The Extended Generalized Haar-Walsh Transform and Applications

By

Yiqun Shao

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Naoki Saito (Chair)

James Bremer

James Sharpnack

Committee in Charge

2020

Contents

The extended Generalized Haar-Walsh Transform and applications

Abstract

Extending computational harmonic analysis tools from the classical setting of regular lattices to the more general setting of graphs and networks is very important and much research has been done recently. The previous Generalized Haar-Walsh Transform (GHWT) is a multiscale transform for signals on graphs, which is a generalization of the classical Haar and Walsh-Hadamard Transforms. We propose the extended Generalized Haar-Walsh Transform (eGHWT). The eGHWT and its associated best-basis selection algorithm for graph signals will significantly improve the performance of the previous GHWT with the similar computational cost, $O(N \log N)$, where N is the number of nodes of an input graph. While the previous GHWT/best-basis algorithm seeks the most suitable orthonormal basis for a given task among more than $(1.5)^N$ possible orthonormal bases in $\mathbb{R}^N$, the eGHWT/best-basis algorithm can find a better one by searching through more than $0.618(1.84)^N$ possible orthonormal bases in $\mathbb{R}^N$. This dissertation describes the details of the eGHWT/basis-basis algorithm and demonstrates its superiority using several examples including genuine graph signals as well as conventional digital images viewed as graph signals. Futhermore, we display how eGHWT can be extended to 2D signals by viewing them as tensors of graphs and the associated applications to efficient image approximation and nonnegative matrix factorizations.

Acknowledgments

First and foremost, I'd like to thank my advisor, Professor Naoki Saito. He has given me a lot of guidance and advice in my research. He has invested a lot of time and energy in me, while always being patient and helpful. I am happy and grateful to have him as my adviser. He has also provided me with financial support and travel funding. I really appreciate all the things he has done for me.

In addition, I'd like to thank my parents. They've been there supporting and encouraging me for so many years. Without them, I couldn't have finished my PHD degree. I also need to thank my grandfather, who has taught me a lot of math knowledge and inspired me when I was young.

I want to thank all the members of my dissertation committee, Professors Naoki Saito, James Bremer and James Sharpnack, for taking the time to read my dissertation and give their helpful feedback.

I'd like to thank my collegues in our group, Jeff Irion, Alex Berrian, David Weber, and Haotian Li for discussing with me on research and giving me helpful advice. In particular, the MTSG toolbox (`https://github.com/JeffLIrion/MTSG_Toolbox`) developed by Jeff Irion was very helpful in designing my own toolbox.

Thanks to my officemates, Ji Chen, Bohan Zhou, Yunshen Zhou, and Jiaxiang Li. They have helped me a lot in discussing problems, TA work, and course work.

I also need to thank my girlfriend Yuqing Su, for allowing me to stay at her home while I finished writing my dissertation. I appreciate all the encouragement she gave me, especially during the difficult time of COVID-19.

CHAPTER 1

Introduction

In recent years, research on graphs and networks is experiencing rapid growth due to a confluence of several trends in science and technology: the advent of new sensors, measurement technologies, and social network infrastructure has provided huge opportunities to visualize complicated interconnected network structures, record data of interest at various locations in such networks, analyze such data, and make inferences and diagnostics. We can easily observe such network-based problems in truly diverse fields: biology and medicine (e.g., connectomes); computer science (e.g., social networks); electrical engineering (e.g., sensor networks); hydrology and geology (e.g., ramified river networks); and civil engineering (e.g., road networks), to name just a few. Consequently, there is an explosion of interest and demand to analyze data sampled on such graphs and networks, which are often called "network data analysis" or "graph signal processing"; see e.g., recent books [**8**, **16**, **48**, **52**] and survey articles [**53**, **60**], to see the evidence of this trend. This trend has forced the signal processing and applied mathematics communities to extend classical techniques on regular domains to the setting of graphs. Much efforts have been done to develop wavelet transforms for signals on graphs (or the so-called graph signals) [**7**, **10**, **11**, **25**, **34**, **41**, **51**, **56**, **64**]. Comprehensive reviews of transforms for signals on graphs have also been written [**53**, **60**].

The Generalized Haar-Walsh Transform (GHWT) [**30**, **31**, **33**], developed by Irion and Saito, has achieved superior results over other transforms in terms of both approximation and denoising of signals on graphs (or graph signals for short). It is a generalization of the classical Haar and Walsh-Hadamard Transforms. In this dissertation, we propose and develop the *extended Generalized Haar-Walsh Transform* (eGHWT). The eGHWT and its associated best-basis selection algorithm for graph signals will significantly improve the performance of the previous GHWT with the similar computational cost, $O(N \log N)$ where N is the number of nodes of an input graph. While the previous GHWT/best-basis algorithm seeks the most suitable orthonormal basis (ONB) for a given task among more than $(1.5)^N$ possible orthonormal bases in $\mathbb{R}^N$, the eGHWT/best-basis

algorithm can find a better one by searching through more than $0.618 \cdot (1.84)^N$ possible orthonormal bases in $\mathbb{R}^N$. It can be extended to 2D signals in a more subtle way than GHWT. In this dissertation, we describe the details of the eGHWT/basis-basis algorithm and demonstrate its superiority. Moreover, we showcase the versatility of eGHWT by applying it to genuine graph signals, classical digital images, and matrix data.

The organization of this dissertation is as follows. In Chapter 2, we review background concepts, including Haar-Walsh wavelet packets (§2.1), graph signal processing (§2.2) and recursive graph partitioning (§2.3), which is a common strategy used by researches to develop graph transforms. In Chapter 3, the GHWT/best-basis algorithm is reviewed. In Chapter 4, we provide an overview of the eGHWT. We start by reviewing the algorithm developed by [**65**] (§4.1). Then we illustrate how that algorithm can be modified to construct the eGHWT (§4.2, §4.3, §4.4). An example illustrating the difference between the GHWT and the eGHWT on a six-node path graph signal is given (§4.5). We finish the chapter by expaining how the eGHWT can be extended to 2D signals (§4.6). In Chapter 5, we demonstrate the superiority of the eGHWT over the GHWT (including the graph Haar and Walsh bases) using real datasets. In Chapter 6, we discuss how the eGHWT can be used to make the nonnegative matrix factorization (NMF) algorithm more efficient by finding a better initial estimate to start its iterative algorithms. In addition, we give a brief description of another project [**17**] on matrix completion, with the same assumption with NMF that many real matrices datasets are low rank. We conclude with a summary of the eGHWT and its applications.

The software toolbox including the eGHWT, written entirely in the Julia programming language [**3**], is available from `https://gitlab.com/BoundaryValueProblems/MTSG.jl`.

A preliminary version of the part of the material in this dissertation was presented at the SPIE Conference on Wavelets and Sparsity XVIII, Aug. 2019, San Diego, CA [**58**].

CHAPTER 2

Background

2.1. Haar-Walsh Wavelets Packets

The Fourier transform is everywhere in physics and mathematics. Function can be mapped from time domain to the frequency domain through the Fourier transform and mapped backwards from frequency domain to time domain through the inverse Fourier transform.

For a function $f(t) \in L^1(\mathbb{R})$, the Fourier transform $\mathcal{F} : f \to \hat{f}$ is defined as [**62**]

$$[\mathcal{F}f](\omega) := \hat{f}(\omega) := \int_{-\infty}^{\infty} f(t)\mathrm{e}^{-\mathrm{i}\omega t}\mathrm{d}t$$

If $\hat{f}(\omega)$ also belongs to $L^1(\mathbb{R})$, then the inverse Fourier transform $\mathcal{F}^{-1} : \hat{f} \to f$ is defined as [**62**]

$$[\mathcal{F}^{-1}\hat{f}](t) := \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)\mathrm{e}^{\mathrm{i}\omega t}\mathrm{d}\omega = f(t)$$

Since compactly supported smooth functions are integrable and dense in $L^2(\mathbb{R})$, the Plancherel theorem allows us to extend the definition of the Fourier transform to $f \in L^2(\mathbb{R})$ by continuity arguments [**49**]. The more regular $f(t)$, the faster the decay of the amplitude $|\hat{f}(\omega)|$, when ω increases.

Although Fourier transform works well for analyzing *global and smooth* functions, it does not work well for *localized* functions. Indeed, the support of $\mathrm{e}^{\mathrm{i}\omega t}$ covers the whole real line, so $\hat{f}(\omega)$ depends on the values $f(t)$ for all times $t \in \mathbb{R}$. This global "mix" of information makes it difficult to analyze or represent any local property of $f(t)$ from $\hat{f}(\omega)$. Among many mathematical tools to address this shortcoming, the wavelet transform has proven to be very useful. Wavelets are well localized and few coefficients are needed to represent local transient structures. As opposed to a Fourier basis, a wavelet basis can often generate a sparse representation of piecewise regular signals, which may include transients and singularities [**49**].

A *wavelet*, or a *mother wavelet*, is a function $\psi \in L^2(\mathbb{R})$ satisfying

$$\|\psi\|_2 = 1,$$

$$\int_{-\infty}^{\infty} \psi(t)\mathrm{d}t = \hat{\psi}(0) = 0.$$

Coming in pair is a *scaling function* ϕ, or a *father wavelet*, which satisfies $\|\phi\|_2 = 1$ [**49**].

A family of functions can then be generated by translating and dilating the mother wavelet and father wavelet. The translation operator is defined as

$$T_u f(t) := f(t-u)$$

and dilation operator is defined as

$$D_s f(t) := \frac{1}{\sqrt{s}} f\left(\frac{t}{s}\right).$$

To reduce redundancy, we choose scales $s = 2^j$ and locations $u = 2^j k$, with $j, k \in \mathbb{Z}$. We then define [**49**]

$$\phi_{j,k}(t) := T_{2^j k} D_{2^j} \phi(t) = \frac{1}{\sqrt{2^j}} \phi\left(\frac{t - 2^j k}{2^j}\right) \tag{2.1}$$

$$\psi_{j,k}(t) := T_{2^j k} D_{2^j} \psi(t) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t - 2^j k}{2^j}\right) \tag{2.2}$$

The continuous wavelet transform of a function $f \in L^2(\mathbb{R})$ at scale 2^j and time $2^j k$ is [**49**]

$$Wf(j,k) := \langle f, \psi_{j,k} \rangle = \int_{-\infty}^{\infty} f(t) \overline{\frac{1}{\sqrt{2^j}} \psi(\frac{t - 2^j k}{2^j})} \mathrm{d}t. \tag{2.3}$$

Suppose $\exists B \geq A > 0$ such that

$$A\|f\|_2^2 \leq \sum_{j,k} |\langle f, \psi_{j,k} \rangle|^2 \leq B\|f\|_2^2, \ \forall f \in L^2(\mathbb{R})$$

then f can be reconstructed in the L^2 sense as

$$f(t) = \sum_{k=-\infty}^{\infty} \langle f, \phi_{j_{\max},k} \rangle \tilde{\phi}_{j_{\max},k}(t) + \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{j_{\max}} \langle f, \psi_{j,k} \rangle \tilde{\psi}_{j,k}(t). \tag{2.4}$$

where $\tilde{\psi}_{j,k}(t)$ are the *dual wavelets*, $\tilde{\phi}_{j,k}(t)$ are the *dual scaling function* and $j_{\max}$ denotes the coarsest level. They satisfy the biorthogonality relations

$$\begin{aligned}
\langle \psi_{j,k}, \tilde{\psi}_{j',k'} \rangle &= \delta(j-j')\delta(k-k') && \forall j, j', k, k' \in \mathbb{Z} \\
\langle \phi_{j,k}, \tilde{\phi}_{j,k'} \rangle &= \delta(k-k') && \forall j, k, k' \in \mathbb{Z} \\
\langle \tilde{\phi}_{j,k}, \phi_{j,k'} \rangle &= 0 && \forall j, k, k' \in \mathbb{Z} \\
\langle \psi_{j,k}, \tilde{\phi}_{j,k'} \rangle &= 0 && \forall j, k, k' \in \mathbb{Z}
\end{aligned}$$

where δ is the Kronecker delta. From now on, we restrict to the case of *orthogonal wavelets*, i.e., $\phi_{j,k} = \tilde{\phi}_{j,k}$, $\psi_{j,k} = \tilde{\psi}_{j,k}$. In this case, $\phi_{j,k}$ and $\psi_{j,k}$ can be written as a linear combination of $\psi_{j-1,k}$ at a finer scale.

$$\phi_{j,k}(t) = \sum_n \underbrace{h(n-2k)}_{=\langle \phi_{j,k}, \phi_{j-1,n} \rangle} \phi_{j-1,n}(t) \tag{2.5}$$

$$\psi_{j,k}(t) = \sum_n \underbrace{g(n-2k)}_{=\langle \psi_{j,k}, \phi_{j-1,n} \rangle} \phi_{j-1,n}(t) \tag{2.6}$$

Here h and g are called low-pass and high-pass filters, respectively [**49**]. With the aid of filters, we can perform the Fast Wavelet Transform without constructing all the wavelets and computing coeffcients through inner products explicitly. In other words, the signal can be reconstructed by plugging (2.5) and (2.6) into (2.7) and replacing dual functions,

$$f(t) = \sum_{k=-\infty}^{\infty} c_{j_{\max}}(k)\phi_{j_{\max},k}(t) + \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{j_{\max}} d_j(k)\psi_{j,k}(t). \tag{2.7}$$

where

$$c_j(k) := \langle f, \phi_{j,k} \rangle = \sum_n h(n-2k)c_{j-1}(n) \tag{2.8}$$

$$d_j(k) := \langle f, \psi_{j,k} \rangle = \sum_n g(n-2k)c_{j-1}(n). \tag{2.9}$$

The coefficients c_j and d_j can be computed through the recursion relations (2.8) and (2.9).

Now we have defined the wavelet transform for continuous functions, we then discuss the more efficient implementation on discrete signals. Let us consider the dyadic discrete signal $\boldsymbol{f} \in \mathbb{R}^N$, where $N = 2^m$ ($m \in \mathbb{Z}$). Then scaling coefficients on the finest level are just the input signal itself, i.e., $c_0(k) := \boldsymbol{f}[k],\ \forall\ k \in \{1, \ldots, N\}$. Note that the recursive relations (2.8) and (2.9) still hold in discrete case [**49**]. Then they can be used to generate the c_j and d_j at the coarser levels iteratively. Since there are only a finite number of scales j and locations k, the reconstruction formula (2.7) is generalized to the discrete case as

$$\boldsymbol{f}[n] = \sum_{k=0}^{2^{m-j_{\max}}-1} \underbrace{c_{j_{\max}}(k)}_{:=\langle f, \phi_{j_{\max},k}\rangle} \phi_{j_{\max},k}(n) + \sum_{j=1}^{j_{\max}} \sum_{k=0}^{2^{m-j}-1} \underbrace{d_j(k)}_{:=\langle f, \psi_{j,k}\rangle} \psi_{j,k}(n), \tag{2.10}$$

where c_j and d_j computed through (2.8) and (2.9). Making use of filters and their recursive relations, the time cost of the discrete wavelet transform is only $O(N)$ operations, which is faster than Fast Fourier Transform with $O(N \log N)$.

However, there is one drawback of wavelets. For signals with high-frequency semi-global support components, the representation is poor. It is because that the filters are only applied to scaling vectors. To address this shortcoming, Coifman, Meyer and Wickerhauser developed *wavelet packets* [**12**]. In addition to the scaling vectors, low-pass and high-pass filters are also applied to the high-frequency wavelets. Using the notation in [**55**], we set $w^0_{j,k}(t) := \phi_{j,k}(t)$ and $w^1_{j,k}(t) := \psi_{j,k}(t)$. Then the wavelet packet functions are generated by

$$w^{2l}_{j,k}(t) := \sum_n h(n-2k) w^l_{j-1,n}(t) \tag{2.11}$$

$$w^{2l+1}_{j,k}(t) := \sum_n g(n-2k) w^l_{j-1,n}(t) \tag{2.12}$$

and wavelet packet coefficients via

$$d^{2l}_j(k) := \langle \boldsymbol{f}, w^l_{j,k}\rangle = \sum_n h(n-2k) d^l_{j-1}(n) \tag{2.13}$$

$$d_j^{2l+1}(k)(t) := \langle \boldsymbol{f}, w_{j,k}^l \rangle = \sum_n g(n-2k) d_{j-1}^l(n) \tag{2.14}$$

Note that the scaling and wavelet coefficients are a subset of the wavelet packet coefficients with $c_j(k) = d_j^0(k)$ and $d_j(k) = d_j^1(k)$. The same goes for the corresponding vectors.

Similarly, let us still consider $\boldsymbol{f} \in \mathbb{R}^N (N = 2^m,\ m \in \mathbb{Z})$. We perform the wavelet packet transform using refinement relations (2.13) and (2.14) in the same manner as (2.8) and (2.9). The output of the wavelet packet transform is an $N \times (m+1)$ matrix of expansion coefficient, meaning N coefficients on each level of $(m+1)$ levels. The time cost of wavelet packet transform is then $O(N \log N)$ now, since $m = \log_2 N$.

This dissertation mainly studies the generalized Haar-Walsh transform on the graph. Here we give an example of the classical Haar-Walsh wavelet packets. The Haar wavelet is

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \le t < 1/2 \\ -1 & \text{if } -1/2 \le t < 0 \\ 0 & \text{otherwise} \end{cases} \cdot$$

The scaling function is

$$\phi(t) = \begin{cases} 1 & \text{if } -1/2 \le t < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

The Haar wavelet has compact support of length 1 and has 1 vanishing moment, which means that it is orthogonal to constant functions.

In the case of $N = 8$, Figure 2.1 [**29**] displays the Haar-Walsh wavelet packet functions and Figure 2.2 [**29**] displays the wavelet packet coefficients. As we can see, these functions are piecewise-constant. Specifically, the funtions on the bottom level are known as the *Walsh functions*, with values only from $\{-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}\}$. These Walsh functions correspond to the rescaled columns of the $N \times N$ *Hadamard matrix* H_N [**20**]. H_N assumes only the values $\{-1, 1\}$ and satisfies $H_N H_N^\intercal = H_N^\intercal H_N = N I_N$.

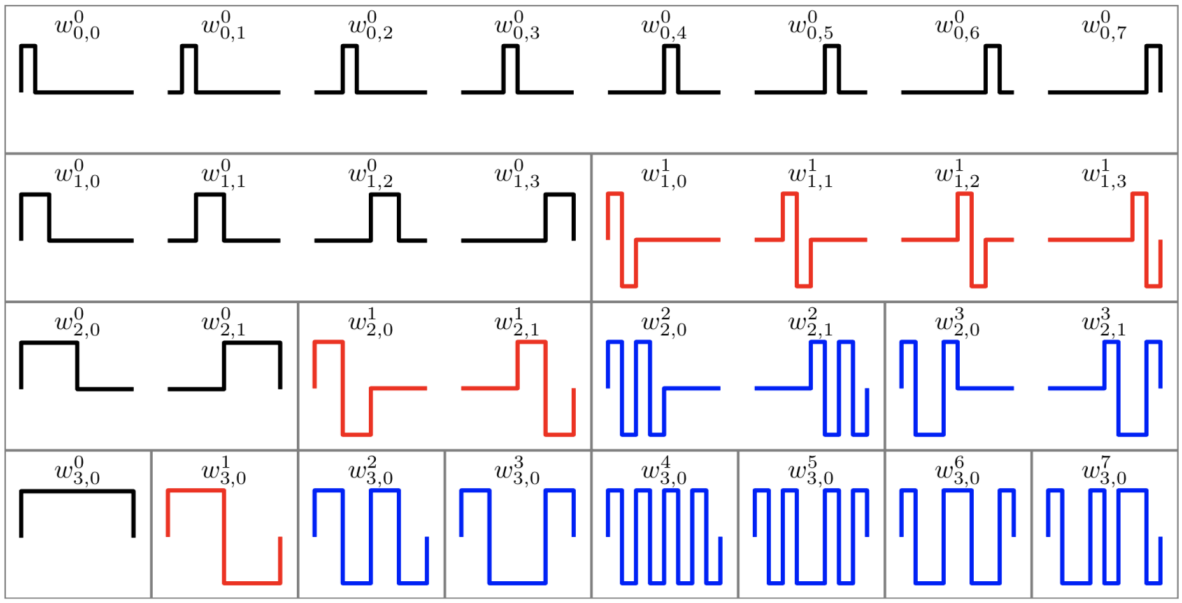


FIGURE 2.1. The Haar-Walsh wavelet packets for a discrete signal with 8 nodes. Scaling functions ($l = 0$) are in black, wavelet functions ($l = 1$) are in red, and wavelet packet functions ($l \geq 2$) are in blue. The functions on the bottom level are called Walsh functions [**29**].

In Figure 2.1, the functions in red together with $w^0_{3,0}$ form the basis vectors of the *Haar wavelet transform*.

Obviously, the number of basis vectors ($N(\log_2 N + 1)$) inside a wavelet packet is redundant to represent $\boldsymbol{f} \in \mathbb{R}^N$. The *best-basis algorithm* will be used to select the "best" basis over these vectors according to the signals and tasks at hand. We will go through best-basis algorithm in detail in the next chapter.

$j=0$	$d_0^0(0)$	$d_0^0(1)$	$d_0^0(2)$	$d_0^0(3)$	$d_0^0(4)$	$d_0^0(5)$	$d_0^0(6)$	$d_0^0(7)$
$j=1$	$d_1^0(0)$	$d_1^0(1)$	$d_1^0(2)$	$d_1^0(3)$	$d_1^1(0)$	$d_1^1(1)$	$d_1^1(2)$	$d_1^1(3)$
$j=2$	$d_2^0(0)$	$d_2^0(1)$	$d_2^1(0)$	$d_2^1(1)$	$d_2^2(0)$	$d_2^2(1)$	$d_2^3(0)$	$d_2^3(1)$
$j=3$	$d_3^0(0)$	$d_3^1(0)$	$d_3^2(0)$	$d_3^3(0)$	$d_3^4(0)$	$d_3^5(0)$	$d_3^6(0)$	$d_3^7(0)$

FIGURE 2.2. Wavelet packet coefficients for a discrete signal with 8 nodes. Scaling coefficients ($l = 0$) are in black, wavelet coefficients ($l = 1$) are in red, and wavelet packet coefficients ($l \geq 2$) are in blue [**29**].

2.2. Basics of Graph Theory and Notation

In this section, we cover some fundamental graph theory and introduce the graph notations that will be used throughout this dissertation.

A graph is a pair $G = (V, E)$, where $V = V(G) = \{v_1, v_2, \ldots, v_N\}$ is the vertex (or node) set of G, and $E = E(G) = \{e_1, e_2, \ldots, e_M\}$ is the edge set, where each edge connects two nodes v_i, v_j for some $1 \leq i \neq j \leq N$. We only deal with finite N and M in this dissertation. For simplicity, we often write i instead of v_i.

