1687–1691.

[65] B. Li, J. Han, and Y. Xu, "Fast transform type selection using conditional Laplacian distribution based rate estimation," in *Applications of Digital Image Processing XLIII*, vol. 11510. SPIE, 2020, pp. 461–468.

[66] S. Z. Li, *Markov Random Field Modeling in Image Analysis*, 3rd ed. Springer Publishing Company, Incorporated, 2009.

[67] J. Liu, E. Isufi, and G. Leus, "Filter design for autoregressive moving average graph filters," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 1, pp. 47–60, July 2019.

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

[69] K.-S. Lu and A. Ortega, "Fast GFT based on graph symmetry." [online] https://github.com/kslu/fastgft.

[70] ——, "Symmetric line graph transforms for inter predictive video coding," in *Picture Coding Symposium (PCS)*, Dec. 2016, pp. 1–5.

[71] ——, "A graph Laplacian matrix learning method for fast implementation of graph Fourier transform," in *IEEE International Conference on Image Processing (ICIP)*, Sep. 2017, pp. 1677–1681.

[72] ——, "Fast graph Fourier transforms based on graph symmetry and bipartition," *IEEE Transactions on Signal Processing*, vol. 67, no. 18, pp. 4855–4869, 2019.

[73] ——, "Lapped transforms: A graph-based extension," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5401–5405.

[74] K.-S. Lu, A. Ortega, D. Mukherjee, and Y. Chen, "Efficient rate-distortion approximation and transform type selection using Laplacian operators," in *Picture Coding Symposium (PCS)*, June 2018, pp. 76–80.

[75] ——, "DCT and DST graph filtering with sparse shift operators," *in preparation*, 2020.

[76] ——, "Perceptually-inspired weighted MSE optimization using irregularity-aware graph Fourier transform," *IEEE International Conference on Image Processing (ICIP)*, 2020.

[77] K.-S. Lu, E. Pavez, and A. Ortega, "On learning Laplacians of tree structured graphs," in *IEEE Data Science Workshop (DSW)*, June 2018, pp. 205–209.

[78] J. Ma, W. Huang, S. Segarra, and A. Ribeiro, "Diffusion filtering of graph signals and its use in recommendation systems," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4563–4567.

[79] H. S. Malvar, "Lapped transforms for efficient transform/subband coding," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 6, pp. 969–978, June 1990.

[80] ——, *Signal Processing with Lapped Transforms*, ser. Artech House communications library. Artech House, 1992.

[81] H. S. Malvar and D. H. Staelin, "The LOT: transform coding without blocking effects," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 4, pp. 553–559, April 1989.

[82] S. A. Martucci, "Symmetric convolution and the discrete sine and cosine transforms," *IEEE Transactions on Signal Processing*, vol. 42, no. 5, pp. 1038–1051, 1994.

[83] J. McClellan, T. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 21, no. 6, pp. 506–526, 1973.

[84] J. Mei and J. M. F. Moura, "Signal processing on graphs: Causal modeling of unstructured data," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 2077–2092, 2017.

[85] M. Ménoret, N. Farrugia, B. Pasdeloup, and V. Gripon, "Evaluating graph signal processing for neuroimaging through classification and dimensionality reduction," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov. 2017, pp. 618–622.

[86] B. Mohar, "The Laplacian spectrum of graphs," in *Graph Theory, Combinatorics, and Applications*. Wiley, 1991, pp. 871–898.

[87] S. Narang and A. Ortega, "Perfect Reconstruction Two-Channel Wavelet Filter Banks for Graph Structured Data," *Signal Processing, IEEE Transactions on*, vol. 60, no. 6, pp. 2786–2799, Jun. 2012.

[88] H. Q. Nguyen and M. N. Do, "Downsampling of signals on graphs via maximum spanning trees." *IEEE Trans. Signal Processing*, vol. 63, no. 1, pp. 182–191, 2015.

[89] M. Onuki, S. Ono, M. Yamagishi, and Y. Tanaka, "Graph signal denoising via trilateral filter on graph spectral domain," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 2, no. 2, pp. 137–148, June 2016.

[90] A. Ortega, *Graph Signal Processing: An Introduction*. Cambridge University Press, 2021.

[91] Y. Park and H. Park, "Design and analysis of an image resizing filter in the block-DCT domain," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 2, pp. 274–279, 2004.