An edge connecting a node i and itself is called a *loop*. If there exists more than one edge connecting some i, j, then they are called *multiple edges*. A graph having loops or multiple edges is called a *multiple graph* (or multigraph); a graph with neither of these is called a *simple graph*. A *directed graph* is a graph in which edges have orientations while *undirected graph* is a graph in which edges do not have orientations. If each edge $e \in E$ has a weight (normally nonnegative), then G is called a *weighted graph*. A *path* from i to j in a graph G is a subgraph of G consisting of a sequence of distinct nodes starting with i and ending with j such that consecutive nodes are adjacent. A path starting from i that returns to i (but is not a loop) is called a *cycle*. For any two distinct nodes in V, if there is a path connecting them, then such a graph is said to be *connected*. In this dissertation, we mainly consider *undirected weighted simple connected graphs*. Our method can be easily adapted to other *undirected graph*, but we do not consider *directed graphs* here.

Sometimes, each node is associated with spatial coordinates in $\mathbb{R}^d$. For example, if we want to analyze a network of sensors and build a graph whose nodes represent the sensors under consideration, then these nodes have spatial coordinates in $\mathbb{R}^2$ or $\mathbb{R}^3$ indicating their current locations. In that case, we write $\boldsymbol{x}[i] \in \mathbb{R}^d$ for the location of node i. Denote the functions supported on graph as $\boldsymbol{f} = (f[1], \ldots, f[N])^\intercal \in \mathbb{R}^N$. It is a data vector (often called a *graph signal*) where $f[i] \in \mathbb{R}$ is the value measured at the node i of the graph.

We now discuss several matrices associated with undirected simple graphs. The information in both V and E is captured by the *edge weight matrix* $W(G) \in \mathbb{R}^{N \times N}$, where $W_{ij} \geq 0$ is the edge weight between nodes i and j. In an unweighted graph, this is restricted to be either 0 or 1, depending on whether nodes i and j are adjacent, and we may refer to $W(G)$ as an *adjacency matrix*. In a weighted graph, W_{ij} indicates the *affinity* between i and j. In either case, since G is undirected, $W(G)$ is a symmetric matrix. We then define the *degree matrix*

$$D(G) := \operatorname{diag}(d_1, \ldots, d_N), \text{ where } d_i := \sum_j W_{ij}.$$

With this in place, we are now able to define the *(unnormalized) Laplacian* matrix, *random-walk normalized Laplacian* matrix, and *symmetric normalized Laplacian* matrix respectively, as

$$L(G) := D(G) - W(G),$$

$$L_{\text{rw}}(G) := D(G)^{-1} L(G),$$

$$L_{\text{sym}}(G) := D(G)^{-1/2} L(G) D(G)^{-1/2}.$$

See [**68**] for the details of the relationship between these three matrices and their spectral properties. We use $0 = \lambda_0 < \lambda_1 \leq \ldots \leq \lambda_{N-1}$ to denote the sorted Laplacian eigenvalues and $\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_{N-1}$ to denote their corresponding eigenvectors, where the specific Laplacian matrix to which they refer will be clear from either context or subscripts.

Laplacian eigenvectors can then be used for graph partitioning. Spectral clustering [**68**] performs k-means on the first few eigenvectors to partition the graph. This approach is justified from the fact that it is an approximate minimizer of the graph-cut criterion called *Ratio Cut* [**23**] (or the *Normalized Cut* [**59**]) when L (or L_{rw}, respectively) is used. Suppose G is partitioned into A

and A^c, then *Ratio Cut* and *Normalized Cut* are defined by

$$\begin{aligned}
\text{cut}(A, A^c) &:= \sum_{i \in A, j \in A^c} W_{ij} \\
\text{vol}(A) &:= \sum_{i \in A} d_i \\
\text{Ratio Cut}(A, A^c) &:= \frac{\text{cut}(A, A^c)}{|A|} + \frac{\text{cut}(A, A^c)}{|A^c|} \\
\text{Normalized Cut}(A, A^c) &:= \frac{\text{cut}(A, A^c)}{\text{vol}(A)} + \frac{\text{cut}(A, A^c)}{\text{vol}(A^c)}
\end{aligned}$$

To reduce the computational complexity (as we did for the GHWT construction), we only use the *Fiedler vector* [**18**], i.e., the eigenvector $\boldsymbol{\phi}_1$ corresponding to the smallest nonzero eigenvalue λ_1, to bipartition a given graph (or subgraph) in this dissertation. For a connected graph G, Fiedler showed that *Fiedler vector* partitions the vertices into two sets by letting

$$V_1 = \{i \,|\, \boldsymbol{\phi}_1[i] \geq 0\},$$

$$V_2 = V \setminus V_1,$$

such that the subgraphs induced on V_1 and V_2 by G are both connected graphs [**18**].

2.3. Recursive Partitioning of Graphs

The foundation upon which the GHWT (and the eGHWT) is constructed is a *binary partition tree* (also known as a *hierarchical bipartition tree*) of an input graph $G(V, E)$: a set of tree-structured subgraphs of G constructed by recursively bipartitioning G. This bipartitioning operation ideally splits each subgraph into two smaller subgraphs that are roughly equal in size while keeping tightly-connected nodes grouped together. As mentioned in the last section, we typically use the Fiedler vectors of the L_{rw} matrices of subgraphs for this bipartitioning. More specifically, let G_k^j denote the kth subgraph on level j of the binary partition tree of G. Note $G_0^0 = G$ and level $j = 0$ represents the root node of this tree. Then the two children of G_k^j in the tree, $G_{k'}^{j+1}$ and $G_{k'+1}^{j+1}$, are obtained through partitioning G_k^j using the Fiedler vector of $L_{\text{rw}}(G_k^j)$. The graph partitioning is recursively performed until each subgraph corresponding to the leaf contains only one node.

In general, other spectral clustering methods with different number of eigenvectors or different Laplacian matrices are applicable as well. We impose five conditions on the binary partition tree

(1) The root of the tree is the entire graph, i.e., $G_0^0 = G$.
(2) The leaves of the tree are single-node graphs, i.e., $|V(G_k^{j_{\max}})| = 1$. Here $j_{\max}$ is the height of the tree.
(3) All regions on the same level are disjoint, i.e., $V(G_k^j) \cup V(G_{k'}^j) = \emptyset$ if $k \neq k'$.
(4) Each subgraph with more than one node is partitioned into exactly two subgraphs.
(5) (Optional) In practice, the size of the two children, $|V(G_{k'}^{j+1})|$ and $|V(G_{k'+1}^{j+1})|$ should not be too far apart to reduce inefficiency.

Even other graph cut methods can be used to form the binary partition tree, as long as those conditions are satisfied. The flexibility of our methods is advantageous.

We demonstrate two examples illustrating the partition tree here. The first one is a simple 6-node path graph. It has five edges with equal weights connecting adjacent nodes. Figure 2.3 is the partition tree formed on the graph. In the first iteration, it is bipartitioned into two subgraphs with 3 nodes each. Then each of those 3 nodes graphs is biparitioned into an 1-node graph and a 2-nodes graph. In the end, the subgraphs are all 1-node graph.

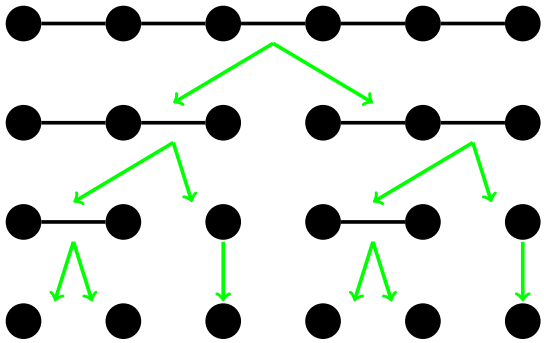


FIGURE 2.3. An example of a hierarchical tree for a path-graph with $N = 6$ nodes, where the edge weights are equal. The root is the whole graph. The Fiedler vector of L_{rw} is used to do the bipartition.

The second example is the the vehicular traffic volume data on the Toronto road network [1], which contains the most recent 8 peak-hour vehicle volume counts collected at intersections where there are traffic signals. The data was typically collected between the hours of 7:30 am and 6:00 pm, over the period of 03/22/2004–02/28/2018. We generated the road network of Toronto using the street names and intersection coordinates included in the dataset. The graph has $N = 2275$ nodes and $M = 3381$ edges. Figure 2.4 displays this vehicular volume data where the size of the marker is proportional to the vehicular volume. Figure 2.5 gives us a visualization of the first 3 levels of the partition tree on Toronto road data. The Fiedler vector of L_{rw} is used to do the bipartition here. Note that how partition tree is computed is not affected by the signal values on the nodes, i.e., the vehicular volume here.

[1] `https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/transportation/#7c8e7c62-7630-8b0f-43ed-a2dfe24aadc9`

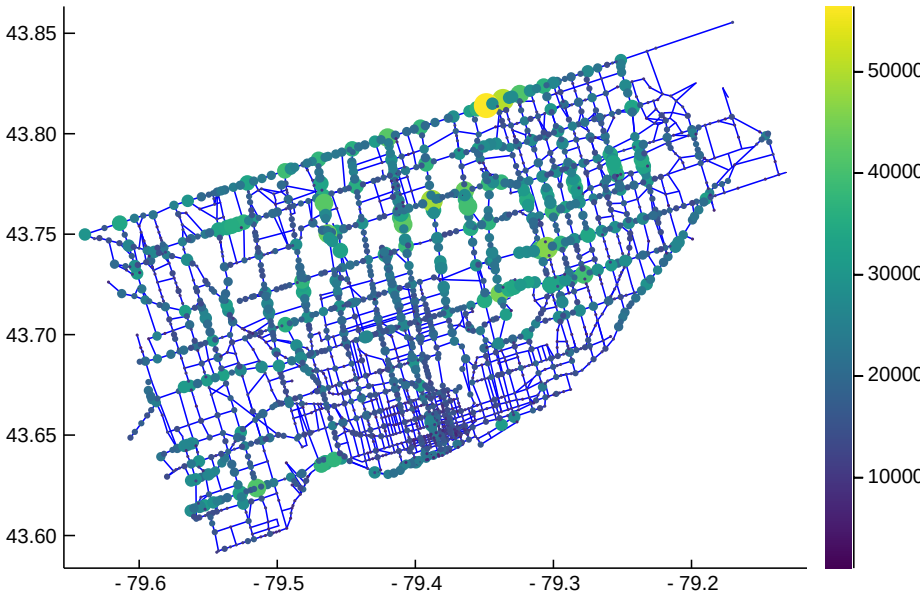
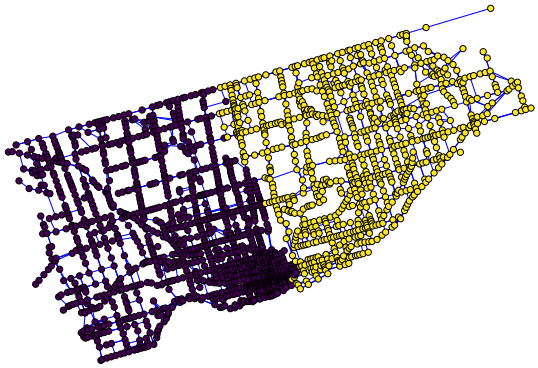
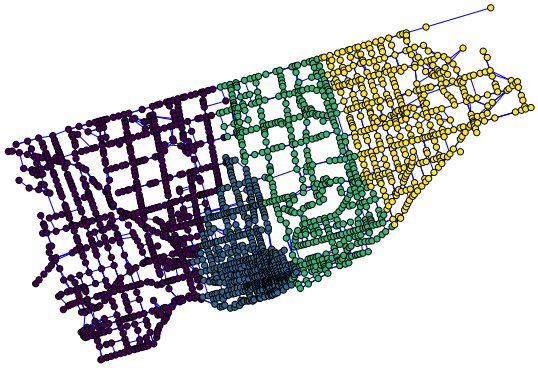


FIGURE 2.4. Vehicular traffic volume data on the Toronto road network

(a) Level $j = 0$



(b) Level $j = 1$



(c) Level $j = 2$

FIGURE 2.5. A demonstration of partition tree. On the same level, different colors correspond to different regions.

CHAPTER 3

The Generalized Haar-Walsh Transform (GHWT)

In this chapter, we will review the Generalized Haar-Walsh Transform (GHWT) [**30**, **31**, **33**]. It is a multiscale transform for graph signals and a *true* generalization of the classical Haar and Walsh-Hadamard transforms: if an input graph is a simple path graph whose number of nodes is dyadic, then the GHWT reduces to the classical counterpart *exactly*.

3.1. Overcomplete Dictionaries of Bases

After the binary partition tree of the input graph with depth $j_{\max}$ is generated, an overcomplete dictionary of basis vectors is composed. Each basis vector is denoted as $\boldsymbol{\psi}_{k,l}^{j}$, where $j \in [0, j_{\max}]$ denotes the level, $k \in [0, K^j)$ denotes the region and l denotes the tag. K^j is the number of sub-graphs on level j. The tag l assumes a distinct integer value within the range $[0, 2^{j_{\max}-j})$. The tag l, when expressed in binary, specifies the sequence of average and difference operations by which $\boldsymbol{\psi}_{k,l}^{j}$ was generated. For example, $l = 6$ written in binary is 110, which means that the basis vector/expansion coefficient was produced by two differencing operations (two 1s) followed by an averaging operation (one 0). Generally speaking, larger l values indicate more oscillation, with exceptions when imbalances occur in the recursive partitioning. We refer to basis vectors with tag $l = 0$ as *scaling vectors*, those with tag $l = 1$ as *Haar vectors*, and those with tag $l \geq 2$ as *Walsh vectors*.

The GHWT begins by defining an orthonormal basis on level $j_{\max}$ and obtaining the corresponding expansion coefficients. The canonical basis is used here since each single region are 1-node graph: $\boldsymbol{\psi}_{k,0}^{j_{\max}} := \mathbf{1}_{V(G_k^{j_{\max}})}$, where $k \in [0, N)$ and $|V(G_k^{j_{\max}})| = 1$. The expansion coeffcients $d_{k,0}^{j_{\max}}$ are then simply the reordered input signal $\boldsymbol{f}$. From here the algorithm proceeds recursively, the basis vectors and expansion coefficients on level $j-1$ are computed from those on level j. The GHWT proceeds as in Algorithms 1 and 2.

Algorithm 1: Generating the GHWT Dictionary of basis vectors [**30**, **31**, **33**]

Input: A binary partition tree $\{G_k^j\}$ of the graph G, $0 \le j \le j_{\max}$ and $0 \le k < K^j$. $N_k^j := |V(G_k^j)|$. K^j denotes the number of subgraphs on level j.

Output: An overcomplete dictionary of basis vectors $\{\boldsymbol{\psi}_{k,l}^j\}$

for $k = 0, \ldots, N-1$ **do** `// Basis vectors on level` $j_{\max}$ `are unit vectors`

  $\boldsymbol{\psi}_{k,0}^{j_{\max}} \leftarrow 1_{V(G_k^{j_{\max}})} \in \mathbb{R}^N$

end

for $j = j_{\max}, \ldots, 1$ **do** `// Compose basis vectors on level` $j-1$ `from level` j

  for $k = 0, \ldots, K^{j-1} - 1$ **do**

    $\boldsymbol{\psi}_{k,0}^{j-1} \leftarrow 1_{V(G_k^{j-1})} / \sqrt{N_k^{j-1}}$ `// Compute the scaling vector`

    `// Basis vectors supported on` $V(G_k^{j-1})$ `are computed from those on` $V(G_{k'}^j)$ `and` $V(G_{k'+1}^j)$. $G_{k'}^j$ `and` $G_{k'+1}^j$ `are the two subgraphs of` G_k^{j-1}

    if $N_k^{j-1} > 1$ **then**

      $\boldsymbol{\psi}_{k,1}^{j-1} \leftarrow \frac{N_{k'+1}^j \sqrt{N_{k'}^j} \boldsymbol{\psi}_{k',0}^j - N_{k'}^j \sqrt{N_{k'+1}^j} \boldsymbol{\psi}_{k'+1,0}^j}{\sqrt{N_{k'}^j (N_{k'+1}^j)^2 + N_{k'+1}^j (N_{k'}^j)^2}}$ `// Compute the Haar vector`

    end

    if $N_k^{j-1} > 2$ **then**

      for $l = 1, \ldots, 2^{j_{\max}-j} - 1$ **do** `// Compute the Walsh-like vectors`

        if *both subregions* k' *and* $k'+1$ *have a basis vector with tag* l **then**

          $\boldsymbol{\psi}_{k,2l}^{j-1} \leftarrow (\boldsymbol{\psi}_{k',l}^j + \boldsymbol{\psi}_{k'+1,l}^j)/\sqrt{2}$;

          $\boldsymbol{\psi}_{k,2l+1}^{j-1} \leftarrow (\boldsymbol{\psi}_{k',l}^j - \boldsymbol{\psi}_{k'+1,l}^j)/\sqrt{2}$;

        else if *(without loss of generality) only subregion* k' *has a basis vector with tag* l **then**

          $\boldsymbol{\psi}_{k,2l}^{j-1} \leftarrow \boldsymbol{\psi}_{k',l}^j$

        else if *Neither subregion has a basis vector with tag* l **then**

          Do nothing

      end

    end

  end

end

Algorithm 2: Generating the GHWT Dictionary coeffcients [**30**, **31**, **33**]

Input: A binary partition tree $\{G_k^j\}$ of the graph G, $0 \le j \le j_{\max}$ and $0 \le k < K^j$. $N_k^j := |V(G_k^j)|$. Input signal $\boldsymbol{f}$ supported on G

Output: The set of expanding coefficients $\{d_{k,l}^j\}$ of signal $\boldsymbol{f}$ on the GHWT dictionary $\{\boldsymbol{\psi}_{k,l}^j\}$

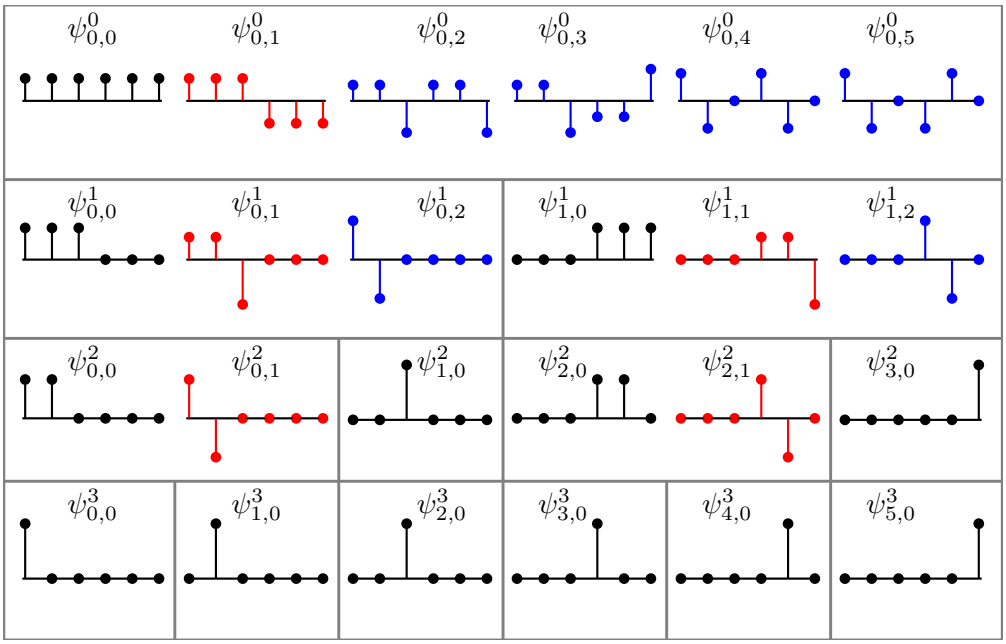
for $k = 0, \ldots, N-1$ **do** `// Basis vectors on level` $j_{\max}$ `are unit vectors`
  $d_{k,0}^{j_{\max}} \leftarrow \langle 1_{V(G_k^{j_{\max}})}, \boldsymbol{f} \rangle$
end
for $j = j_{\max}, \ldots, 1$ **do** `// Compute coefficients on level` $j-1$ `from level` j
  for $k = 0, \ldots, K^{j-1} - 1$ **do**
    $d_{k,0}^{j-1} \leftarrow \langle 1_{V(G_k^{j-1})} / \sqrt{N_k^{j-1}}, \boldsymbol{f} \rangle$ `// Compute the scaling coefficient`
    `// Coefficients of basis vectors supported on` $V(G_k^{j-1})$ `are computed from those on` $V(G_{k'}^j)$ `and` $V(G_{k'+1}^j)$. $G_{k'}^j$ `and` $G_{k'+1}^j$ `are the two subgraphs of` G_k^{j-1}
    if $N_k^{j-1} > 1$ **then**
      $d_{k,1}^{j-1} \leftarrow \frac{N_{k'+1}^j \sqrt{N_{k'}^j} d_{k',0}^j - N_{k'}^j \sqrt{N_{k'+1}^j} d_{k'+1,0}^j}{\sqrt{N_{k'}^j (N_{k'+1}^j)^2 + N_{k'+1}^j (N_{k'}^j)^2}}$ `// Compute the Haar coefficient`
    end
    if $N_k^{j-1} > 2$ **then**
      for $l = 1, \ldots, 2^{j_{\max}-j} - 1$ **do** `// Compute the Walsh-like coefficients`
        if *both subregions* k' *and* $k'+1$ *have a basis vector with tag* l **then**
          $d_{k,2l}^{j-1} \leftarrow (d_{k',l}^j + d_{k'+1,l}^j)/\sqrt{2}$;
          $d_{k,2l+1}^{j-1} \leftarrow (d_{k',l}^j - d_{k'+1,l}^j)/\sqrt{2}$;
        else if *(without loss of generality) only subregion* k' *has a basis vector with tag* l **then**
          $d_{k,2l}^{j-1} \leftarrow d_{k',l}^j$
        else if *Neither subregion has a basis vector with tag* l **then**
          Do nothing
      end
    end
  end
end

Algorithms 1 and 2 can be performed simultaneously. In practice, when analyzing the input signal $\boldsymbol{f}$, we only need Algorithm 2 to compute the coefficients without computing the basis vectors explicitly.

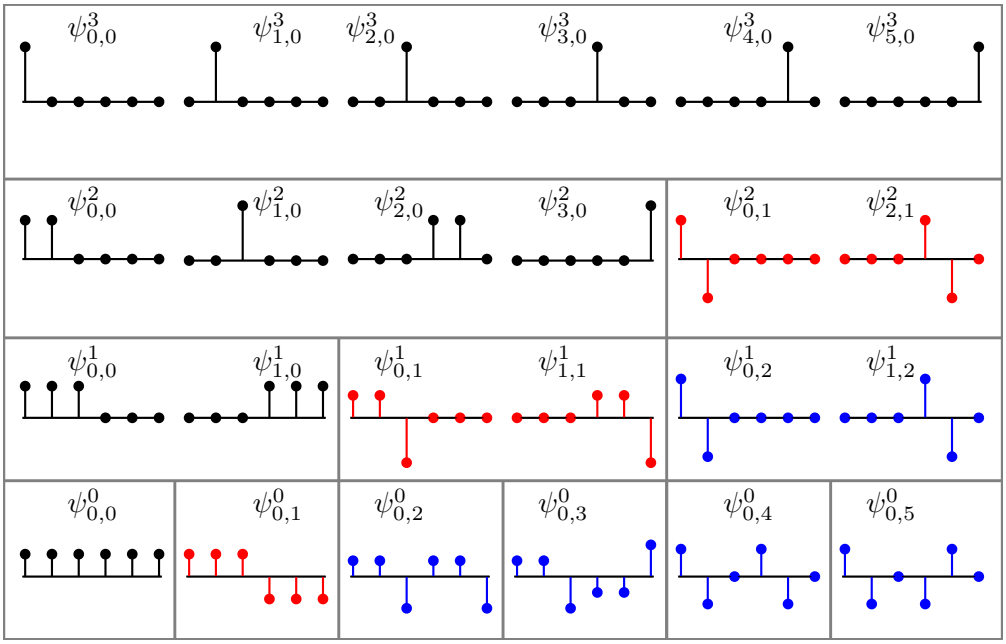
For the dictionary of basis vectors, several observations are in order.

- First, the basis vectors on each level are localized. In other words, $\boldsymbol{\psi}_{k,l}^{j}$ is supported on $V(G_k^j)$. If $V(G_k^j) \cap V(G_{k'}^{j'}) = \emptyset$, then the basis vectors $\{\boldsymbol{\psi}_{k,l}^{j}\}_l$ and $\{\boldsymbol{\psi}_{k',l'}^{j'}\}_{l'}$ are mutually orthogonal.
- Second, the basis vectors corresponding to G_k^j span the same linear subspace as the union of those corresponding to $G_{k'}^{j+1}$ and $G_{k'+1}^{j+1}$, where $G_{k'}^{j+1}$ and $G_{k'+1}^{j+1}$ are the two subgraphs of G_k^j.
- Third, the depth of the dictionary is the same as the binary partition tree, which is approximately $O(\log N)$ if the tree is nearly balanced. There are N vectors on each level, so the total number of basis vectors is approximately $O(N \log N)$.

In addition, we can arrange these basis vectors by region, which we call the *coarse-to-fine* (c2f) dictionary. It is the same as the result obtained from Algorithm 1. Or we can arrange them by tag, which we call the *fine-to-coarse* (f2c) dictionary [**30**, **31**, **33**]. The c2f dictionary corresponds to a collection of basis vectors by recursively partitioning the "time" domain information of the input graph signal while the f2c dictionary corresponds to those by recursively partitioning the "frequency" (or "sequency") domain information of the input graph signal. Each dictionary contains more than $(1.5)^N$ choosable ONBs; see, e.g., Thiele and Villemoes [**65**] for the details on this number. Note, however, that exceptions can occur when the recursive partitioning generates a highly imbalanced tree. Figure 3.1 shows the examples of these dictionaries for a simple path graph consisting of six nodes. Figure 3.2 shows examples of the vectors from the GHWT dictionary computed on the Toronto traffic dataset.

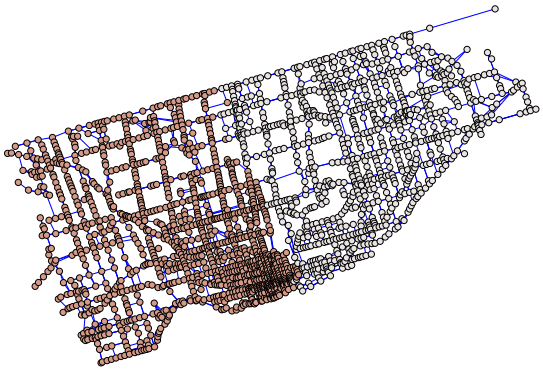


(a) The c2f GHWT dictionary

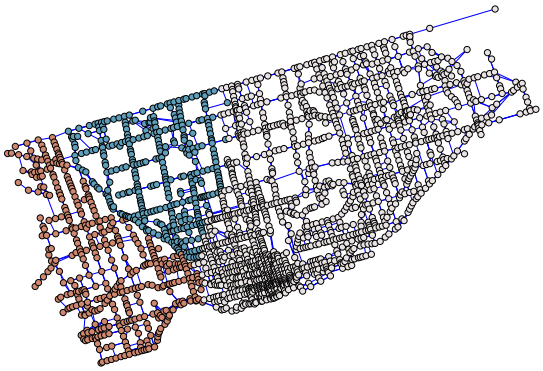


(b) The f2c GHWT dictionary

FIGURE 3.1. The c2f and f2c GHWT dictionaries on the simple path P_6. Stem plots with black, red, blue colors correspond to the scaling, Haar, and Walsh vectors, respectively.



(a) Scaling vector $\boldsymbol{\psi}_{0,0}^{1}$



(b) Haar vector $\boldsymbol{\psi}_{0,1}^{2}$

(c) Walsh vector $\boldsymbol{\psi}_{1,34}^{5}$

FIGURE 3.2. Examples of vectors from GHWT dictionary computed on Toronto traffic dataset

3.2. The Previous Best-Basis Algorithm in the GHWT

To select the basis from a dictionary of wavelet packets that is "best" for approximation/compression, a *best-basis algorithm* is required. The previous best-basis algorithm for the GHWT is a straightforward generalization of the Coifman-Wickerhauser algorithm [**13**], which was developed for non-graph signals of dyadic length. The algorithm requires a real-valued cost function $\mathcal{J}$, and aims to find the basis whose coefficients minimize $\mathcal{J}$. The algorithm initiates the best basis as the whole set of vectors at the bottom level of the dictionary. Then it proceeds upwards, comparing the cost of the expansion coefficients corresponding to two children subgraphs to the cost of those of their parent subgraph. The best basis is updated if the cost of the parent subgraph is smaller than that of its children subgraphs. The algorithm continues until it reaches the top (i.e., the root) of the binary partition tree (i.e., the dictionary). Algorithm 3 describes the details of this procedure.

Algorithm 3: The best-basis algorithm of Coifman-Wickerhauser [**13**] tailored for the GHWT [**30**, **31**, **33**]

Input: An overcomplete dictionary of basis vectors $\{\boldsymbol{\psi}_{k,l}^j\}$ from Algorithm 1, $0 \leq j \leq j_{\max}$ and $0 \leq k < K^j$ where K^j denotes the number of subgraphs on level j; an additive cost function $\mathcal{J}(\boldsymbol{d}) = \sum_{i=1}^N g(d_i)$; an input graph signal $\boldsymbol{f} \in \mathbb{R}^N$.

Output: The best basis $B(G)$; the minimal cost $A(G)$.

for $k = 0, \ldots, N-1$ **do** `// Initialize the best basis on level` $j_{\max}$
  $B(G_k^{j_{\max}}) \leftarrow \{\boldsymbol{\psi}_{k,0}^{j_{\max}}\}$;
  $A(G_k^{j_{\max}}) = g(\langle \boldsymbol{\psi}_{k,0}^{j_{\max}}, \boldsymbol{f} \rangle)$
end
for $j = j_{\max} - 1, \ldots, 0$ **do** `// Update the best basis upwards`
  for $k = 0, \ldots, K^j$ **do**
    $B(G_k^j) \leftarrow \{\boldsymbol{\psi}_{k,l}^j\}$;
    $A(G_k^j) \leftarrow \sum_l g(\langle \boldsymbol{\psi}_{k,l}^j, \boldsymbol{f} \rangle)$;
    if G_k^j *is split into two subgraphs* $G_{k'}^{j+1}$ *and* $G_{k'+1}^{j+1}$ **then**
      if $A(G_k^j) \geq A(G_{k'}^{j+1}) + A(G_{k'+1}^{j+1})$ **then**
        $B(G_k^j) \leftarrow B(G_{k'}^{j+1}) \cup B(G_{k'+1}^{j+1})$;
        $A(G_k^j) \leftarrow A(G_{k'}^{j+1}) + A(G_{k'+1}^{j+1})$
      end
    end
  end
end
$B(G) \leftarrow B(G_0^0)$;
$A(G) \leftarrow A(G_0^0)$;

The c2f and f2c dictionaries are searched separately to obtain two sets of the best bases, among which the one with smaller cost is chosen as the final best basis of the GHWT dictionaries. We note here that the graph Haar basis is selectable *only in the f2c dictionary* while the graph Walsh basis is selectable in either dictionary.

In the GHWT case, the algorithm works as long as $\mathcal{J}$ is nonnegative and additive [1] of the form $\mathcal{J}(\boldsymbol{d}) := \sum_i g(d_i)$ with $g : \mathbb{R} \to \mathbb{R}_{\geq 0}$, where $\boldsymbol{d}$ is the expansion coefficients of an input graph signal on each region. For example, if one wants to promote sparsity in graph signal representation or approximation, $\mathcal{J}(\boldsymbol{d})$ function can be chosen as: either the p-th power of ℓ^p-(quasi)norm $\sum_i |d_i|^p$ for $0 < p < 2$ or the ℓ^0-pseudonorm $|\{d_i \,|\, d_i \neq 0\}|$. Note that the smaller the value of p is, the more emphasis in sparsity is placed. Here the 2-norm is not used, since for each orthonormal basis, the 2-norm is the same as that of input signal. The Shannon entropy of the expansion coefficients can be used as well. Of course, the choice of cost functional depends on the task at hand. For example, the 1-norm is often used to promote sparsity.

[1] The additivity property can be dropped in principle by following the work of Saito and Coifman on the local regression basis [**57**]

CHAPTER 4

The extended Generalized Haar-Walsh Transform (eGHWT)

In this chapter, we describe the *extended* GHWT, i.e., our new best-basis algorithm on the GHWT dictionaries, which *simultaneously* considers the "time" domain split *and* "frequency" (or "sequency") domain split of an input graph signal. This transform will allow us to deploy the modified best-basis algorithm that can select the best ONB for one's task (e.g., efficient approximation, denoising, etc.) among a much larger set ($> 0.618 \cdot (1.84)^N$) of ONBs than the c2f and f2c GHWT dictionaries would provide ($> (1.5)^N$). The previous best-basis algorithm only searches through the c2f dictionary and f2c dictionary separately, but this new method makes use of those two dictionaries simultaneously.

Thiele and Villemoes [**65**] proposed an algorithm to find the best basis among the ONBs of $\mathbb{R}^N$ consisting of discretized rescaled Walsh functions for an input 1D (non-graph) signal of length N, where N must be a dyadic integer. Their algorithm operates in the time-frequency plane by constructing a tiling of minimal cost among all possible tilings with dyadic rectangles of area one. Here we adapt their method to our graph setting that does not have to be dyadic. In addition, its generalization for 2D signals developed by Lindberg and Villemoes [**47**] can be generalized to the 2D eGHWT, as we will discuss more in Section 4.6 and Section 5.2.

4.1. Fast Adaptive Time-Frequency Tilings

In this section, a brief description of Thiele-Villemoes algorithm [**65**] is given. Let $W_0(t) = 1$ for $0 \leq t < 1$ and zero elsewhere, and define W_1,W_2,... recursively by

$$\begin{aligned} W_{2l}(t) &= W_l(2t) + (-1)^l W_l(2t-1), \\ W_{2l+1}(t) &= W_l(2t) - (-1)^l W_l(2t-1). \end{aligned} \tag{4.1}$$

Then $\{W_l\}_{l=0}^{\infty}$ is *the Walsh system in sequency order*. It is an orthonormal basis for $L^2(0,1)$ and each basis function is piecewise equal to either 1 or -1 on [0, 1). The scaling and Haar vectors are included in this Walsh system.

Viewing $S := [0,1) \times [0,\infty)$ as a time-frequency plane, the *dyadic rectangle* corresponding to the rescaled Walsh function

$$w_p(t) := 2^{j/2} W_l(2^j t - k)$$

is defined as

$$p := [2^{-j}k, 2^{-j}(k+1)) \times [2^j l, 2^j(l+1)). \tag{4.2}$$

Then, we have the following theorems:

THEOREM 4.1.1. *The functions w_p and w_q are orthogonal if and only if the tiles p and q are disjoint.*

THEOREM 4.1.2. *Let $T = I \times \omega$ be a dyadic rectangle of area $|T| = |I||\omega| \geq 1$, and assume that $\mathcal{B}$ and $\mathcal{B}'$ are two collections of tiles, both defining disjoint coverings of T. Then $\{w_p \,|\, p \in \mathcal{B}\}$ and $\{w_p \,|\, p \in \mathcal{B}'\}$ are orthonormal bases of the same subspace of $L^2(0,1)$.*

According to the paper [**65**], the discrete signal space $\mathbb{R}^N$ $(N = 2^n)$ can be identified with the subset $S_n := [0,1) \times [0,N)$ of S in the time-frequency plane. Given the overcomplete dictionary of Haar-Walsh functions on $\mathbb{R}^N$, the best-basis algorithm now is equivalent to finding a set of basis vectors with minimal cost and their tiles form a disjoint tiling of S_n. That tiling is called the *minimizing tiling.*

LEMMA 4.1.3. *Let $T \subset S$ be a rectangle of area greater or equal to two, with left half L, right half R, lower half D, and upper half U. Assume each tile $p \subset T$ has the cost $c(p)$. Define*

$$m_T := \min\{\sum_{p \in \mathcal{B}} c(p) \,|\, \mathcal{B} \text{ is a disjoint covering of } T\}$$

and similarly m_L,m_R,m_D,m_U. Then

$$m_T = \min\{m_L + m_R, m_D + m_U\}.$$

This lemma tells us that the minimizing tiling of T can be split either in the time-direction into two minimizing tilings of L and R respectively, or in the frequency-direction into those of D and U respectively. It enables dynamic programming algorithm (Algorithm 4) to find the minimizing tiling of S_n.

Algorithm 4: Fast Time-Frequency Tilings [**65**]

Input: All tiles contained in S_n. A cost function $c(p)$ defined on each tile p.
Output: The minimizing tiling $B(S_n)$
Compute the cost $c(p)$ of all tiles contained in S_n. And initialize $B(p) = p$ for every p. **for** $m = 1, \ldots, n$ **do**
  for *all dyadic rectangles* $P \subset S_n$ *of area* 2^m **do**
    Check the left half P_L, right half P_R, upper half P_U and lower half P_D of P, **if** $c(P_L) + c(P_R) \geq c(P_U) + c(P_D)$ **then**
      $B(P) \leftarrow B(P_U) \cup B(P_D)$
      $c(P) \leftarrow c(P_U) + c(P_D)$
    else
      $B(P) \leftarrow B(P_L) \cup B(P_R)$
      $c(P) \leftarrow c(P_L) + c(P_R)$
    end
  end
end
Finally $B(S_n)$ is a minimizing tiling of S_n with the minimal cost $c(S_n)$.

A simple example is the easiest way to understand this algorithm. Consider the signal $\boldsymbol{f} = (3, 1, 2, 2, 3, 1, 1, 3) \in \mathbb{R}^8$. The time-frequency plane is $S_3 = [0, 1) \times [0, 8)$, and the frequency axis is scaled so that S_3 is a square in the Figure 4.1 for visualization purpose. Here we use the cost function $\mathcal{J}(\boldsymbol{d}) = \sum_i |d_i|$, which means the cost of a tile p is $c(p) = |\langle \boldsymbol{f}, \boldsymbol{w_p} \rangle|$. The collection of all 32 tiles and the corresponding coefficients are placed on four copies of S_3 in the top row of squares of Figure 4.1. They are all of size 1. More specifically, the basis vectors corresponding to the tiles are (from left to right in the top row of the figure)

- Eight unit vectors.
- $\frac{1}{\sqrt{2}} \times$

$(1, -1, 0, 0, 0, 0, 0, 0)$	$(0, 0, 1, -1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, -1, 0, 0)$	$(0, 0, 0, 0, 0, 0, 1, -1)$
$(1, 1, 0, 0, 0, 0, 0, 0)$	$(0, 0, 1, 1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, 1, 0, 0)$	$(0, 0, 0, 0, 0, 0, 1, 1)$

- $\frac{1}{2} \times$

$(1, -1, -1, 1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, -1, -1, 1)$
$(1, -1, 1, -1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, -1, 1, -1)$
$(1, 1, -1, -1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, 1, -1, -1)$
$(1, 1, 1, 1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 1, 1, 1, 1)$

•

$$\frac{1}{2\sqrt{2}} \times \begin{array}{|c|}\hline (1, -1, -1, 1, -1, 1, 1, -1) \\ \hline (1, -1, -1, 1, 1, -1, -1, 1) \\ \hline (1, -1, 1, -1, -1, 1, -1, 1) \\ \hline (1, -1, 1, -1, 1, -1, 1, -1) \\ \hline (1, 1, -1, -1, -1, -1, 1, 1) \\ \hline (1, 1, -1, -1, 1, 1, -1, -1) \\ \hline (1, 1, 1, 1, -1, -1, -1, -1) \\ \hline (1, 1, 1, 1, 1, 1, 1, 1) \\ \hline \end{array}$$

In the first iteration ($m = 1$) in Figure 4.1[1], the minimizing tilings for all the dyadic rectangles of size 2 are computed from those with size 1. For example, for the left most dyadic rectangle with cost 4 at $m = 1$. It can be composed by the two tiles of $(1, 0, 0, 0, 0, 0, 0, 0)$ and $(0, 1, 0, 0, 0, 0, 0, 0)$, or $\frac{1}{\sqrt{2}}(1, -1, 0, 0, 0, 0, 0, 0)$ and $\frac{1}{\sqrt{2}}(1, 1, 0, 0, 0, 0, 0, 0)$. The cost are $|3| + |1|$ or $|\sqrt{2}| + |2\sqrt{2}|$, the minimal of which is 4. The minimizing tiling for that dyadic rectangle is $(1, 0, 0, 0, 0, 0, 0, 0)$ and $(0, 1, 0, 0, 0, 0, 0, 0)$ with cost 4. In the second iteration, the algorithm finds the minimizing tilings for all dyadic rectangles with size 4. In the third iteration, the algorithm finds the minimizing tiling for the whole S_3.

Note that the Coifman-Wickerhauser best-basis algorithm considers only the split of best basis in the time direction. The Thiele-Villemoes best-basis algorithm considers the split in both the time and frequency direction in the special case of Walsh functions.

[1]Reprinted from Applied and Computational Harmonic Analysis, Christoph M. Thiele and Lars F. Villemoes, *A fast algorithm for adapted time–frequency tilings*, Page 96, Copyright (2020), with permission from Elsevier.

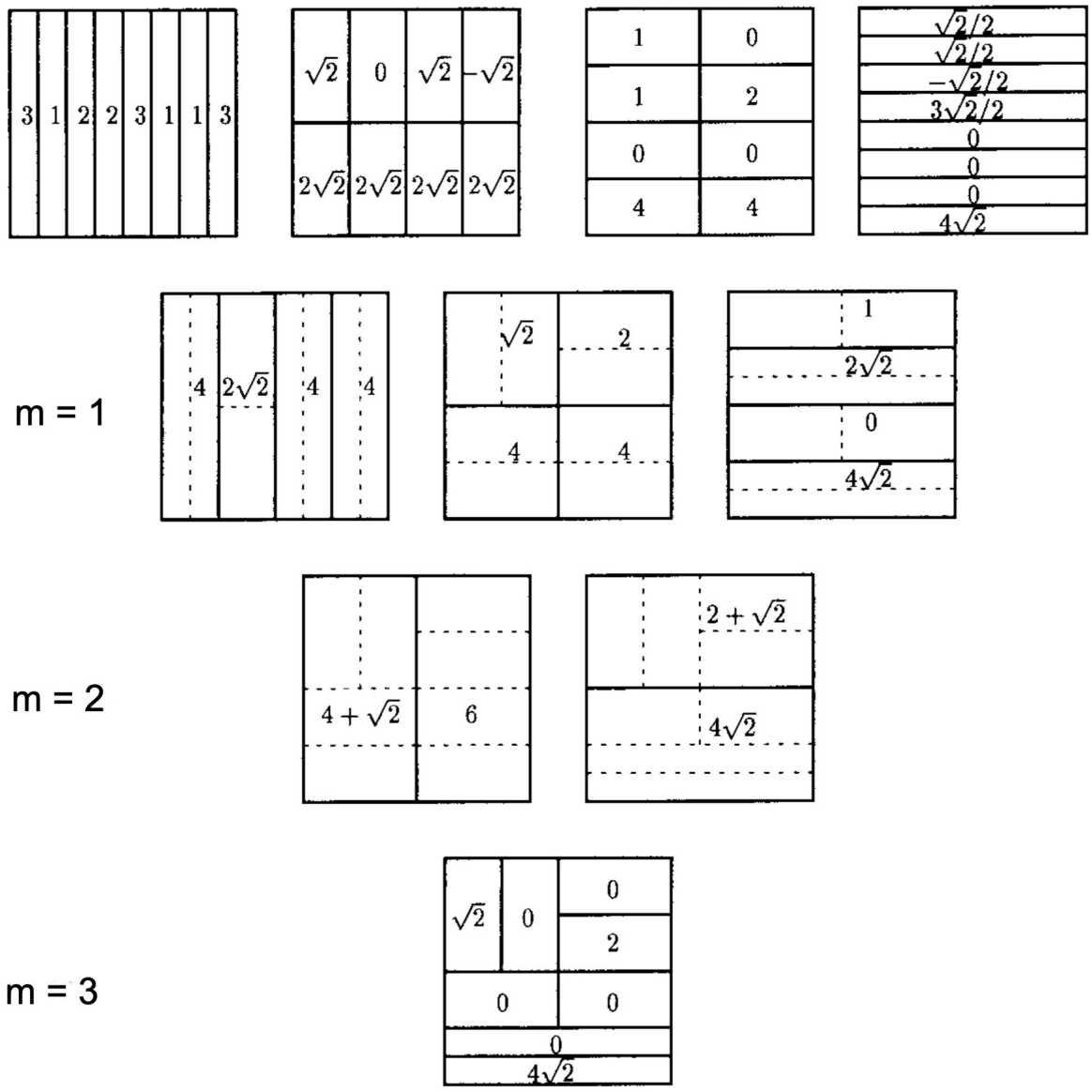


FIGURE 4.1. Graphical representation of Algorithm 4 for the simple signal in $\mathbb{R}^8$ The cost function is chosen to be the sum of absolute values of the expansion coefficients. The top row contains all tiles and coefficients. The bottom row represents the best basis. (This figure is from [**65**].)