[92] B. Pasdeloup, V. Gripon, G. Mercier, D. Pastor, and M. G. Rabbat, "Characterization and inference of graph diffusion processes from observations of stationary signals," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 3, pp. 481–496, Sep. 2018.

[93] Y. C. Pati, R. Rezaiifar, Y. C. P. R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, 1993, pp. 40–44.

[94] E. Pavez and A. Ortega, "Generalized Laplacian precision matrix estimation for graph signal processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 6350–6354.

[95] E. Pavez, A. Ortega, and D. Mukherjee, "Learning separable transforms by inverse covariance estimation," *IEEE International Conference on Image Processing (ICIP)*, 09 2017.

[96] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica ®*. New York, NY, USA: Cambridge University Press, 2003.

[97] M. Peris, S. Martull, A. Maki, Y. Ohkawa, and K. Fukui, "Towards a simulation driven stereo vision system," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, Nov. 2012, pp. 1038–1042.

[98] N. P. Pitsianis, "The Kronecker Product in Approximation and Fast Transform Generation," Ph.D. dissertation, Cornell University, Ithaca, NY, USA, 1997, uMI Order No. GAX97-16143.

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

[100] I. Rotondo, G. Cheung, A. Ortega, and H. E. Egilmez, "Designing sparse graphs via structure tensor for block transform coding of images," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, Dec. 2015, pp. 571–574.

[101] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications*. CRC Press, 2005.

[102] A. Said and W. A. Pearlman, "Low-complexity waveform coding via alphabet and sample-set partitioning," in *Proceedings of IEEE International Symposium on Information Theory*, June 1997, pp. 61–73.

[103] V. Sanchez, P. Garcia, A. M. Peinado, J. C. Segura, and A. J. Rubio, "Diagonalizing properties of the discrete cosine transforms," *IEEE Transactions on Signal Processing*, vol. 43, no. 11, pp. 2631–2641, 1995.

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

[105] ——, "Discrete signal processing on graphs: Frequency analysis," *Signal Processing, IEEE Transactions on*, vol. 62, no. 12, pp. 3042–3054, Jun. 2014.

[106] S. Sardellitti, S. Barbarossa, and P. D. Lorenzo, "On the graph Fourier transform for directed graphs," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 796–811, Sep. 2017.

[107] A. Saxena and F. C. Fernandes, "DCT/DST-based transform coding for intra prediction in image/video coding," *IEEE Transactions on Image Processing*, vol. 22, no. 10, pp. 3974–3981, Oct. 2013.

[108] S. Segarra, A. G. Marques, G. Mateos, and A. Ribeiro, "Network topology inference from spectral templates," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 3, pp. 467–483, Sep. 2017.

[109] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Transactions on Signal Processing*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[110] L. Seidenari, V. Varano, S. Berretti, A. D. Bimbo, and P. Pala, "Recognizing actions from depth cameras as weakly aligned multi-part bag-of-poses," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2013, pp. 479–485.

[111] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[112] G. Shen and A. Ortega, "Tree-based wavelets for image coding: Orthogonalization and tree selection," in *Picture Coding Symposium*, May 2009, pp. 1–4.

[113] H. S. Shin, C. Lee, and M. Lee, "Ideal filtering approach on DCT domain for biomedical signals: index blocked DCT filtering method (IB-DCTFM)," *J. Med. Syst.*, vol. 34, no. 2, pp. 741–753, Aug. 2010.

[114] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *Signal Processing Magazine, IEEE*, vol. 30, no. 3, pp. 83–98, May 2013.

[115] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via Chebyshev polynomial approximation," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 4, pp. 736–751, Dec. 2018.

[116] A. Smola and R. Kondor, "Kernels and regularization on graphs," in *Learning Theory and Kernel Machines*, vol. 2777, Jan 2003, pp. 144–158.

[117] G. Strang, "The discrete cosine transform," *SIAM review*, vol. 41, no. 1, pp. 135–147, 1999.

[118] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1997.

[119] H. Su, M. Chen, A. Bokov, D. Mukherjee, Y. Wang, and Y. Chen, "Machine learning accelerated transform search for AV1," in *Picture Coding Symposium (PCS)*, 2019, pp. 1–5.

[120] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[121] T. Tanaka, T. Uehara, and Y. Tanaka, "Dimensionality reduction of sample covariance matrices by graph Fourier transform for motor imagery brain-machine interface," in *IEEE Statistical Signal Processing Workshop (SSP)*, June 2016, pp. 1–5.