4.2. Numerical Analysis

We point out an error in the time cost analysis of Algorithm 4 in the original paper [**65**], which does not affect the final result. There are $(n-m+1)2^{n-m}$ dyadic subrectangles of S_n with area 2^m, instead of $(n-m+1)2^{n-1}$ in the paper. So the mth iteration involves $(n-m+1)2^{n-m}$ times two additions and one comparison. This gives a total of

$$3\sum_{m=1}^{n}(n-m+1)2^{n-m} = 3(n+1)2^n.$$

Therefore the search for the best basis takes around $3(\log_2 N + 1)N$ operations. Adding the $O(N\log_2 N)$ operations to compute all the expansion coefficients on the whole dictionary, the whole process of representing $\boldsymbol{f}$ with the best basis takes $O(N\log N)$ operations.

According to [**65**], the number of ONBs searched for $\mathbb{R}^N$ is between $0.618(1.84)^N$ and $0.618(1.85)^N$. The sketch of the proof is as follows. Denote the number of searchable ONBs for $\mathbb{R}^N$ $(N = 2^n)$ in their algorithm as a_n. Then a_n is equal to the number of tilings of a dyadic rectangle T of area 2^n with dyadic tiles of area 1. Each tiling can be split vertically or horizontally, and the subtilings have a_{n-1} possibilities. Therefore, there are totally $2a_{n-1}^2$ possibilities for tilings of T, except for the doubly counted tilings which can be split vertically and horizontally. The doubly counted tilings can be split into four subtilings, each with a_{n-2} possibilities. Therefore there are a_{n-2}^4 doubly counted tilings, so we have

$$a_n = 2a_{n-1}^2 - a_{n-2}^4 \ (n \geq 2).$$

With $a_0 = 1, a_1 = 2$, it can be verified by induction that $0.618(1.84)^{2^n} \leq a_n \leq 0.618(1.85)^{2^n}$.

4.3. Relabeling Region Indices

When the input signal is dyadic and the partition tree is a balanced complete binary tree, the GHWT dictionary is the same as regular Haar-Walsh wavelet packet dictionary, on which the Thiele-Villemoes algorithm [**65**] can be applied in a straightforward manner. To adapt the algorithm to a graph signal with an arbitrary size or an unbalanced incomplete binary partition tree, we need to modify the GHWT dictionary first.

Specifically, the region index k of G_k^j and $\boldsymbol{\psi}_{k,l}^j$ needs to be *relabeled*. Previously, on level j, the region index k takes all the integer values in $[0, K^j)$ where K^j is the total number of subgraphs (or regions) on level j. After relabeling, k takes an integer values in $[0, 2^j)$ according to its location in the binary tree. The whole procedure is described by Algorithm 5. Then the region indices of the basis vectors $\{\boldsymbol{\psi}_{k,l}^j\}$ supported on the subgraph G_k^j are also relabeled accordingly.

Algorithm 5: Relabeling the GHWT Dictionary

Input: A binary partition tree denoted by $\{G_k^j\}$, $0 \le k < K^j$, $0 \le j < j_{\max}$, K^j denotes the number of subgraphs on level j

Output: The same binary partition tree $\{G_k^j\}$ with region index k relabeled

`// On level 0, there is only one region` G_0^0`, so no relabeling is required.`

for $j = 1, \ldots, j_{\max}$ **do**

  if G_k^{j-1} *is split into* $G_{k'}^j$ *and* $G_{k'+1}^j$ **then**

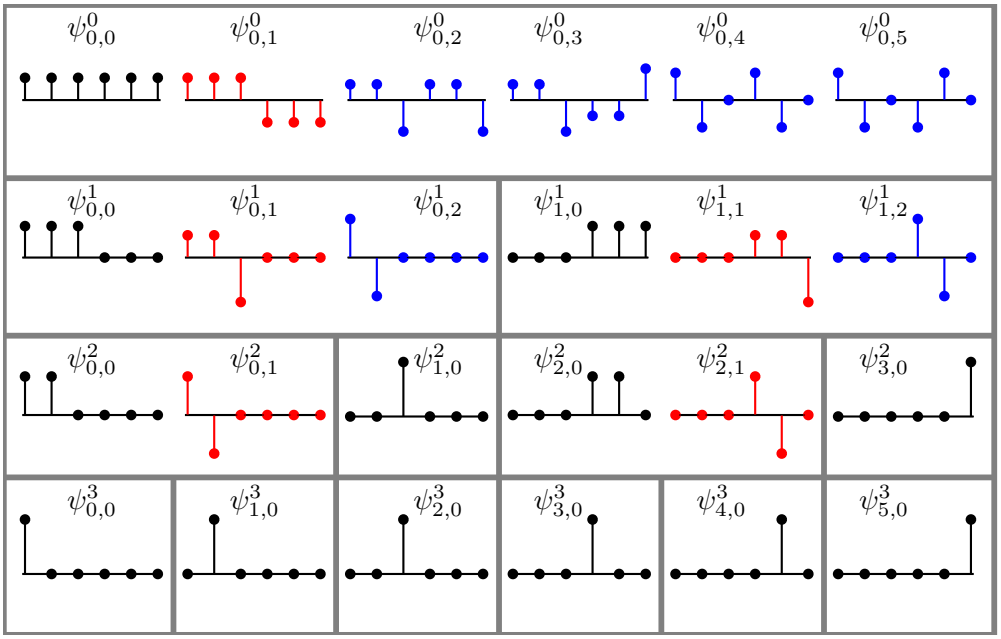
    The two subgraphs are relabeled as G_{2k}^j and G_{2k+1}^j

  else if G_k^{j-1} *is kept as* $G_{k'}^j$ **then** `// the subgraph contains only one node.`

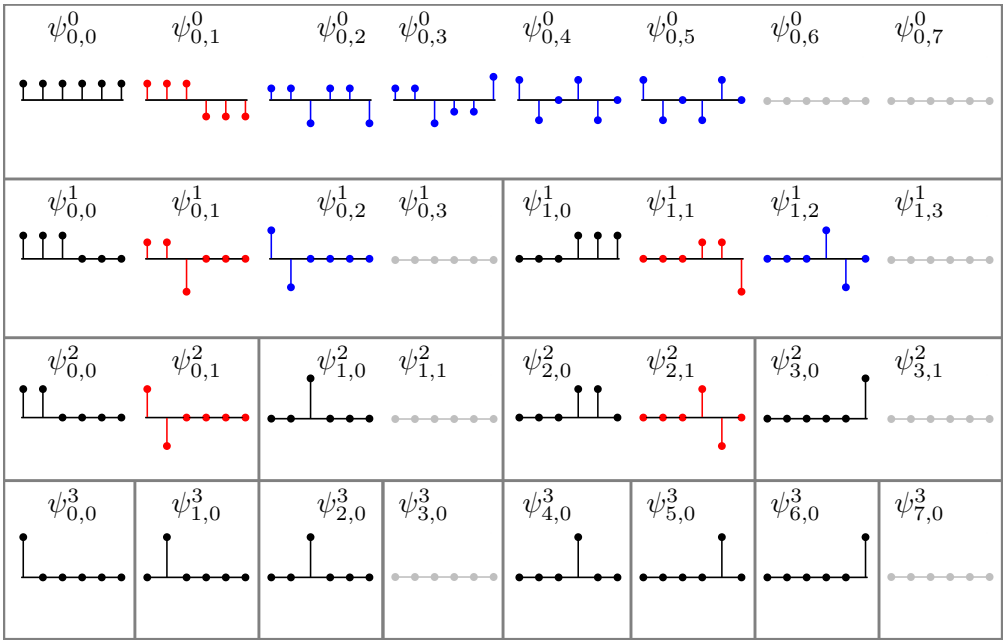
    The subgraph is relabeled as G_{2k}^j

end

Figure 4.2b shows the result of Algorithm 5 applied to the c2f GHWT dictionary shown in Figure 4.2a on a simple path graph with $N = 6$. Before the relabeling, the dictionary forms an unbalanced binary tree. After the relabeling, the dictionary is a subset of a perfect binary tree.



(a) The c2f GHWT dictionary



(b) The relabeled c2f GHWT dictionary

FIGURE 4.2. (a) The c2f GHWT dictionary on the simple path P_6. Stem plots with black, red, blue colors correspond to the scaling, Haar, and Walsh vectors, respectively. (b) The relabeled c2f GHWT dictionary by Algorithm 5 applied to the c2f GHWT dictionary shown in (a). The gray stem plots indicate the "fictitious" (or "non-existent") nodes newly generated by Algorithm 5.

4.4. The New Best-Basis (eGHWT) Algorithm

We can now apply Algorithm 6 to search for the best basis in the new GHWT dictionary. This whole procedure is called the *eGHWT*.

Algorithm 6: The New Best-Basis (eGHWT) Algorithm

Input: The dictionary of basis vectors $\{\boldsymbol{\psi}_{k,l}^{j}\}$ from Algorithm 5; an additive cost function $\mathcal{J}(\boldsymbol{c}) = \sum_{i=1}^{N} g(c[i])$; an input graph signal $\boldsymbol{f} \in \mathbb{R}^N$.

Output: The best basis B.

Initialize the associative array[1] A_0 as the expansion coefficients[2] of the dictionary evaluated through $g(\cdot)$, i.e., $A_0[j,k,l] = g(\langle \boldsymbol{\psi}_{k,l}^{j}, \boldsymbol{f} \rangle)$;

Initialize another associative array I_0 by $I_0[j,k,l] = 1$ if $(j,k,l) \in A_0$;

for $m = 0, \ldots, j_{\max} - 1$ **do** `// Compute` A_{m+1} `and` I_{m+1}

  Initialize A_{m+1} and I_{m+1} as empty associative arrays;

  for $(j,k,l) \in A_m$ *with* $j < j_{\max}$ *and* $l \equiv 0 \pmod 2$ **do**

    if $(A_m[j,k,l] + A_m[j,k,l+1]) \leq (A_m[j+1,2k,l/2] + A_m[j+1,2k+1,l/2])$ **then**

      $I_{m+1}[j,k,l/2] \leftarrow 0$;

      $A_{m+1}[j,k,l/2] \leftarrow (A_m[j,k,l] + A_m[j,k,l+1])$;

    else

      $I_{m+1}[j,k,l/2] \leftarrow 1$;

      $A_{m+1}[j,k,l/2] \leftarrow (A_m[j+1,2k,l/2] + A_m[j+1,2k+1,l/2])$;

    end

    `// Any of` $A_m[j,k,l+1], A_m[j+1,2k,l/2], A_m[j+1,2k+1,l/2]$ `will be replaced by` 0 `in the above steps if it does not exist.`

  end

end

$I \leftarrow I_{j_{\max}}$;

for $m = j_{\max} - 1, \ldots, 0$ **do** `// Recover the best basis from` $\{I_m\}$

  Initialize I_{temp} as an empty associative array;

  for $(j,k,l) \in I$ **do**

    if $I[j,k,l] = 0$ **then**

      $I_{\text{temp}}[j,k,l] \leftarrow I_m[j,k,2l]$ if $(j,k,2l) \in I_m$;

      $I_{\text{temp}}[j,k,l] \leftarrow I_m[j,k,2l+1]$ if $(j,k,2l+1) \in I_m$;

    else

      $I_{\text{temp}}[j+1,2k,l] \leftarrow I_m[j+1,2k,l]$ if $(j+1,2k,l) \in I_m$;

      $I_{\text{temp}}[j+1,2k+1,l] \leftarrow I_m[j+1,2k+1,l]$ if $(j+1,2k+1,l) \in I_m$;

    end

  end

  $I \leftarrow I_{\text{temp}}$;

end

$B \leftarrow \{\}$; `// Initialize` B `as an empty set`

Add $\boldsymbol{\psi}_{k,l}^{j}$ into B if $(j,k,l) \in I$. `//` B `is the best basis`

Several remarks on this algorithm are in order:

- $A_m[j,k,l]$ is the minimal cost of ONBs in a linear subspace. Generally, $A_m[j,k,l]$ is computed from the minimal of $A_{m-1}[j,k,2l] + A_{m-1}[j,k,2l+1]$ and $A_{m-1}[j+1,2k,l] + A_{m-1}[j+1,2k+1,l]$. The linear subspace of $A_m[j,k,l]$ is the direct sum of the two linear subspaces corresponding to $A_{m-1}[j,k,2l]$ and $A_{m-1}[j,k,2l+1]$, which is the same as the direct sum of those corresponding to $A_{m-1}[j+1,2k,l]$ and $A_{m-1}[j+1,2k+1,l]$. In other words, when we compute A_m from A_{m-1}, we are concatenating linear subspaces. This process is similar to finding the best tilings for dyadic rectangles from those with half the size in Thiele-Vellemoes algorithm [**65**] as described in Section 4.1.
- The linear subspace of each entry in A_0 is one dimensional since it is spanned by a single basis vector. In other words, $A_0[j,k,l]$ corresponds to the linear subspace spanned by $\boldsymbol{\psi}_{k,l}^j$.
- $A_{j_{\max}}$ has only one entry $A_{j_{\max}}[0,0,0]$, which corresponds to the whole $\mathbb{R}^N$. Its value is the minimal cost of all the ONBs, i.e., the cost of the best basis.
- If the signal is of dyadic length, then $(A_{m-1}[j,k,2l], A_{m-1}[j,k,2l+1])$ corresponds to splitting the subspace of $A_m[j,k,l]$ in the "frequency" domain in the "time-frequency plane" while $(A_{m-1}[j+1,2k,l], A_{m-1}[j+1,2k+1,l])$ does the split in the "time" domain.
- If the signal is of dyadic length, the eGHWT can select among a much larger set ($> 0.618 \cdot (1.84)^N$) of ONBs than what each of the c2f and f2c GHWT dictionaries would provide ($> (1.5)^N$) [**65**]. The numbers are similar even for non-dyadic cases as long as the partition trees are essentially balanced. The essence of this algorithm is that at each step of the recursive evaluation of the costs of subspaces, it compares the cost of the parent subspace with not only its two children subspaces partitioned in the "frequency" domain (the f2c GHWT does this), but also its two children subspaces partitioned in the "time" domain (the c2f GHWT does this).
- When implementing the algorithm, *associate arrays*[1] are used to store the costs $\{A_m\}$ and indices $\{I_m\}$. Since many $\boldsymbol{\psi}_{k,l}^j$ may be *fictitious*, which we do not need to store or

[1] An *associative array* is an abstract data type composed of a collection of (key, value) pairs such that each possible key appears at most once in the collection.

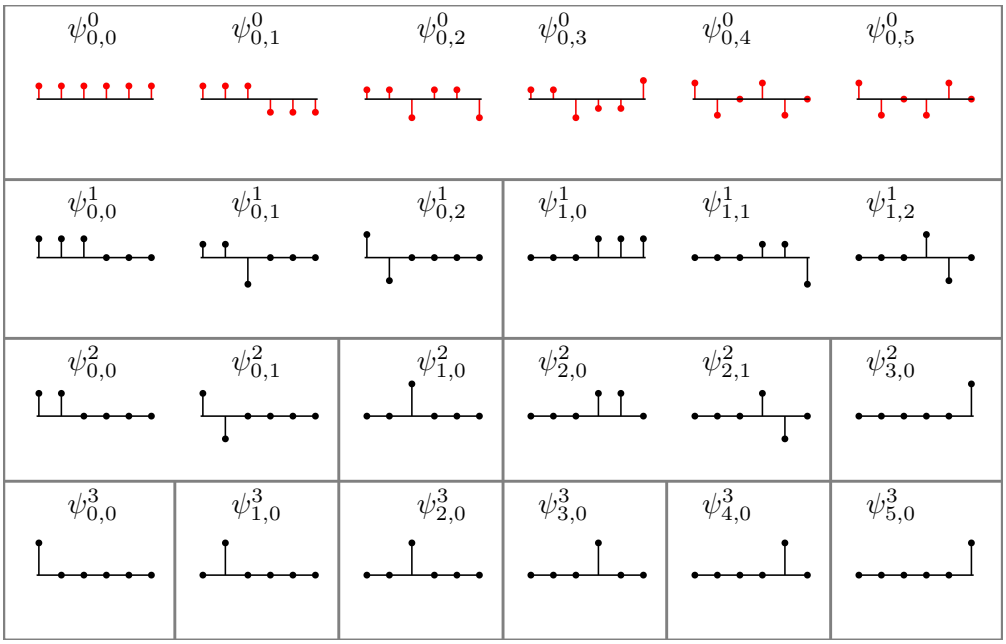
[2] $((j,k,l), g(\langle \boldsymbol{\psi}_{k,l}^j, \boldsymbol{f} \rangle))$ is a pair of (key, value) of the associative array A_0. Here we use $(j,k,l) \in A_0$ to denote that (j,k,l) is a valid key of A_0. Therefore, $(j,k,l) \in A_0$ if and only if $\boldsymbol{\psi}_{k,l}^j$ exists. Since we relabeled $\boldsymbol{\psi}_{k,l}^j$, there is no corresponding $\boldsymbol{\psi}_{k,l}^j$ for some triple (j,k,l). In that case, $(j,k,l) \notin A_0$.

compute, using regular matrices to store them will be wasteful. On the other hand, due to the various range of j, k, l, using associative arrays give us more flexibility and efficiency than using sparse matrices.

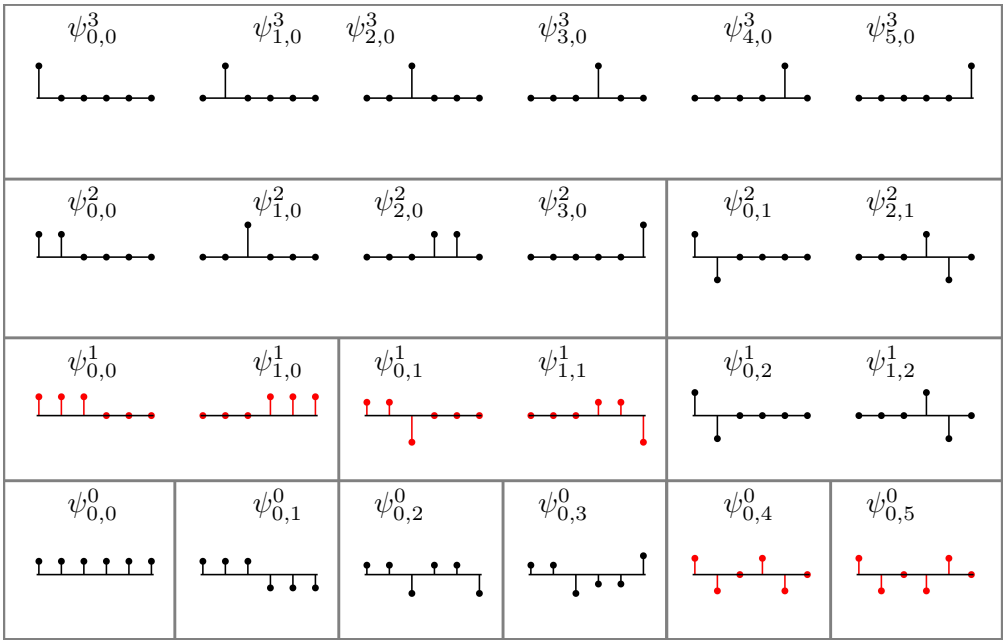
4.5. The eGHWT illustrated by a graph signal on P_6

Let $\boldsymbol{f} = [2, -2, 1, 3, -1, -2]^{\mathsf{T}} \in \mathbb{R}^6$ be an example graph signal on the simple 6-node path P_6 to analyze. The ℓ^1-norm is chosen as the cost function. Figure 4.3 shows that the c2f-GHWT best basis is actually the Walsh basis, and its representation is $\frac{\sqrt{6}}{6}\boldsymbol{\psi}^0_{0,0} + \frac{\sqrt{6}}{6}\boldsymbol{\psi}^0_{0,1} + \frac{2\sqrt{3}}{3}\boldsymbol{\psi}^0_{0,2} + \frac{4\sqrt{3}}{3}\boldsymbol{\psi}^0_{0,3} + 4\boldsymbol{\psi}^0_{0,4} + 0\boldsymbol{\psi}^0_{0,5}$ with cost ≈ 8.28 and the f2c-GHWT best basis representation is $\frac{\sqrt{3}}{3}\boldsymbol{\psi}^1_{0,0} + 0\boldsymbol{\psi}^1_{1,0} + \frac{\sqrt{6}}{3}\boldsymbol{\psi}^1_{0,1} + \sqrt{6}\boldsymbol{\psi}^1_{1,1} + 4\boldsymbol{\psi}^0_{0,4} + 0\boldsymbol{\psi}^0_{0,5}$ with cost ≈ 7.84 while Figure 4.4 demonstrates that the best basis representation chosen by the eGHWT is $0\boldsymbol{\psi}^2_{0,0} + 1\boldsymbol{\psi}^2_{1,0} + 0\boldsymbol{\psi}^1_{1,0} + \sqrt{6}\boldsymbol{\psi}^1_{1,1} + 4\boldsymbol{\psi}^0_{0,4} + 0\boldsymbol{\psi}^0_{0,5}$ with cost ≈ 7.45, which is the smallest among these three best-basis representations. The indices used here are before relabeling for the illustration purpose.

To furthur illustrate the difference between the eGHWT and c2f/f2c GHWTs using this example, let us consider the vectors $\psi^1_{1,0}$ and $\psi^0_{0,4}$ in the eGHWT best basis. From Figure 4.4a, we can see that $\psi^1_{1,0}$ is supported on the child graph that was generated by bipartitioning the input graph where $\psi^0_{0,4}$ is supported. Therefore, they cannot be selected in the c2f-GHWT best basis simultaneously. A similar argument applies to $\psi^2_{1,0}$ and $\psi^1_{1,0}$ in the eGHWT best basis: they cannot be selected in the f2c-GHWT best basis simultaneously.

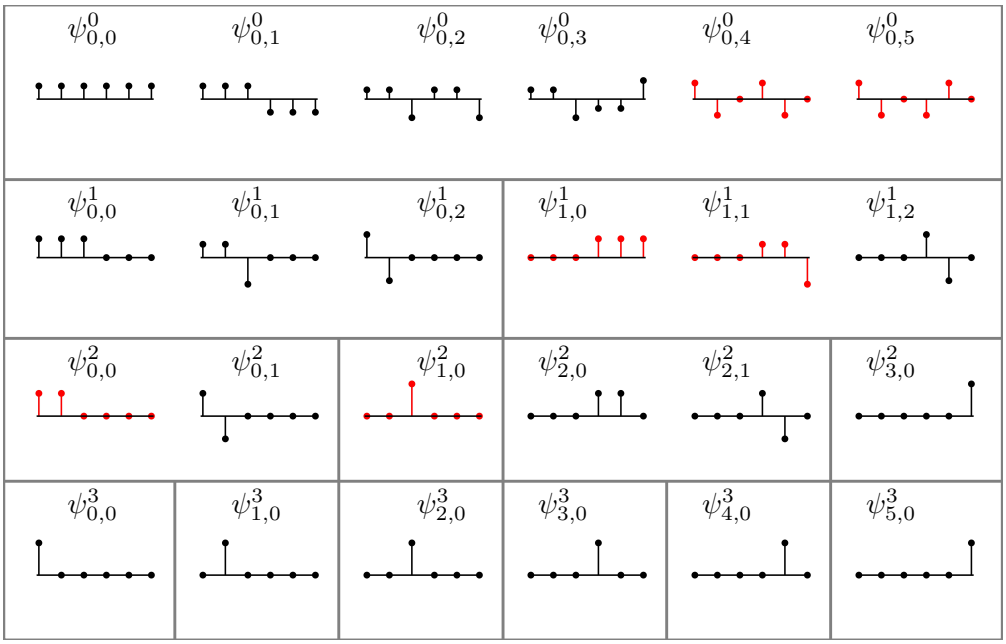


(a) coarse-to-fine dictionary

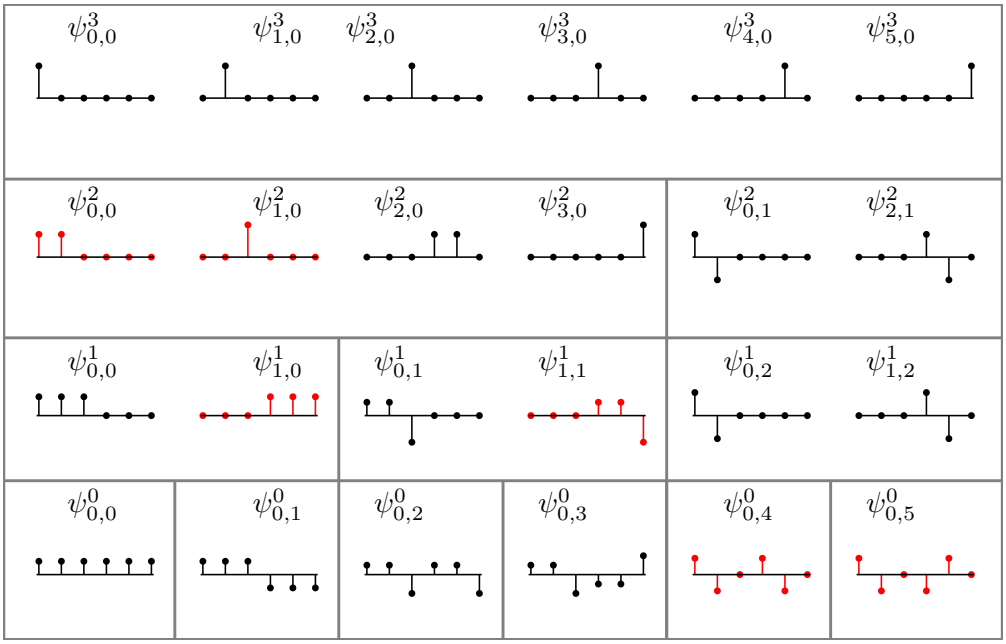


(b) fine-to-coarse dictionary

FIGURE 4.3. The c2f (a) and f2c (b) GHWT dictionaries for P_6. The basis vectors are grouped by region in (a) and by tag in (b). The best basis vectors obtained by the GHWT best-basis algorithm (Algorithm 3) in each dictionary for the input signal $\boldsymbol{f} = [2, -2, 1, 3, -1, -2]^\mathsf{T}$ are indicated by red.



(a) coarse-to-fine



(b) fine-to-coarse

FIGURE 4.4. The best basis vectors for the signal $\boldsymbol{f} = [2, -2, 1, 3, -1, -2]^\mathsf{T}$, selected by the eGHWT (Algorithm 6) are indicated by red in the c2f GHWT dictionary (a) and the f2c GHWT dictionary (b). Note the orthogonality of these vectors.

4.6. Generalization to 2D Signals

The Thiele-Villemoes algorithm [**65**], has been extended to 2D signals by Linderberg and Villemoes [**47**]. As we described in Section 4.1, the best tiling chooses a bipartition with a smaller cost from that of the time domain and that of the frequency domain. For 2D signals, the time-frequency domain has four axes instead of two. It has time and frequency axes on the $x-$(or column-) and $y-$(or row-) component. The best tiling comes from the split in the time or frequency directions in either the $x-$ or $y-$component. This gives us four options instead of two for each split. Similarly to the 1D signal case, dynamic programming is used to find the minimizing tiling for a given 2-D signal.

For a 2D signal, we can compose the affinity matrices on the row and column directions separately, thus define graphs that the rows and columns are supported on. In this way, the 2D signal can be viewed as a tensor product of two graphs. Then eGHWT can be extended to 2D signal from 1D in a similar way as how [**47**] extends [**65**]. Examples will be given in Section 5.2.

CHAPTER 5

Efficient Graph Signal Approximation using the eGHWT

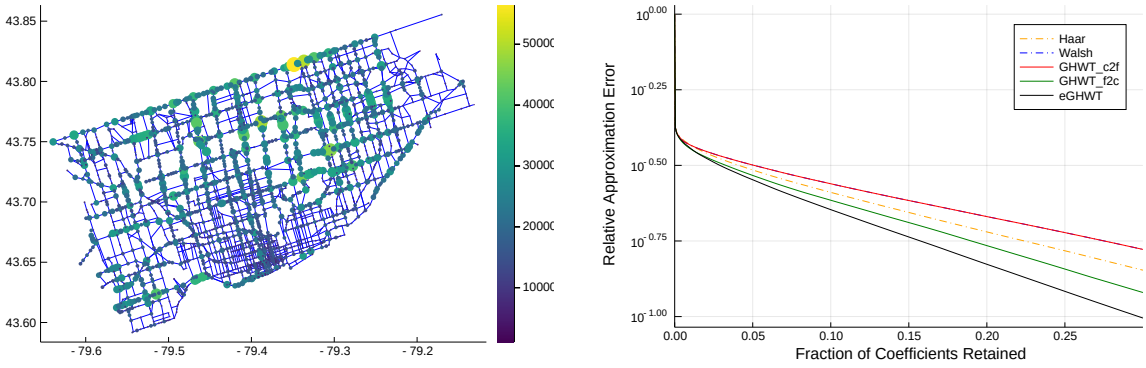
In this section, we will demonstrate the usefulness and efficiency of the eGHWT in graph signal approximation using several real datasets.

5.1. Efficient Approximation of Graph Signals

Here we analyze the vehicular traffic volume data on the Toronto road network mentioned in Chapter 2. In addition to the eGHWT best basis, the graph Haar basis, the graph Walsh basis, the c2f GHWT best basis and the f2c GHWT best basis are used to compare the performance. Figure 5.1b shows the performance comparison. The y-axis denotes the relative approximation error $\|\boldsymbol{f} - \mathcal{P}_n \boldsymbol{f}\|_2 / \|\boldsymbol{f}\|_2$, where $\mathcal{P}_n \boldsymbol{f}$ denotes the approximation of $\boldsymbol{f}$ with the basis vectors having the n largest coefficients in magnitude. The x-axis denotes n/N, i.e., the fraction of coefficients retained. We can see that the error of the eGHWT decays fastest.

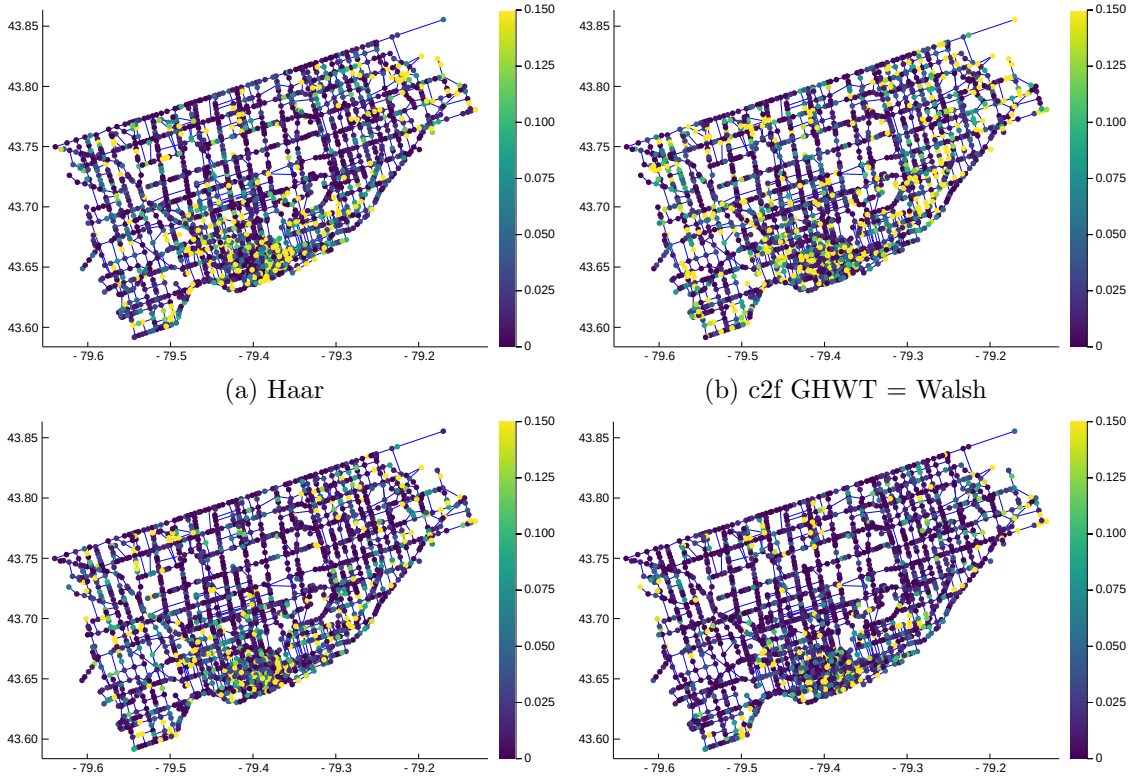
In Figure 5.2, we display the residual plots when 25% coefficients are used. We can clearly see that the eGHWT plot is much closer to 0 than the rest of the methods.

To emphasize the argument of the superiority of the eGHWT over the other bases, we try the same experiment on the pedestrian data on the same Toronto road network (Figure 5.3a). In Figure 5.3b, we can see that the eGHWT still has the fastest decaying curve. One observation is that the Walsh basis performs the worst due to the localized feature of the data.



(a) (b)

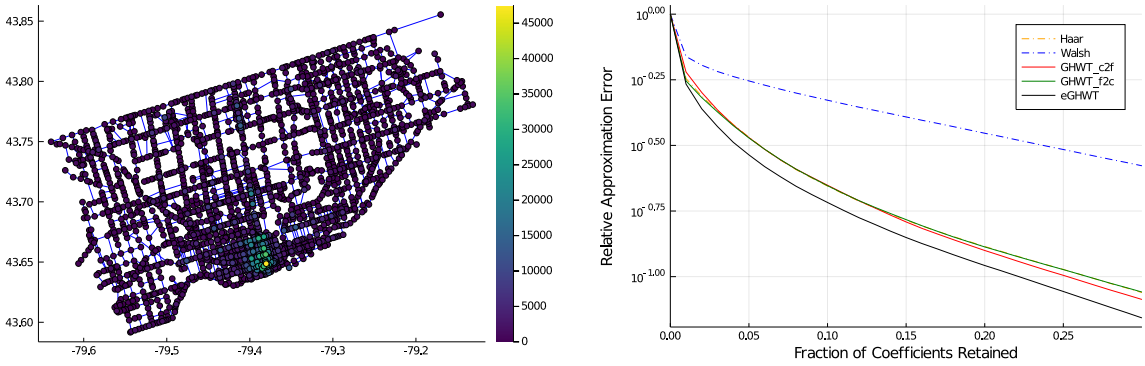
FIGURE 5.1. (a): Traffic volume data in the city of Toronto; (b): Relative ℓ^2 approximation error of the data shown in (a)



(a) Haar (b) c2f GHWT = Walsh

(c) f2c GHWT (d) eGHWT

FIGURE 5.2. Pointwise relative ℓ^2-error using 25% of the best-basis coefficients



(a) (b)

FIGURE 5.3. (a): Pedestrian volume data in the city of Toronto; (b): Relative ℓ^2 approximation error of the data shown in (a)

5.2. Viewing an Image (or a General Matrix Signal) as a Tensor Product of Graphs

To analyze and process a single signal supported on a graph, we can now use the eGHWT to produce a suitable ONB. Then for a collection of signals in a matrix form (including regular digital images), we can also compose the affinity matrix of the rows and that of the columns separately, thus define graphs that the rows and columns are supported on as was done previously [**10**, **32**]. Those affinity matrices can be either computed from the similarity of rows or columns directly or can be composed from information outside the original matrix signal. For example, Kalofolias et al. [**36**] used row and column graphs to analyze recommender systems.

After the affinity graphs on rows and columns are obtained, we can use the eGHWT to produce ONBs on rows and columns separately. Then the matrix signal can be analyzed or compressed by the tensor product of those two ONBs. In addition, as mentioned in Section 4.6, we have also extended the eGHWT to the tensor product of row and column affinity graphs and search for best 2D ONB on the matrix signal directly. Note that we can also specify or compute the binary partition trees in a non-adaptive manner (e.g., recursively splitting at the middle of each region), typically for signals supported on a regular lattice.

5.2.1. Approximation of the Barbara Image. In this section, we use the famous Barbara image. The size of the image is 512×512. The partition trees on the rows and columns are specified

explicitly: every bipartition is forced at the middle of each region. Therefore, those two trees are perfect binary trees with depth equal to $\log_2(512) + 1 = 10$.

Figure 5.4 displays the approximation performance of the Haar, c2f GHWT, f2c GHWT, and eGHWT bases. We can see that with the same fraction ($1/32 = 3.125\%$) of the coefficients retained, the eGHWT has much higher performance with less blocky artifacts than that of Haar and c2f/2c GHWTs. Figure 5.5 shows the zoomed-up face and left leg of those approximations. Especially for the leg region that has some specific texture, i.e., stripe patterns, the eGHWT outperformed the rest of the methods. The performance is measured by PSNR (peak signal-to-noise ratio). Given an $m \times n$ monochrome image I and its approximation K, the PSNR is defined as

$$PSNR = 10 \log_{10} \frac{(\max_{ij} I_{ij})^2}{MSE}$$
$$MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (I_{ij} - K_{ij})^2$$

5.2.2. The Haar Transform for Images with Non-Dyadic Size. For non-dyadic images, there is no straightforward way to obtain the partition trees in a non-adaptive manner as we did for the Barbara image in the previous section. This is a problem faced by the classical Haar transform as well, which requires an input image to be dyadic. Non-dyadic images are often modified by zero padding, even extension, or other methods before the Haar transform is applied. We propose to apply the Haar transform on a non-dyadic image without modifying the input image using the eGHWT.

To obtain the binary partition trees, we need to cut an input image $\boldsymbol{F}$ horizontally or vertically into two parts recursively. Apart from using the affinity matrices, we propose the *penalized total variation cost* to cut the input image. Denote the two sub-parts as $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$. We search for the optimal cut such that

$$\text{Penalized Total Variation Cost} := \frac{\|\boldsymbol{F}_1\|_{\text{TV}}}{|\boldsymbol{F}_1|^p} + \frac{\|\boldsymbol{F}_2\|_{\text{TV}}}{|\boldsymbol{F}_2|^p} \ (p > 0)$$

is minimized, where $\|\boldsymbol{F}_k\|_{\text{TV}} := \sum_{i,j}(|F_k[i+1,j] - F_k[i,j]| + |F_k[i,j+1] - F_k[i,j]|)$, and $|F_k|$ indicates the number of pixels in $\boldsymbol{F}_k$, $k = 1, 2$. The denominator is used to make sure that the size of $\boldsymbol{F}_1$ and that of $\boldsymbol{F}_2$ are close so that the tree becomes nearly balanced. Recursively applying the

(a) Haar, PSNR = 24.8dB

(b) c2f GHWT, PSNR = 23.9dB

(c) f2c GHWT, PSNR = 25.6dB

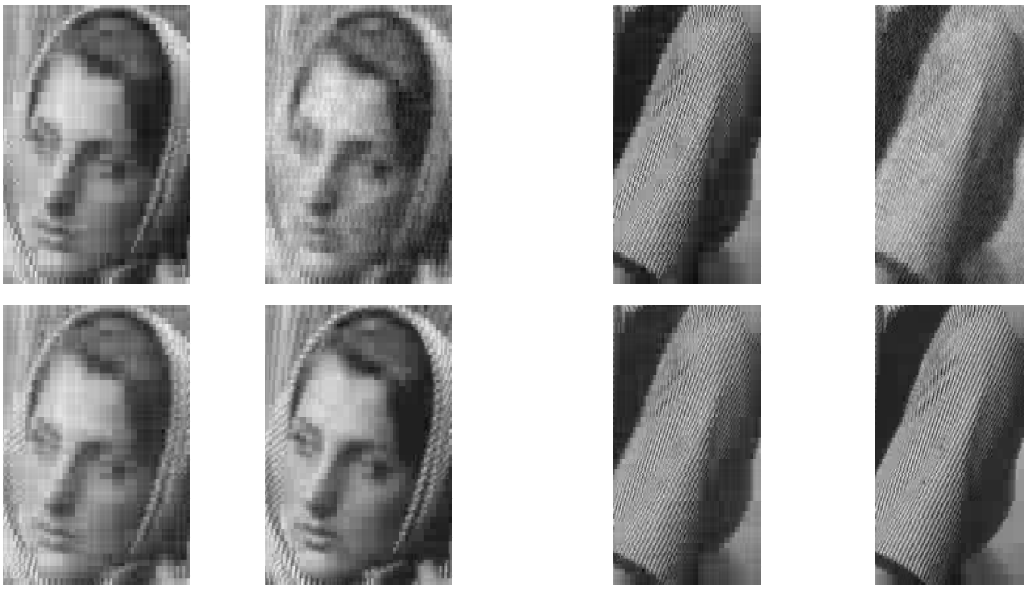
(d) eGHWT, PSNR = 28.1dB

FIGURE 5.4. Comparison of various bases: using only 3.125% of coefficients

horizontal cut on the rows of $\boldsymbol{F}$ and the vertical cut on the columns of $\boldsymbol{F}$ will give us two binary partition trees. We can then select the 2D Haar basis from the eGHWT dictionary or search for the best basis with minimal cost (note that this cost function for the best-basis search is different from the penalized total variation cost above).

Here we chose an image patch of size 100×100 around the face part from the original 512×512 Barbara image so that it is non-dyadic.

We chose $p = 3$ here.. To determine the value of p, we need to balance between the total variation and structure of the partition tree. Larger p means less total variation value after split but less balanced partition tree. The value of p can be fine-tuned based on the evaluation of the final task, for example, the area under the curve of the relative approximation error in the compression task [**33**].



(a) face (b) leg

FIGURE 5.5. Comparison of various bases: using only 3.125% of coefficients. Methods used from left to right and top to bottom: Haar; c2 GHWT; f2c GHWT; eGHWT.

Figure 5.6 shows that the decay speed of the eGHWT basis vector coefficients (regardless of the fixed Haar basis or the optimized best basis) is faster than the classical Haar transform (using non-adaptive cuts after even reflection to make the input image dyadic); however, we must say that the performance gain of the new Haar transform proposed here over the classical one is minimal; the eGHWT best basis performance is far better than both versions of the Haar transform.

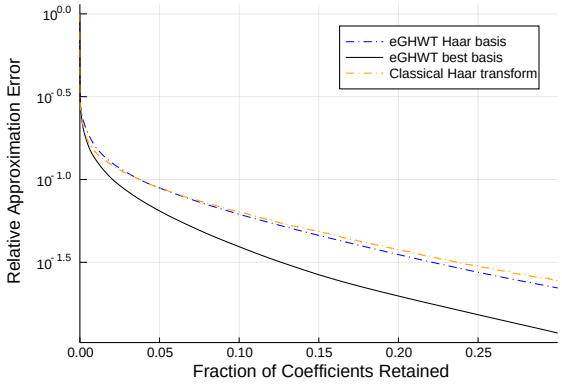


FIGURE 5.6. Comparison of the classical Haar transform and the eGHWT bases on the face part (of size 100×100) of the Barbara image

5.3. Constructing a Graph from an Image for Efficient Approximation

We can view a digital image of size $M \times N$ as a signal on a graph consisting of MN nodes by viewing each pixel as a node. Note that the underlying graph is not a regular 2D lattice of size $M \times N$. Rather it is a graph reflecting the relationship or affinity between pixels. In other words, w_{ij}, the weight of the edge between ith and jth pixels in that graph should reflect the affinity between local region around these two pixels, and this weight may not be 0 even if ith and jth pixels are remotely located. This idea have been used in image segmentation [**59**] as well as image denoising [**63**].

Here we define the edge weight w_{ij} as Szlam at al. [**63**] did:

$$w_{ij} = \mathrm{e}^{\frac{-\|F[i]-F[j]\|_2^2}{\sigma_F}} \cdot \begin{cases} \mathrm{e}^{\frac{-\|X[i]-X[j]\|_2^2}{\sigma_X}} & \text{if } \|X[i]-X[j]\|_2 < r \\ 0 & \text{otherwise} \end{cases}$$

where $X[i]$ is the spatial location of node (pixel) i, and $F[i]$ is a feature vector based on intensity, color, or texture information of the local region centered at that node. As one can see from the above weight, the pixels located within a disk with center $X[i]$ and radius r are considered to be the neighbors of the ith pixel. The scale parameters, σ_F and σ_X must be chosen appropriately. Once we construct this graph, we can apply the eGHWT in a straightforward manner.

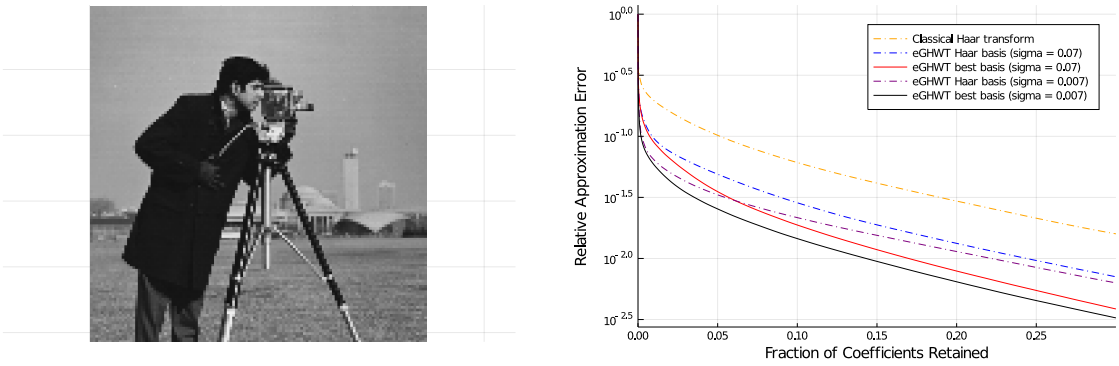
We examine two images here. The first one (Figure 5.7a) is the subsampled version of the camera man image of size 128×128 (we subsampled the original camera man image of size 512×512 to

reduce computational cost). We did two experiments to demonstrate the importance of the affinity matrix setup. As to the feature vector $F[i]$ at the ith pixel, we simply used its pixel (intensity) value here. For the other parameters, we used $r = 5, \sigma_X = \infty$ for both experiments, while $\sigma_F = 0.007$ for the first experiment and $\sigma_F = 0.07$ for the second experiment. The parameters can firstly be chosen according to the performance of the image cut reviewed by human, then fine-tuned based on the evaluation of the final task, for example, the area under the curve of the relative approximation error here [**33**].