[122] Y. Tanaka and A. Sakiyama, "M-channel oversampled graph filter banks," *IEEE Transactions on Signal Processing*, vol. 62, no. 14, pp. 3578–3590, 2014.

[123] O. Teke and P. P. Vaidyanathan, "Extending classical multirate signal processing theory to graphs–part ii: M-channel filter banks," *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 423–437, Jan 2017.

[124] O. Teke and P. P. Vaidyanathan, "Sparse eigenvectors of graphs," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 3904–3908.

[125] ——, "Uncertainty principles and sparse eigenvectors of graphs," *IEEE Transactions on Signal Processing*, vol. 65, no. 20, pp. 5406–5420, Oct. 2017.

[126] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996.

[127] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-filtering," *IEEE Transactions on Signal Processing*, vol. 51, no. 6, pp. 1557–1571, June 2003.

[128] N. Tremblay, G. Puy, R. Gribonval, and P. Vandergheynst, "Compressive spectral clustering," in *International Conference on International Conference on Machine Learning (ICML)*, vol. 48. JMLR.org, 2016, pp. 1002–1011.

[129] U. Tuna, S. Peltonen, and U. Ruotsalainen, "Gap-filling for the high-resolution pet sinograms with a dedicated DCT-domain filter," *IEEE Transactions on Medical Imaging*, vol. 29, no. 3, pp. 830–839, 2010.

[130] J. Valin, T. B. Terriberry, N. E. Egge, T. Daede, Y. Cho, C. Montgomery, and M. Bebenita, "Daala: Building a next-generation video codec from unconventional technology," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, Sep. 2016, pp. 1–6.

[131] C. F. Van Loan and N. Pitsianis, *Approximation with Kronecker Products*. Dordrecht: Springer Netherlands, 1993, pp. 293–314.

[132] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[133] C.-Y. Wang, S.-M. Lee, and L.-W. Chang, "Designing JPEG quantization tables based on human visual system," *Signal Processing: Image Communication*, vol. 16, no. 5, pp. 501–506, 2001.

[134] Z. Wang, "Fast algorithms for the discrete w transform and for the discrete Fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 4, pp. 803–816, Aug. 1984.

[135] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, 2004.

[136] Z. Wang and B. Hunt, "The discrete W transform," *Applied Mathematics and Computation*, vol. 16, no. 1, pp. 19–48, 1985.

[137] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, Nov. 2003, pp. 1398–1402.

[138] A. C. Yağan and M. T. Özgen, "A spectral graph wiener filter in graph Fourier domain for improved image denoising," in *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2016, pp. 450–454.

[139] C. Yeo, Y. H. Tan, Z. Li, and S. Rahardja, "Mode-dependent fast separable klt for block-based intra coding," in *IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 621–624.

[140] P. Yip and K. Rao, "A fast computational algorithm for the discrete sine transform," *IEEE Transactions on Communications*, vol. 28, no. 2, pp. 304–307, Feb. 1980.

[141] W. Zeng, S. Daly, and S. Lei, "Point-wise extended visual masking for JPEG-2000 image compression," in *Proceedings 2000 International Conference on Image Processing*, vol. 1, Sep. 2000, pp. 657–660.

[142] C. Zhang and D. Florêncio, "Analyzing the optimality of predictive transform coding using graph-based models," *Signal Processing Letters, IEEE*, vol. 20, no. 1, pp. 106–109, Jan. 2013.

[143] C. Zhang, D. Florêncio, and P. A. Chou, "Graph signal processing-A probabilistic framework," *Technical Report*, Apr. 2015.

[144] D. Zhang and J. Liang, "Graph-based transform for 2D piecewise smooth signals with random discontinuity locations," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 1679–1693, April 2017.

[145] H. Zhang and F. Ding, "On the Kronecker products and their applications," *Journal of Applied Mathematics*, vol. 2013, 06 2013.

[146] X. Zhao, L. Zhang, S. Ma, and W. Gao, "Video coding with rate-distortion optimized transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 1, pp. 138–151, Jan 2012.

[147] F. Zou, O. C. Au, C. Pang, J. Dai, X. Zhang, and L. Fang, "Rate-distortion optimized transforms based on the Lloyd-type algorithm for intra block coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 1072–1083, Dec 2013.