Figure 5.7b shows our results on the subsampled camera man image. Figure 5.7b demonstrates that the decay rate of the expansion coefficients using the eGHWT dictionary is much faster than that of the classical Haar transform. Moreover, the basis vectors from the eGHWT best basis display some meaningful features of the image. Figure 5.8 shows the nine best-basis vectors corresponding to the largest expansion coefficients in magnitude. We can see that the human part or the camera part are captured by individual basis vectors. Even more, we can tell that Figure 5.8a has better segmentation result than Figure 5.8b, which coincides with the better decay rate from Figure 5.7b. The experiment with $\sigma_F = 0.007$ has better decay rate than the one with $\sigma_F = 0.07$. Therefore, we can conclude that good segmentation result (or good affinity matrix setup) is very important for the basis construction.

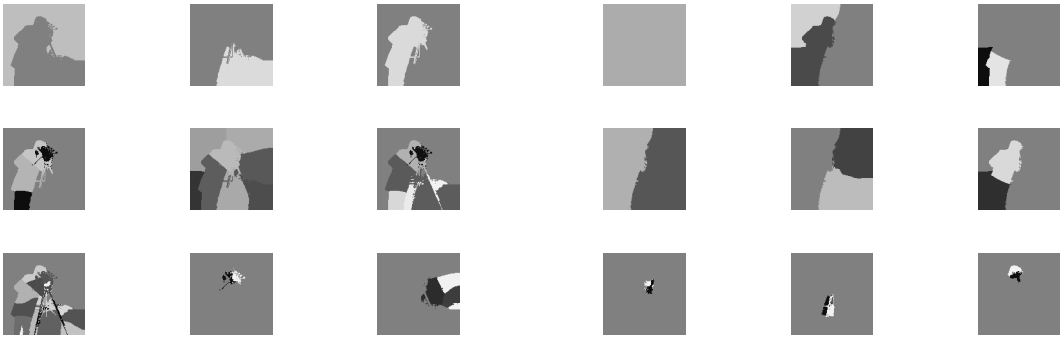
When we tuned the parameters, we found that simply choosing raw pixel value as $F[i]$ can get very good results for regular images. Among the parameters, σ_F is the most important one to get good segmentation results. One tuning trick is starting from the median of all possible values of the numerator in the exponential term, i.e., $-\|F[i]-F[j]\|_2^2$ [**63**]. Meanwhile, r is relatively flexible and $\sigma_X = \infty$ is always a good and simple choice. When we use $\sigma_X = \infty$, the weight between two pixels with distance larger than r is 0 and the affinity matrix becomes sparse.

The second example is a composite texture image. Our method, i.e., applying the eGHWT on an image viewed as a graph, allows us to generate basis vectors with irregular support that is adapted to the structure of the input image as shown in the camera man image example above, which works particularly well on this composite texture image. Figure 5.9a displays the original composite texture image. To compute the graph weights, we construct the mask image (Figure 5.9b) from the Gabor features [**22**] of the original texture image. Specifically, for a group of 2D Gabor



(a) (b)

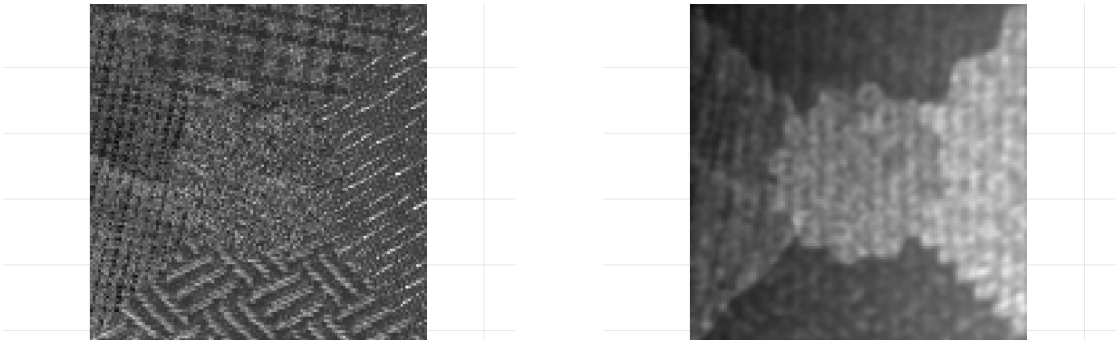
FIGURE 5.7. (a): The rescaled camera man image of size 128×128; (b): Relative ℓ^2 approximation error of (a) using five methods.



(a) (b)

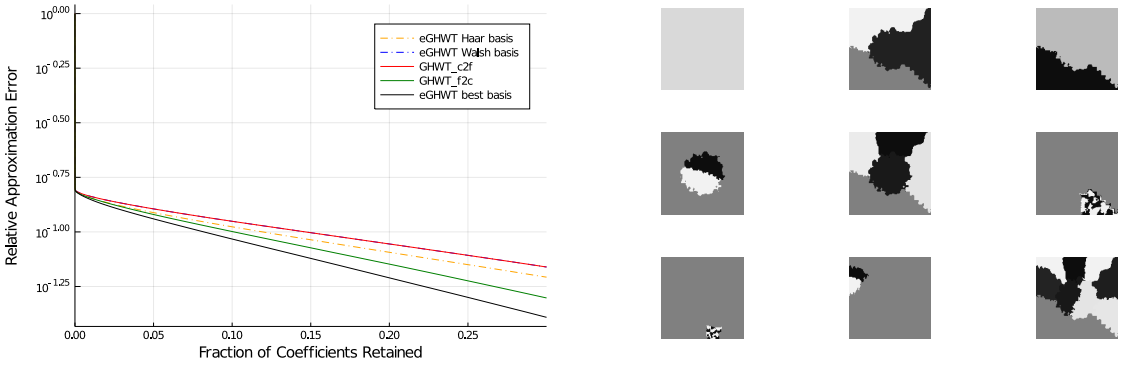
FIGURE 5.8. The top nine eGHWT best-basis vectors (a): $\sigma_F = 0.007$; (b): $\sigma_F = 0.07$.

filters with various wavelengths and orientations, we perform Gabor transform of the original image for each Gabor filter. Then the absolute values of the Gabor transform coefficients form a matrix with the same size as the original image for each Gabor filter. Then Principal Component Analysis (PCA) [**35**] is performed on all the coefficient matrices. The first component of PCA is used as the mask image. The graph weights are computed from the mask image in a similar manner as the camera man image. Figure 5.10a shows the approximation performance of five different methods. Figure 5.10b displays the top basis vectors of the eGHWT best basis. We can see that the support of the basis vectors coincide with the five sections of the composite texture image.



(a) (b)

FIGURE 5.9. (a): A composite textured image of size 128×128; (b): Mask image computed from PCA of Gabor transform coefficients



(a) (b)

FIGURE 5.10. (a): Relative ℓ^2 approximation error of Figure 5.9a using five methods; (b): Top 9 eGHWT best basis vectors

CHAPTER 6

Low rank approximation of matrices

In machine learning, low rank approximations to datasets are often employed to impute missing data, denoise noisy data, or perform feature extraction [**66**]. These techniques have applications in fields such as astronomy [**4**], computer vision [**42**], document clustering [**61**], and recommender systems [**19**].

To give a glimpse of the assumption, let us consider the movie rating matrix where the rows are users and columns are movies. We can reasonably assume that people can be clustered into k groups depending on their preference. If we further assume that the people in the same group have similar ratings on movies, then the rating matrix is close to k rank. More rigorous argument has been given by [**67**], by considering a generative model. Suppose that each row or column is associated to a (possibly high dimensional) bounded latent variable, then entries of that $m \times n$ matrix are within a fixed absolute error of a low rank matrix whose rank grows as $O(\log(m+n))$.

In this chapter, we discuss two methods based on the assumption that real data are low rank. One is the Nonnegative matrix factorization (NMF) algorithm and its initiation with eGHWT. The other is the matrix completion algorithm for recommender systems.

For simplicity of notation, we do not use bold font for vectors in this chapter.

6.1. NMF Initialization via eGHWT

6.1.1. Introduction. Nonnegative matrix factorization (NMF) is a useful decomposition for multivariate nonnegative data. Given data matrix $Y \in \mathbb{R}_{\geq 0}^{m \times n}$ with $Y_{ij} \geq 0$ and a pre-specified positive integer $k < \min(m, n)$, the goal is to find two nonnegative matrices $W \in R_{\geq 0}^{m \times k}$ and $H \in R_{\geq 0}^{k \times n}$ such that

$$Y \approx WH.$$

In practice, usually $k \ll \min(m, n)$. One conventional approach to find W and H is by solving the following optimization problem

$$\min_{W \in \mathbb{R}^{m \times k} H \in \mathbb{R}^{k \times n}} \|(Y - WH)\|_F^2 \quad \mathbf{s.t.} \quad W \geq 0, H \geq 0. \tag{6.1}$$

If we view the columns of W as k "basis vectors", then the j-th column of Y can be represented with k coefficients from the j-th column of H. To measure the dissimilarity between Y and WH, one can use other metrics instead of the Frobenius norm. For example, the Kullback-Leiber (KL) Divergence [**42**]

$$\min_{W \in \mathbb{R}^{m \times k} H \in \mathbb{R}^{k \times n}} D_{KL}(Y \,\|\, WH) \quad \mathbf{s.t.} \quad W \geq 0, H \geq 0,$$

or the Bregman Divergence [**61**]

$$\min_{W \in \mathbb{R}^{m \times k} H \in \mathbb{R}^{k \times n}} D_{Bregman}(Y \,\|\, WH) \quad \mathbf{s.t.} \quad W \geq 0, H \geq 0.$$

In this dissertation, we only consider the formulation (6.1).

One classical approach to solve (6.1) is the multiplicative update algorithm [**43**]. This fixed point algorithm updates W and H iteratively by

$$W_{ia} \leftarrow W_{ia} \frac{(YH^\intercal)_{ia}}{(WHH^\intercal)_{ia}},$$
$$H_{bj} \leftarrow H_{bj} \frac{(W^\intercal Y)_{bj}}{(W^\intercal WH)_{bj}}.$$

It is simple to implement and often yields good results. However, it is relatively slow and lacks convergence properties [**21**].

Another popular method is the *Alternating Nonnegative Least Squares Using Projected Gradient Methods* (ALSPGRAD) [**44**]. Let us break down its long name. "Alternating" means that it is an alternating method which fixes one matrix of W and H and improves the other in each iteration

$$W^{(t+1)} \leftarrow \arg\min_{W \geq 0} \|Y - WH^{(t)}\|_F^2,$$
$$H^{(t+1)} \leftarrow \arg\min_{H \geq 0} \|Y - W^{(t)}H\|_F^2,$$

where t is the iteration index. The sub-problems reduce to "least square" problems with "nonnegative" constraints. To solve the sub-progblems, "projected gradient method" is used. The details are summarized in Algorithm 7.

Algorithm 7: Alternating Nonnegative Least Squares Using Projected Gradient Methods (ALSPGRAD) [**44**]

Input: Input data matrix $Y \in \mathbb{R}^{m\times n}_{\geq 0}$, rank k
Output: $W \in \mathbb{R}^{m\times k}_{\geq 0}, H \in \mathbb{R}^{k\times n}_{\geq 0}$
Initialize $W^{(1)}, H^{(1)}$
for $t = 1, 2, ...$ **do**
  // $W^{(t+1)} \leftarrow \arg\min_{W\geq 0} \|Y - WH^{(t)}\|_F^2$
  $W_0^{(t)} \leftarrow W^{(t)}$
  for $p = 1, 2, ...$ **do** // Projected Gradient Descent
    $W_p^{(t)} \leftarrow [W_{p-1}^{(t)} - \alpha\nabla_W \|Y - W_{p-1}^{(t)} H^{(t)}\|_F^2)]_{\geq 0}$
    // break if stopping criterion is met
  end
  $W^{(t+1)} \leftarrow W_p^{(t)}$
  $H_0^{(t)} \leftarrow H^{(t)}$
  // $H^{(t+1)} \leftarrow \arg\min_{H\geq 0} \|Y - W^{(t+1)}H\|_F^2$
  for $q = 1, 2, ...$ **do** // Projected Gradient Descent
    $H_q^{(t)} \leftarrow [H_{q-1}^{(t)} - \alpha\nabla_H \|Y - W^{(t+1)} H_{q-1}^{(t)}\|_F^2)]_{\geq 0}$
    // break if stopping criterion is met
  end
  $H^{(t+1)} \leftarrow H_q^{(t)}$
  // break if stopping criterion is met
end

In Algorithm 7(and from now on), we define $(\cdot)_+ : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}_{\geq 0}$ by

$$[(X)_+]_{ij} = \begin{cases} X_{ij}, \text{ if } X_{ij} \geq 0 \\ 0, \text{ else} \end{cases}$$

The step size α can be selected through suitable line search algorithm. The time cost is

$$\#\text{iterations} \times (O(nmk) + \#\text{sub-iterations} \times O(smk^2 + snk^2))$$

where #iterations correspond to the index t in Algorithm 7; #sub-iterations correspond to the indices (p, q); and s is the average number of steps in the line search algorithm to find α. In practice $k \ll \min(m, n)$. Making use of this property, the algorithm converges quickly, although only the first order derivative is used. In the original paper [**44**], the author also tried projected gradient methods on (6.1) directly, meaning that W and H are updated simultaneously instead of alternatively, but it is not as fast as ALSPGRAD.

Another popular method is Hierarchical Alternating Least Squares (HALS) algorithm [**9**]. Multiple algorithms are introduced in the original paper [**9**], some of which can even be applied to tensor factorizations. In this dissertation, we only consider HALS on NMF. Similar to ALSPGRAD, HALS is an alternating algorithm. In ALGSPGRAD, one of W or H is fixed to improve the other. In HALS, only one column of W or one row of H is updated when the rest of W and H are fixed in each iteration. Let $W(:, j)$ denotes the j-th column of W and $H(j, :)$ denotes the j-th row of H. Define the residue

$$Y_j^{(t)} = Y - \sum_{p \neq j} W^{(t)}(:, p) H^{(t)}(p, :),$$

where t is the iteration index, then the columns of W and rows of H are updated by

$$W^{(t+1)}(:, j) \leftarrow \underset{w \in \mathbb{R}_{\geq 0}^{m \times 1}}{\arg\min} \|Y_j^{(t)} - w H^{(t)}(j, :)\|_F^2,$$

$$H^{(t+1)}(j, :) \leftarrow \underset{h \in \mathbb{R}_{\geq 0}^{1 \times n}}{\arg\min} \|Y_j^{(t)} - W^{(t)}(:, j) h\|_F^2.$$

Similarly, the sub-problems reduce to least square problems, which have closed form solutions. The details are summarized in Algorithm 8. Furthermore, the author improved HALS to develop FAST HALS. They can also be extended to NMF with additional constraints and tensor factorizations.

Algorithm 8: Hierarchical Alternating Least Squares (HALS) for NMF [**9**]

Input: Input data matrix $Y \in \mathbb{R}_{\geq 0}^{m \times n}$, rank k
Output: $W \in \mathbb{R}_{\geq 0}^{m \times k}, H \in \mathbb{R}_{\geq 0}^{k \times n}$
Initialize $W^{(1)}, H^{(1)}$
Normalize the columns(and rows) of W (and H) to unit $\ell^2$2-norm length
$E \leftarrow Y - W^{(1)}H^{(1)}$
for $t = 1, 2, ...$ **do**
  for $j = 1, 2, ..., k$ **do**
    $Y_j^{(t)} \leftarrow E + W^{(t)}(:, j)H^{(t)}(j, :)$
    $H^{(t+1)}(j, :) \leftarrow [(W^{(t)}(:, j))^{\mathsf{T}} Y_j^{(t)}]+$
    $H^{(t+1)}(j, :) \leftarrow H^{(t+1)}(j, :)/\|H^{(t+1)}(j, :)\|_2$
    $W^{(t+1)}(:, j) \leftarrow [Y_j^{(t)}(H^{(t+1)}(j, :))^{\mathsf{T}}]_+$
    $W^{(t+1)}(:, j) \leftarrow W^{(t+1)}(:, j)/\|W^{(t+1)}(:, j)\|_2$
    $E \leftarrow Y_j^{(t)} - W^{(t)}(:, j)H^{(t)}(j, :)$
  end
  `// break if stopping criterion is met`
end

6.1.2. Using the eGHWT for NMF initialization. Since the algorithms described in the previous section are all iterative methods, so as most of the available NMF algorithms, the initialization of the pair (W, H) is important in the design of successful implementation. Most of the NMF algorithms in the literature use random nonnegative initialization. Since they are likely to converge to a local minimum, it becomes necessary to run several instances of the algorithm using different random initializations and then select the best solution. Therefore there is a need to investigate good initialization strategies [**2**].

The Nonnegative Double Singular Value Decomposition (NNDSVD) method developed in [**5**] is a popular method for NMF initialization. It computes the rank k approximation through SVD of the input matrix Y. Then it extracts the positive section of those singular vectors as the initial estimates of W and H. The details are described in Algorithm 9. To illustrate the algorithm, suppose

$$Y \approx Y_k = \sum_{i=1}^{k} \sigma_i u_i v_i^{\mathsf{T}} = \sum_{i=1}^{k} y_i z_i,$$

where u_i, v_i are the left and right singular vectors, σ_i are the corresponding singular values and $y_i = \sqrt{\sigma_i} u_i$ and $z_i = \sqrt{\sigma_i} v_i^{\mathsf{T}}$. Using the operator $(\cdot)_+$ defined in previous section, the positive part

Algorithm 9: Nonnegative Double Singular Value Decomposition (NNDSVD) [**5**]

Input: Input data matrix $Y \in \mathbb{R}^{m\times n}_{\geq 0}$, rank k

Output: $W \in \mathbb{R}^{m\times k}_{\geq 0}, H \in \mathbb{R}^{k\times n}_{\geq 0}$

Compute the partial k SVD of $Y \approx \sum_{j=1}^{k} \sigma_j u_j v_j^{\mathsf{T}}$.

$W(:,1) \leftarrow \sqrt{\sigma_1} u_1;\ H(1,:) \leftarrow \sqrt{\sigma_1} v_1^{\mathsf{T}}$

for $j = 2, 3, ..., k$ **do**

  $x \leftarrow u_j$;

  $y \leftarrow v_j$;

  $x_+ \leftarrow (x)_+;\ x_- \leftarrow x - x_+$;

  $y_+ \leftarrow (y)_+;\ y_- \leftarrow y - y_+$;

  if $\|x_+\|\|y_+\| > \|x_-\|\|y_-\|$ **then**

    $u \leftarrow x_+/\|x_+\|;\ v \leftarrow y_+/\|y_+\|;\ \delta \leftarrow \|x_+\|\|y_+\|$

  else

    $u \leftarrow x_-/\|x_-\|;\ v \leftarrow y_-/\|y_-\|;\ \delta \leftarrow \|x_-\|\|y_-\|$

  end

  $W(:,j) \leftarrow \sqrt{\sigma_j \delta} u;\ H(j,:) \leftarrow \sqrt{\sigma_j \delta} v^{\mathsf{T}}$

end

of $y_{i+} = (y_i)_+$ and negative part of $y_{i-} = y_i - y_{i+}$. The same goes for z_i. Then we have

$$Y \approx \sum_{i=1}^{k} (y_{i+} - y_{i-})(z_{i+} - z_{i-}) = \sum_{i=1}^{k} y_{i+} z_{i+} + \sum_{i=1}^{k} y_{i-} z_{i-} - \sum_{i=1}^{k} y_{i+} z_{i-} - \sum_{i=1}^{k} y_{i-} z_{i+} \tag{6.2}$$

Algorithm 9 uses the first two summands of (6.2) to approximate W and H, but choose only one term from $y_{i+}z_{i+}$ or $y_{i-}z_{i-}$ for each i.

An improved SVD-based initialization is proposed in [**1**]. They use the first two summands of (6.2) to approximate W and H as well. However, instead of choosing only one term for each i in NNDSVD, they used both $y_{i+}z_{i+}$ and $y_{i-}z_{i-}$ for each i. Therefore approximately only $k/2$ SVD triplets of Y are required, then all the terms in the first two summands of (6.2) are used to approximate W and H. The algorithm is named as Nonnegative Singular Value Decomposition with Low-Rank Correction (NNSVD-LRC).

Here we propose to utilize the eGHWT for initialization of the NMF algorithms. The steps are as follows: 1) Construct the full row and column eGHWT dictionaries for a given nonnegative matrix Y; 2) Use the lasso with nonnegativity constraint [**27**] on the eGHWT scaling vectors to select the best k linear combination of rank 1 matrices $Y \approx \beta_1 \phi_{r,(1)} \phi_{c,(1)}^{\mathsf{T}} + \ldots + \beta_k \phi_{r,(k)} \phi_{c,(k)}^{\mathsf{T}}$, where $\phi_{r,(i)}, \phi_{c,(i)}, i = 1:k$ are the row and column eGHWT scaling vectors selected by lasso with

nonnegativity constraints; and 3) Initialize W and H by assigning the i-th column of W to be $\sqrt{\beta_i}\phi_{r,(i)}$ and the i-th row of H to be $\sqrt{\beta_i}\phi_{c,(i)}^{\mathsf{T}}$.

6.1.3. Numerical Results. We compare our method with NNDSVD [**5**] and NNSVD-LRC [**1**] on a group of synthetic data. The synthetic data is generated by $\tilde{V} = \tilde{W}\tilde{H}^\intercal + \sigma\tilde{\Sigma}$, where $\tilde{W}$, $\tilde{H}$ and $\tilde{\Sigma}$ are nonnegative matrices with random entries sampled uniformly from $[0, 1]$. $\sigma\tilde{\Sigma}$ serves as the noise matrix and σ controls the strength of the noise.

For each combination of m, n, k, σ, we generate $\tilde{V}$ 50 times. Each time, we initialize a given NMF algorithm with eGHWT, NNDSVD, and NNSVD-LRC separately. Then we run the main algorithm. The performance is measured by the number of iterations to converge on the main algorithm to the same tolerance. We sum up the number of trials with the eGHWT converging faster than NNDSVD, then divide it by the total number of trials, i.e., 50, to approximate the probability that the eGHWT converges faster than NNDSVD. The same metric is used to compare the eGHWT with NNSVD-LRC and with random initialization separately. Another reasonable metric would be the final residual norm $\|Y - WH\|_F$. However, we observe that these values are very close among different methods, even though they may correspond to different local minimum. Therefore, we use the iteration number as the metric for performance in these experiments.

Here we use the two main algorithms mentioned in Section 6.1.1. The results with ALSPGRAD [**44**] is listed in Table 6.1 and the results with HALS [**9**] are listed in Table 6.2. In those two tables, "eGHWT>NNDSVD" corresponds to the approximate probability that the eGHWT converges faster than NNDSVD. The same goes for the other rows. Bold numbers mean that they are larger than 0.5.

ALSPGRAD, HALS and NNDSVD are implemented in the Julia package `NMF.jl` [**45**]. NNSVD-LRC is implemented by ourselves. Note that in the original paper of NNSVD-LRC [**1**], after the initial matrices W_0 and H_0 are computed through the SVD approximation, they run an NMF algorithm on matrix W_0H_0 to get a new pair of W and H. Then that new pair is used as the initial matrices to feed into the main NMF algorithm. To make a fair comparison with other initialization methods, we only implemented the first half of NNSVD-LRC, i.e., the part using SVD to approximate W and H.

We can see that the eGHWT is better than NNDSVD and much better than random initiation in most of the cases, especially with the main algorithm HALS. The eGHWT is comparable with NNSVD-LRC.

(m, n, k, σ)	$(100, 100, 10, 0)$	$(125, 25, 5, 0)$	$(100, 100, 10, 0.5)$	$(125, 25, 5, 0.5)$
eGHWT > NNDSVD	**0.72**	**0.66**	**0.88**	**0.8**
eGHWT > Random	**0.64**	0.42	**0.70**	0.26
eGHWT > NNSVD-LRC	0.46	**0.50**	0.48	**0.54**

TABLE 6.1. ALSPGRAD

(m, n, k, σ)	$(100, 100, 10, 0)$	$(125, 25, 5, 0)$	$(100, 100, 10, 0.5)$	$(125, 25, 5, 0.5)$
eGHWT > NNDSVD	**0.58**	**0.86**	**0.60**	**0.74**
eGHWT > Random	**0.78**	**0.86**	**0.78**	**0.86**
eGHWT > NNSVD-LRC	**0.56**	0.38	**0.50**	0.48

TABLE 6.2. HALS

6.1.4. Discussion. The row-(or column-) support of a single scaling vector is the support of the corresponding subgraph in the partition tree, which indicates a sub-cluster of the rows (or columns). Using a single scaling vector to approximate Y is equivalent to putting the same value (which is equal to the Lasso coefficient) on a sub-cluster of rows and columns. Using multiple scaling vectors through Lasso is equivalent to putting suitable values (which is equal to the Lasso coefficients) on different sub-clusters followed by summing them up. Our method makes use of clutering techniques to initialize W and H, in a subtle way.

In practice, we can use those scaling vectors only from the top levels of the eGHWT, instead of computing the whole eGHWT dictionary to save computational time. Here we use the bi-clustering method proposed in [**14**] to form the binary partition trees on columns and rows. One thing needs to be considered in Lasso [**27**] is the choice of λ, the penalty paramter on the ℓ^1 norm of the coefficients. To reduce computational cost, we roughly tune λ by selecting the one from $(2^{-15}, 2^{-14}, \ldots, 2^{-6}, 2^{-5})$ so that the corresponding number of non-zero coefficients is closest to k. The final results of NMF can be further improved through fine tuning λ or improving the bipartition results.

We observe that NNSVD-LRC works better than NNDSVD, which concides with the results from [**1**]. One major reason is that the smaller the index i is , the more information is contained in $y_i z_i$. NNDSVD discards more information in $y_i z_i$ for smaller i than NNSVD-LRC. More theoretical arguments have not been given in the original paper. Another observation is that the eGHWT is better than the random initialization when the matrix size is 100×100 but worse when the

size is 125×25 with ALSPGRAD as the main algorithm. To investigate this phenomena, we tried other settings of m and n. In most of the cases, the comparison results of the eGHWT, NNDSVD and NNSVD-LRC are consistent with the table above. But the performance of the random initialization varies with different m and n. The theoretical results of NMF initialization can be a potential research project in the future.

6.2. Bounded Matrix Completion

In this section[1], following the same assumption that some real datasets can be approximated by low rank matrices, we introduce our Bounded Matrix Completion (BMC) [**17**] algorithm for the recommender system. The formulation of matrix completion problem is similar to NMF but the data are partially observed. In the summary, we will discuss how BMC can be combined with the eGHWT in the future.

6.2.1. Introduction. Matrix factorization and matrix completion [**39**] are widely used for recommender systems. Suppose we have a user-item matrix $Y \in \mathbb{R}^{\geq 0 m \times n}$, which is partially observed on the locations $\{(i,j)\} =: \Omega$, the goal is to predict missing ratings. Define the operator $P_\Omega : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ by

$$[P_\Omega(Y)]_{ij} = \begin{cases} Y_{ij}, \text{ if } (i,j) \in \Omega \\ 0, \text{ if } (i,j) \notin \Omega \end{cases}.$$

To rephrase the problem, suppose we are given $P_\Omega(Y)$, the goal is to find suitable X to approximate Y. Two methods, which assumes that X is low rank, among many are to solve the following optimization problems:

$$\min_{W\in\mathbb{R}^{m\times k} H\in\mathbb{R}^{k\times n}} \frac{1}{2}\|\mathcal{P}_\Omega(Y - WH)\|_F^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2) \tag{6.3}$$

$$\min_{X\in\mathbb{R}^{m\times n}} \frac{1}{2}\|\mathcal{P}_\Omega(Y - X)\|_F^2 + \lambda\|X\|_*. \tag{6.4}$$

Equation (6.4) is convex, and equation (6.3) is nonconvex. They are equivalent if k is chosen large enough. In practice, usually $|\Omega| \ll mn$ and $k \ll \min(m,n)$. In (6.3), $W \in \mathbb{R}^{m\times k}$ is a thin matrix and $H \in \mathbb{R}^{k\times n}$ is a flat matrix. Similarly to NMF, WH is the approximation of Y. In (6.4), X is the approximation of Y. $\|X\|_*$ is the nuclear norm of X, which is defined as the sum of singular values, and it is a convex envelope of rank function [**54**]. Specifically, $\|X\|_*$ is the largest convex function such that it is smaller than or equal to $\text{rank}(X)$ for all X.

[1]This section was published in Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, Huang Fang, Zhang Zhen, Yiqun Shao, Cho-Jui Hsieh, *Improved Bounded Matrix Completion for Large-Scale Recommender Systems*, Pages 1654-1660, Copyright IJCAI Organization (2017).

In many real world problems, the ratings are bounded within a certain region. For example, the ratings of Netflix and MovieLens [**26**] datasets are bounded in the range of $[0.5, 5]$, so it is reasonable to impose bounded constraints to (6.3) and (6.4), leading to the following Bounded Matrix Completion (BMC) problem:

$$\min_{W \in \mathbb{R}^{m \times k} H \in \mathbb{R}^{k \times n}} \frac{1}{2} \|\mathcal{P}_\Omega(Y - WH)\|_F^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2) \quad \mathbf{s.t.} \quad r_{\min} \leq WH \leq r_{\max} \tag{6.5}$$

$$\min_{X \in \mathbb{R}^{m \times n}} \frac{1}{2} \|\mathcal{P}_\Omega(Y - X)\|_F^2 + \lambda \|X\|_* \quad \mathbf{s.t.} \quad r_{\min} \leq X \leq r_{\max} \tag{6.6}$$

R. Kannan and Park [**37**] proposed an efficient block coordinate descent algorithm (named the BMA algorithm) for solving the non-convex form (6.5) of the BMC problem. On real datasets with bounded ratings, it outperforms traditional matrix completion algorithms based on (6.3) and (6.4). However, the BMA algorithm does not always converge to stationary points (where the gradients are zero) and can easily stuck in non-stationary points. This unstable convergence behavior leads to a performance drop in practice.

For example, consider the problem with $Y = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $[r_{\min}, r_{\max}] = [0, 1]$, Ω covering all the indexes. Starting from the initial estimates $W = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $H = \begin{pmatrix} 0 & 0 \end{pmatrix}$, the algorithm will stay there and W, H cannot change. But this solution is not optimal or even stationary.

We propose a new algorithm to solve the convex form (6.6). Our algorithm is based on the ADMM (Alternating Direction Method of Multipliers) framework [**6**] and it is guaranteed to converge to the optimal solution. Moreover, with carefully-designed update rules, our algorithm can scale to large datasets without encountering $O(mn)$ space complexity. Experimental results on real world datasets show that our algorithm can reach a better solution and is also much faster than BMA.

6.2.2. The BMC Algorithm. To deploy the ADMM, we split the loss, regularization, and constraints in (6.6), leading to the following equivalent form:

$$\begin{aligned} &\min_{X,Z,D} \frac{1}{2}\|\mathcal{P}_\Omega(Y-X)\|_F^2 + \lambda\|Z\|_* \\ &\textbf{s.t. } X = Z, \quad Z = D, \quad r_{\min} \le D \le r_{\max}. \end{aligned} \tag{6.7}$$

However, X is a dense and full rank matrix during the optimization procedure before convergence, whose space cost is $O(mn)$. To reduce the space cost, we further decompose X into $P_\Omega(X)+P_{\bar{\Omega}}(X)$, where $\bar{\Omega}$ is the complement set of Ω. $P_\Omega(X)$ is a sparse matrix and $P_{\bar{\Omega}}(X)$ is a low rank matrix, which is stored as the product of two low rank matrices. For simplicity, we change the notation by $P_\Omega(X) \to X$ and $P_{\bar{\Omega}}(X) \to E$, then we have

$$\begin{aligned} \min_{X,Z,D} \quad & \frac{1}{2}\|\mathcal{P}_\Omega(Y-X)\|_F^2 + \lambda\|Z\|_* \\ \text{s.t.} \quad & X + E = Z, \quad \mathcal{P}_{\bar{\Omega}}(X) = 0, \quad \mathcal{P}_\Omega(E) = 0 \\ & Z = D, \quad r_{\min} \le D \le r_{\max}. \end{aligned} \tag{6.8}$$

The scaled form of the *augumented partial Lagrangian* [**15**] is

$$\begin{aligned} L(X,Z,E,D,U_1,U_2) \quad &= \frac{1}{2}\|\mathcal{P}_\Omega(Y-X)\|_F^2 + \lambda\|Z\|_* \\ \mathcal{P}_{\bar{\Omega}}(X)=0, \mathcal{P}_\Omega(E)=0, r_{\min}\le D\le r_{\max} \quad &+\frac{\rho_1}{2}\|X-Z+E+U_1\|_F^2 + \frac{\rho_2}{2}\|Z-D+U_2\|_F^2 + \text{const}, \end{aligned} \tag{6.9}$$

where $U_1, U_2 \in \mathbb{R}^{m\times n}$ are scaled Lagrangian multipliers and ρ_1, ρ_2 are penalty parameters. Since (6.8) is closed, proper and convex, it can be solved by finding a saddle point of (6.9). Under the

Gauss-Seidel framework, ADMM finds the saddle point using the following iterative procedure:

$$X^{(t+1)} = \underset{\mathcal{P}_{\bar{\Omega}}(X)=0}{\arg\min} \frac{1}{2}\|\mathcal{P}_{\Omega}(Y-X)\|_F^2 + \frac{\rho_1}{2}\|X - Z^{(t)} + E^{(t)} + U_1^{(t)}\|_F^2 \tag{6.10}$$

$$Z^{(t+1)} = \underset{Z}{\arg\min}\ \lambda\|Z\|_* + \frac{\rho_1}{2}\|X^{(t+1)} - Z + E^{(t)} + U_1^{(t)}\|_F^2 + \frac{\rho_2}{2}\|Z - D^{(t)} + U_2^{(t)}\|_F^2 \tag{6.11}$$

$$E^{(t+1)} = \underset{\mathcal{P}_{\Omega}(E)=0}{\arg\min} \frac{\rho_1}{2}\|X^{(t+1)} - Z^{(t+1)} + E + U_1^{(t)}\|_F^2 \tag{6.12}$$

$$D^{(t+1)} = \underset{r_{\min}\leq D\leq r_{\max}}{\arg\min} \|Z^{(t+1)} - D + U_2^{(t)}\|_F^2 \tag{6.13}$$

$$U_1^{(t+1)} = U_1^{(t)} + X^{(t+1)} - Z^{(t+1)} + E^{(t+1)} \tag{6.14}$$

$$U_2^{(t+1)} = U_2^{(t)} + Z^{(t+1)} - D^{(t+1)}. \tag{6.15}$$

The sub-problems are solved as follows:

- In (6.10), the entries of X are independent terms of a sum, which can be solved in closed form individually: $X_{i,j}^{(t+1)} = \frac{1}{1+\rho_1}(Y_{i,j} + \rho_1 Z_{i,j}^{(t)} - \rho_1 E_{i,j}^{(t)} - \rho_1 U_{1i,j}^{(t)})$ for $(i,j) \in \Omega$ and $X_{i,j}^{(t+1)} = 0$ otherwise.
- (6.11) is equivalent to

$$\underset{Z}{\arg\min}\ \lambda\|Z\|_* + \frac{\rho}{2}\|Z - A\|_F^2, \tag{6.16}$$

where $A = \frac{\rho_1}{\rho_1+\rho_2}(E^{(t)} + X^{(t+1)} + U_1^{(t)}) + \frac{\rho_2}{\rho_1+\rho_2}(D^{(t)} - U_2^{(t)})$ and $\rho = \rho_1 + \rho_2$. Given the SVD decomposition of $A = U\Lambda V$, (6.16) has a closed form solution by soft-thresholding singular values:

$$Z^{(t+1)} = U\text{soft}_{\lambda/\rho}(\Lambda)V^{(t)},$$

$$\text{soft}_{\lambda/\rho}(\Lambda) := \mathbf{diag}[(\lambda_1 - \lambda/\rho)_{\geq 0}, (\lambda_2 - \lambda/\rho)_{\geq 0}, ..., (\lambda_n - \lambda/\rho)_{\geq 0}].$$

Here only top singular values greater than the threshold λ/ρ need to be computed, instead of the full SVD. This can be solved iteratively by power iterations [**24**] or PROPACK [**40**]. PROPACK requires specifying the number of singular values to compute the partial SVD. We can either give a fixed number, or dynamically predict the number of singular values greater than the threshold in each iteration as in [**28**, **50**]. For simplicity, we use fixed rank k with PROPACK during implementation.

- (6.12) has a simple closed form solution: $E_{i,j}^{(t+1)} = Z_{i,j}^{(t+1)} - X_{i,j}^{(t+1)} - U_{1i,j}^{(t)}$ for $(i,j) \notin \Omega$ and $E_{i,j}^{(t+1)} = 0$ elsewhere.
- For eq (6.13), $D^{(t+1)} = \Pi_{[r_{\min}, r_{\max}]}(Z^{(t+1)} + U_2^{(t)})$, where $\Pi_{[r_{\min}, r_{\max}]}$ is the projection of entries into $[r_{\min}, r_{\max}]$.

To summarize, we have Algorithm 10.

Algorithm 10: BMC-ADMM for Bounded Matrix Completion

Input: Observed matrix $P_\Omega(Y)$. Penalty parameters $\lambda, \rho_1, \rho_2 > 0$. Rank k. Upper and lower bound $r_{\min}, r_{\max}$. Maximum iteration number M.
Output: Rating matrix D
Initialize X, D, Z with *baseline initialization* [**38**]
Initialize U_1, U_2 with zeros matrices $\in \mathbb{R}^{m \times n}$
for $t = 1, 2, ..., M$ **do**
  `// Solve` (6.10)
  $\mathcal{P}_\Omega(X^{(t+1)}) \leftarrow \frac{1}{1+\rho_1}\mathcal{P}_\Omega(Y + \rho_1(Z^{(t)} - E^{(t)} - U_1^{(t)}))$
  $\mathcal{P}_{\bar{\Omega}}(X^{(t+1)}) \leftarrow 0$
  `// Solve` (6.11)
  $\rho \leftarrow \rho_1 + \rho_2$
  $A \leftarrow \frac{\rho_1}{\rho_1+\rho_2}(E^{(t)} + X^{(t+1)} + U_1^{(t)}) + \frac{\rho_2}{\rho_1+\rho_2}(D^{(t)} - U_2^{(t)})$
  $(U, S, V) \leftarrow Partial\_SVD(A, k)$
  $D \leftarrow diag(diag(S(1:k, 1:k) - \lambda/\rho)_{\geq 0})$
  $Z^{(t+1)} \leftarrow UDV^\top$
  `// Solve` (6.12)
  $\mathcal{P}_{\bar{\Omega}}(E^{(t+1)}) \leftarrow \mathcal{P}_{\bar{\Omega}}(X^{(t+1)} - Z^{(t+1)} + U_1^{(t)})$
  $\mathcal{P}_\Omega(E^{(t+1)}) \leftarrow 0$
  `// Solve` (6.13)
  $D^{(t+1)} \leftarrow Z^{(t+1)} + U_2^{(t)}$
  $D^{(t+1)}(D^{(t+1)} > r_{\max}) \leftarrow r_{\max}$
  $D^{(t+1)}(D^{(t+1)} < r_{\min}) \leftarrow r_{\min}$
  `// Update Scaled Lagrangian multipliers through` (6.14) `and` (6.15)
  $U_1^{(t+1)} \leftarrow U_1^{(t)} + X^{(t+1)} - Z^{(t+1)} + E^{(t+1)}$
  $U_2^{(t+1)} \leftarrow U_2^{(t)} + Z^{(t+1)} - D^{(t+1)}$
  if *stopping criterion is met* **then**
    break
  end
end

- (Time cost) The most time consuming steps are (6.11) and (6.13). For (6.11), we use PROPACK [**40**] to get truncated SVD, which is implemented through an iterative method. For (6.13), the time cost is $O(mn)$ per iteration to impose the bounded constraints $[r_{\min}, r_{\max}]$ on all entries of $D^{(t+1)}$.
- (Space cost) Here the same trick in [**46**] is used to save the space cost. Y, X, U_1 are sparse matrices($O(|\Omega|)$). U_2 is a sparse matrix with number of nonzero items equal to number of D's entries on the boundary, which means that their values are $r_{\max}$ or $r_{\min}$. Z is stored as the product of two low rank matrices in $\mathbb{R}^{m\times k}$, $\mathbb{R}^{k\times n}$ respectively, thus its space cost is $O(k(m+n))$. E can be expressed by $E = \mathcal{P}_{\bar{\Omega}}(Z)$. When updating D with (6.13), the entries violating the constraints need to be stored. Fortunately, the numerical experiments show that those entries are far less than $|\Omega|$. In summary, the total space cost is $O(k(m+n)+|\Omega|)$

dataset	m	n	$\|\Omega\|$	range
movielens100k	671	9,066	100,004	[0.5, 5]
movielens10m	71,567	10,677	10,000,054	[0.5, 5]
Flixster_subset	14,761	4,903	81,495	[0.5, 5]
Flixster	147,612	48,794	8,196,077	[0.5, 5]
Jester	50,692	150	1,728,847	[-10, 10]

TABLE 6.3. Dataset Statistics

6.2.3. Numerical Experiments. We use five real datasets and the details are listed in Table 6.3. The MovieLens data and Flixster data are different groups of ratings of movies by users. The Jester data is the ratings of jokes by users of the Jester Joke Recommender System. `movielens100k` has 100k ratings and `movielens10m` has 10m ratings. `Flixter_subset` is a subset of `Flixter` randomly selected by ourselves. We compare our our BMC-ADMM algorithm (Algorithm 10) with two existing matrix completion algorithms:

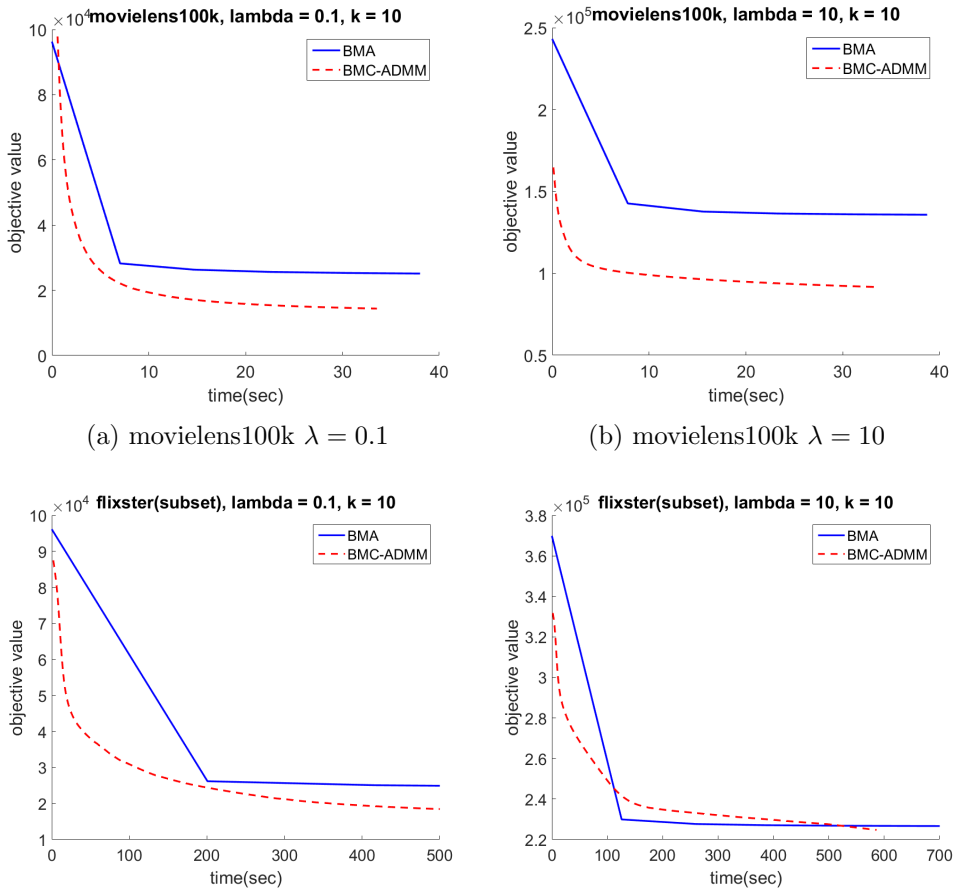
- BMA [**37**]: The coordinate descent algorithm solving (6.5)
- CCD++ [**69**]: The traditional matrix completion algorithm solving (6.3)

For each experiment, we randomly split 80% of the dataset as training data and 20% as testing data. The regularization parameter λ is chosen from $\{0, 0.01, 0.1, 1, 10, 100\}$ through cross-validation if not specified. Note that BMA is slow and requires large memory when data is large, so we only use part of the large dataset when BMA is included in comparison.

In the first set of experiments, we compare BMC-ADMM with BMA [**37**] on `movielens100k` and a subset of `Flixter`. In Figure 6.1, we set $\rho_1 = \rho_2 = 1$ and $k = 10$, and try $\lambda = 0.1$ and $\lambda = 10$ separately. We observe that BMC-ADMM can achieve lower objective function value than BMA. The main reason is that BMA can stuck at non-stationary points while our algorithm always converges to stationary points. In Figure 6.2, we also compare them with different rank k. Results show that BMC-ADMM can always find a better solution than BMA.

We test the scalability of BMC-ADMM by running it on larger datasets, `movielens10m`, `Flixster` and `Jester`. We observe that a larger rank k usually leads to a lower RMSE but slower convergence (Figure 6.3 b,c), and can also leads to over-fitting (Figure 6.3 a).

To show the superiority of BMC-ADMM over the other methods, we list the test RMSE for the three algorithms on all the five datasets in Table 6.4. To have a fair comparison, we tried



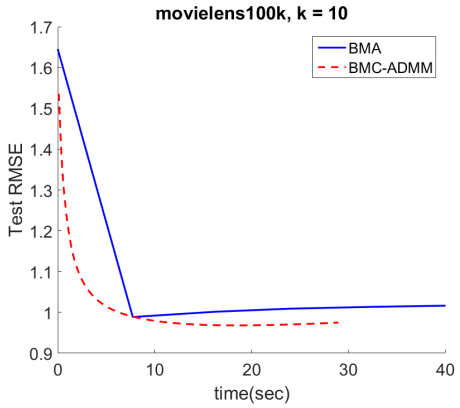
(a) movielens100k $\lambda = 0.1$ (b) movielens100k $\lambda = 10$

(c) Flixster(subset) $\lambda = 0.1$ (d) Flixster(subset) $\lambda = 10$

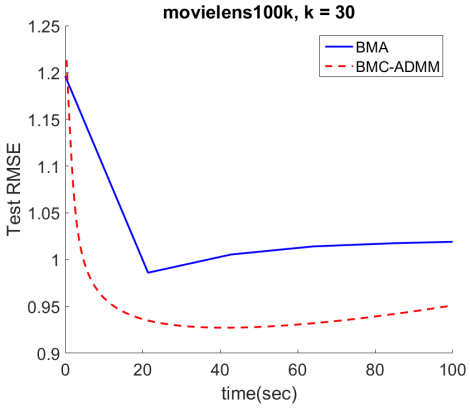
FIGURE 6.1. Convergence behaviour for different values of λ on movielens100k & Flixster(subset)

different settings of rank $k = 5, 10, 30$. The other parameters of each algorithm are chosen by cross-validation individually. As mentioned earlier, BMA is not scalable to large dataset, i.e., the full `Flixter` dataset. For the other experiments, BMC-ADMM can often achieve a lower RMSE than BMA. The last column of Table 6.4 corresponds to the standard matrix completion algorithm [**69**] without the bounded constraints. The results show that BMC-ADMM outperforms it in most cases. It indicates that adding bounded constraints is really useful in practice to achieve better prediction accuracy.

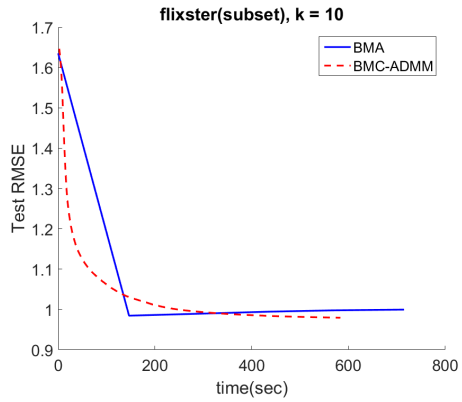
Other experiemnts show that BMC-ADMM is slower than the standard matrix completion algorithm [**69**]. To further study the time cost of our algorithm, we record the run time of each step during implementation on large datasets. The results are displayed in Table 6.5. It coincides with



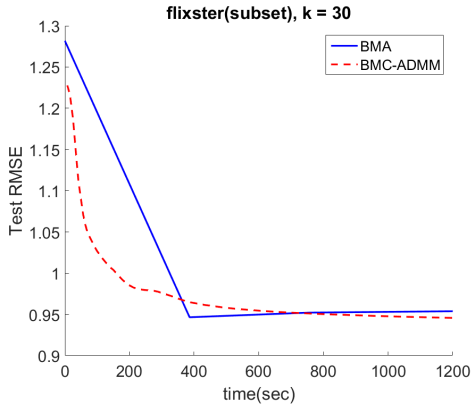
(a) movielens100k, $k = 10$



(b) movielens100k, $k = 30$

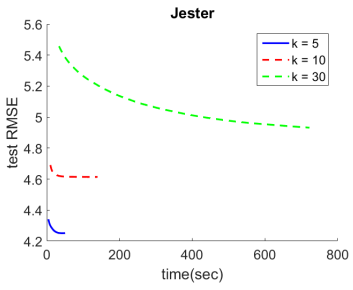


(c) Flixster(subset), $k = 10$

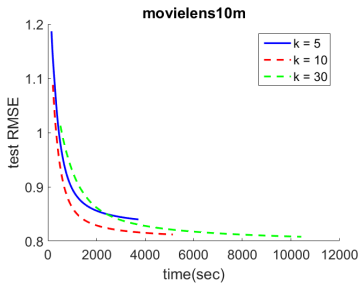


(d) Flixster(subset), $k = 30$

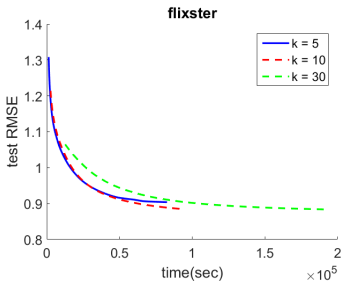
FIGURE 6.2. Test RMSE for different values of k (rank) on movielens100k and Flixster(subset)



(a) Jester



(b) Movielens10m



(c) Flixster

FIGURE 6.3. Test RMSE of BMC-ADMM for larger datasets

dataset	k	Global Mean	BMC-ADMM	BMA	Standard MF (CCD++)
movielens100k	5	1.0617	**1.0073**	1.2545	1.104
movielens100k	10	1.0617	**0.9689**	0.9858	1.096
movielens100k	30	1.0617	**0.9177**	0.9970	1.087
movielens10m	5	1.06	0.8399	0.8887	**0.8205**
movielens10m	10	1.06	0.8122	0.87819	**0.8110**
movielens10m	30	1.06	**0.8080**	0.88488	0.8098
Flixster_subset	5	1.0555	1.0458	**0.9700**	1.2177
Flixster_subset	10	1.0555	1.0014	**0.9664**	1.2205
Flixster_subset	30	1.0555	**0.9287**	0.9592	1.2168
Flixster	5	1.0921	**0.9198**	- -	0.9247
Flixster	10	1.0921	**0.8854**	- -	0.9187
Flixster	30	1.0921	**0.8838**	- -	0.9165
Jester	5	5.2747	**4.2452**	4.4587	4.3268
Jester	10	5.2747	4.6222	4.5772	**4.4069**
Jester	30	5.2747	4.9631	4.501	**4.4262**

TABLE 6.4. Test RMSE Comparison of BMC-ADMM, BMA and CCD++ on five datasets with $k = 5, 10, 30$ separately. The column "Global Mean" is using the mean of all training set to predict all the testing set.

dataset	k	Eq. (6.10)	Eq. (6.11)	Eq. (6.13)
movielens10m	5	0.67	1.47	**11.47**
movielens10m	10	1.04	2.01	**14.6**
Flixster	5	0.69	1.52	**109.42**
Flixster	10	1.20	2.18	**151.84**

TABLE 6.5. Average runtime(sec) of each step in each iteration

our analysis that the step updating D, more specifically thresholding D by constraints, dominates the run time, which is the bottleneck of our algorithm. To improve this, one possible way is to use parallel computing for this step.

6.2.4. Summary. In this section, we considered the bounded matrix completion problem. We pointed out the convergence problem of the existing algorithm BMA and proposed a novel algorithm based on the ADMM framework. Experimental results showed that our approach is faster, more accurate and scalable than BMA.

One possible research in the future is to impose constraints on BMC with the eGHWT vectors coefficients. We can use other information rather than the recommender system itself, for example, how users are related can be specified as a graph with information connected through social network. The graph on movies can be formed in a similar way. Then we can construct the eGHWT

dictionary on the recommender system. The coefficients of the matrix on certain basis vectors of the eGHWT can be used to measure smoothness on those two graphs. We can put constraints on those coefficients as part of the formulation of BMC (or general recommender system algorithms) to improve the final results. Using graphs to analyze matrix has been conducted by Kalofolias et al. [**36**].

CHAPTER 7

Summary

In this dissertation, we have introduced the *extended Generalized Haar-Walsh Transform* (eGHWT). After briefly reviewing the previous Generalized Haar-Walsh transform (GHWT), we have described how the GHWT can be improved with the new best-basis algorithm, which is the generalization of the Thiele-Villemoes algorithm [**65**] for the graph setting. We call this whole procedure of developing the extended Haar-Walsh wavelet packet dictionary on a graph and selecting the best basis from it as the *eGHWT*. Moreover, we have developed the 2D eGHWT for matrix signals by viewing them as tensor products of two graphs, which is a generalization of the Lindberg-Villemoes algorithm [**47**] for the graph setting.

We then showcased the applications of the eGHWT. When analyzing graph signals, we demonstrated the improvement over the GHWT on synthetic and real data. For the synthetic 6-node signal, we showed that the best basis from the eGHWT cannot be selected by the c2f-GHWT or the f2c-GHWT dictionaries and it had the minimal cost value compared to the c2f-GHWT best basis and f2c-GHWT best basis. On the Toronto traffic data, the eGHWT had the best approximation performance among other methods. Then we proceeded to the applications to image analysis. We constructed the unbalanced Haar-Walsh wavelet packet for non-dyadic images through the eGHWT, which showed its superiority over the classical one. In addition, by computing one single graph from image dataset, we could construct basis vectors on image with adaptive support. Last but not least, we demonstrated how the eGHWT can be used in the initialization of the NMF algorithms. With comparison with random initialization, NNDSVD and NNSVD-LRC, our method showed the best performance. We also introduced the BMC algorithm.

The eGHWT is constructed upon the binary partition tree (or tensor products of partition trees in the case of 2D signals). Currently, we use the Fiedler vectors of Laplacian matrices to form the partition tree. However, as we have mentioned earlier, our method is so flexible that any graph cut method or general clutering method can be used, as long as the binary tree is formed.

Another major contribution of our work is the software package we developed. Based on the MTSG toolbox written in MATLAB by Jeff Irion, we developed the `MTSG.jl` package in Julia [**3**], which includes the new eGHWT implementation for 1D and 2D signals. We hope that interested readers will download the software themselves, and conduct their own experiments with it (`https://gitlab.com/BoundaryValueProblems/MTSG.jl`).

There is still much research can be done related to this work. The most important one is using the eGHWT to analyze term-document matrices, or other matrix signals. More precisely, we look for meaningful structures through the eGHWT basis vectors in matrix signals. We have done some related experiments. However, the results highly depend on how the binary partition trees are computed. Finding suitable clutering techniques to fulfill this task remains to be investigated. Another one is to develop theoretical results on the NMF initialization with the eGHWT. In addition, as we mentioned in the BMC section, the eGHWT can be combined with BMC, or other matrix factorization or matrix completion algorithms, by imposing the constraints on the coefficients of the eGHWT.

Bibliography

[1] S. M. Atif, S. Qazi, and N. Gillis, *Improved SVD-based initialization for nonnegative matrix factorization using low-rank correction*, Pattern Recognition Letters, 122 (2019), pp. 53–59.

[2] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons, *Algorithms and applications for approximate nonnegative matrix factorization*, Computational Statistics & Data Analysis, 52 (2007), pp. 155–173.

[3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A fresh approach to numerical computing*, 59 (2017), pp. 65–98.

[4] M. R. Blanton and S. Roweis, *K-corrections and filter transformations in the ultraviolet, optical, and near-infrared*, The Astronomical Journal, 133 (2007), p. 734.

[5] C. Boutsidis and E. Gallopoulos, *SVD based initialization: A head start for nonnegative matrix factorization*, Pattern Recognition, 41 (2008), pp. 1350–1362.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Foundations and Trends in Machine Learning, (2011), pp. 1–122.

[7] J. C. Bremer, R. R. Coifman, M. Maggioni, and A. D. Szlam, *Diffusion wavelet packets*, Applied and Computational Harmonic Analysis, 21 (2006), pp. 95–112.

[8] F. Chung and L. Lu, *Complex Graphs and Networks*, no. 107 in CBMS Regional Conference Series in Mathematics, American Mathematical Society, Providence, RI, 2006.

[9] A. Cichocki and A.-H. Phan, *Fast local algorithms for large scale nonnegative matrix and tensor factorizations*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 92 (2009), pp. 708–721.

[10] R. R. Coifman and M. Gavish, *Harmonic analysis of digital data bases*, in Wavelets and Multiscale Analysis: Theory and Applications, J. Cohen and A. I. Zayed, eds., Applied and Numerical Harmonic Analysis, Boston, MA, 2011, Birkhäuser, pp. 161–197.

[11] R. R. Coifman and M. Maggioni, *Diffusion wavelets*, Applied and Computational Harmonic Analysis, 21 (2006), pp. 53–94.

[12] R. R. Coifman, Y. Meyer, and V. Wickerhauser, *Wavelet analysis and signal processing*, in In Wavelets and their Applications, M.B.Ruskai et al., eds., Jones and Bartlett, 1992, pp. 153–178.

[13] R. R. Coifman and M. V. Wickerhauser, *Entropy-based algorithms for best basis selection*, IEEE Transactions on Information Theory, 38 (1992), pp. 713–718.

[14] I. S. Dhillon, *Co-clustering documents and words using bipartite spectral graph partitioning*, in Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2001, pp. 269–274.

[15] S. S. Du, Y. Liu, B. Chen, and L. Li, *Maxios: Large scale nonnegative matrix factorization for collaborative filtering*, in Proceedings of the NIPS 2014 Workshop on Distributed Matrix Computations, 2014.

[16] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning and a Highly Connected World*, Cambridge University Press, New York, 2010.

[17] H. Fang, Z. Zhang, Y. Shao, and C.-J. Hsieh, *Improved bounded matrix completion for large-scale recommender systems.*, in IJCAI, 2017, pp. 1654–1660.

[18] M. Fiedler, *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory*, Czechoslovak Mathematical Journal, 25 (1975), pp. 619–633.

[19] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, *Large-scale matrix factorization with distributed stochastic gradient descent*, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 69–77.

[20] S. Georgiou, C. Koukouvinos, and J. Seberry, *Hadamard matrices, orthogonal designs and construction algorithms*, in Designs 2002, Springer, 2003, pp. 133–205.

[21] E. F. Gonzalez and Y. Zhang, *Accelerating the Lee-Seung algorithm for nonnegative matrix factorization*, tech. rep., 2005.

[22] S. E. Grigorescu, N. Petkov, and P. Kruizinga, *Comparison of texture features based on Gabor filters*, IEEE Transactions on Image Processing, 11 (2002), pp. 1160–1167.

[23] L. Hagen and A. B. Kahng, *New spectral methods for ratio cut partitioning and clustering*, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 11 (1992), pp. 1074–1085.

[24] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 53 (2011), pp. 217–288.

[25] D. K. Hammond, P. Vandergheynst, and R. Gribonval, *Wavelets on graphs via spectral graph theory*, Applied and Computational Harmonic Analysis, 30 (2011), pp. 129–150.

[26] F. M. Harper and J. A. Konstan, *The movielens datasets: History and context*, Acm Transactions on Interactive Intelligent Systems, 5 (2015), pp. 1–19.

[27] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical learning with sparsity: the lasso and generalizations*, CRC press, 2015.

[28] C.-J. Hsieh and P. Olsen, *Nuclear norm minimization via active subspace selection*, in International Conference on Machine Learning, 2014, pp. 575–583.

[29] J. Irion, *Multiscale Transforms for Signals on Graphs: Methods and Applications*, Ph.D. dissertation, University of California, Davis, 2015.

[30] J. Irion and N. Saito, *The generalized Haar-Walsh transform*, in Proceedings of 2014 IEEE Workshop on Statistical Signal Processing (SSP), IEEE, 2014, pp. 472–475.

[31] J. Irion and N. Saito, *Applied and computational harmonic analysis on graphs and networks*, in Wavelets and Sparsity XVI, Proc. SPIE 9597, M. Papadakis, V. K. Goyal, and D. Van De Ville, eds., 2015. Paper # 95971F.

[32] ———, *Learning sparsity and structure of matrices with multiscale graph basis dictionaries*, in Proc. 2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), A. Uncini, K. Diamantaras, F. A. N. Palmieri, and J. Larsen, eds., 2016.

[33] J. Irion and N. Saito, *Efficient approximation and denoising of graph signals using the multiscale basis dictionaries*, IEEE Transactions on Signal and Information Processing over Networks, 3 (2017), pp. 607–616.

[34] M. Jansen, G. P. Nason, and B. W. Silverman, *Multiscale methods for data on graphs and irregular multidimensional situations*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 71 (2009), pp. 97–125.

[35] I. T. Jolliffe, *Principal Component Analysis and Factor Analysis*, Springer New York, New York, NY, 1986, pp. 115–128.

[36] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst, *Matrix completion on graphs*, in Neural Information Processing Systems workshop "Out of the Box: Robustness in High Dimension", 2014.

[37] R. Kannan, M. Ishteva, B. Drake, and H. Park, *Bounded matrix low rank approximation*, in Non-negative Matrix Factorization Techniques, Springer, 2016, pp. 89–118.

[38] Y. Koren, *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 426–434.

[39] Y. Koren, R. Bell, and C. Volinsky, *Matrix factorization techniques for recommender systems*, IEEE Computer, 42 (2009), pp. 30–37.

[40] R. M. Larsen, *Lanczos bidiagonalization with partial reorthogonalization*, DAIMI Report Series, (1998).

[41] A. Lee, B. Nadler, and L. Wasserman, *Treelets—an adaptive multi-scale basis for sparse unordered data*, Annals of Applied Statistics, 2 (2008), pp. 435–471.

[42] D. D. Lee and H. S. Seung, *Learning the parts of objects by non-negative matrix factorization*, Nature, 401 (1999), pp. 788–791.

[43] ———, *Algorithms for non-negative matrix factorization*, in Advances in Neural Information Processing Systems, 2001, pp. 556–562.

[44] C.-J. Lin, *Projected gradient methods for nonnegative matrix factorization*, Neural Computation, 19 (2007), pp. 2756–2779.

[45] D. Lin, A. Arslan, T. Holy, et al., *NMF.jl*. `https://github.com/JuliaStats/NMF.jl`, 2018.

[46] Z. Lin, R. Liu, and Z. Su, *Linearized alternating direction method with adaptive penalty for low-rank representation*, in Advances in Neural Information Processing Systems, 2011, pp. 612–620.

[47] M. Lindberg and L. F. Villemoes, *Image compression with adaptive Haar-Walsh tilings*, in Wavelet Applications in Signal and Image Processing VIII, Proc. SPIE 4119, International Society for Optics and Photonics, 2000, pp. 911–922.

[48] L. Lovász, *Large Networks and Graph Limits*, vol. 60 of Colloquium Publications, American Mathematical Society, Providence, RI, 2012.

[49] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 3rd ed., 2009.

[50] R. Mazumder, T. Hastie, and R. Tibshirani, *Spectral regularization algorithms for learning large incomplete matrices*, Journal of Machine Learning Research, 11 (2010), pp. 2287–2322.

[51] F. Murtagh, *The Haar wavelet transform of a dendrogram*, Journal of Classification, 24 (2007), pp. 3–32.

[52] M. Newman, *Networks*, Oxford University Press, 2nd ed., 2018.

[53] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, *Graph signal processing: Overview, challenges, and applications*, Proceedings of the IEEE, 106 (2018), pp. 808–828.

[54] B. Recht, M. Fazel, and P. A. Parrilo, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, SIAM review, 52 (2010), pp. 471–501.

[55] D. K. Ruch and P. J. Van Fleet, *Wavelet Theory: an Elementary Approach with Applications*, John Wiley & Sons, 2011.

[56] R. M. Rustamov, *Average interpolating wavelets on point clouds and graphs*, ArXiv Preprint ArXiv:1110.2227, (2011).

[57] N. Saito and R. R. Coifman, *Extraction of geological information from acoustic well-logging waveforms using time-frequency wavelets*, Geophysics, 62 (1997), pp. 1921–1930.

[58] Y. Shao and N. Saito, *The extended generalized Haar-Walsh transform and applications*, in Wavelets and Sparsity XVIII, vol. 11138, International Society for Optics and Photonics, 2019, p. 111380C.

[59] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888–905.

[60] 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*, IEEE Signal Processing Magazine, 30 (2013), pp. 83–98.

[61] S. Sra and I. S. Dhillon, *Generalized nonnegative matrix approximations with Bregman divergences*, in Advances in Neural Information Processing Systems, 2006, pp. 283–290.

[62] E. M. Stein and R. Shakarchi, *Fourier Analysis: An Introduction*, vol. 1, Princeton University Press, 2011.

[63] A. D. Szlam, M. Maggioni, and R. R. Coifman, *Regularization on graphs with function-adapted diffusion processes*, Journal of Machine Learning Research, 9 (2008), pp. 1711–1739.

[64] A. D. Szlam, M. Maggioni, R. R. Coifman, and J. C. Bremer, Jr., *Diffusion-driven multiscale analysis on manifolds and graphs: top-down and bottom-up constructions*, in Wavelets XI, Proc. SPIE 5914, M. Papadakis, A. F. Laine, and M. A. Unser, eds., vol. 5914, International Society for Optics and Photonics, 2005, p. 59141D.

[65] C. M. Thiele and L. F. Villemoes, *A fast algorithm for adapted time–frequency tilings*, Applied and Computational Harmonic Analysis, 3 (1996), pp. 91–99.

[66] M. Udell, C. Horn, R. Zadeh, S. Boyd, et al., *Generalized low rank models*, Foundations and Trends® in Machine Learning, 9 (2016), pp. 1–118.

[67] M. Udell and A. Townsend, *Why are big data matrices approximately low rank?*, SIAM Journal on Mathematics of Data Science, 1 (2019), pp. 144–160.

[68] U. Von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing, 17 (2007), pp. 395–416.

[69] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon, *Scalable coordinate descent approaches to parallel matrix factorization for recommender systems*, in 2012 IEEE 12th International Conference on Data Mining, IEEE, 2012, pp. 765–774.