USC-SIPI REPORT #403

Lifting Transforms on Graphs: Theory and Applications

by

Godwin Shen

August 2010

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

LIFTING TRANSFORMS ON GRAPHS: THEORY AND APPLICATIONS

by

Godwin Shen

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

August 2010

Dedication

To all of my wonderful friends and family.

Acknowledgments

I would first like to thank my advisor, Professor Antonio Ortega, whose wonderful, tireless guidance has shaped my ideas about research, and moreover, has helped to re-shape and refine my approach to writing, analytical / critical thinking and problem solving. Thanks is also due to Professor Bhaskar Krishnamachari and Professor Ramesh Govindan for serving on my dissertation committee, as well as to Professor C.-C. Jay Kuo and Professor Alexandros Dimakis for being members in my qualifying exam committee. It is a privilege to have their advice on my work.

I would also like to thank Hua Xie, Samuel Dolinar, Matthew Klimesh, Aaron Kiely and Michael Cheng from Jet Propulsion Laboratory, who have provided great support throughout my research studies. Interactions with you have been wonderful. I am also grateful for the many fruitful discussions and fun times we had during my summer in Pasadena. I also owe thanks to Jaejoon Lee and HoCheon Wey from Samsung Electronics Co., Ltd. Your support during the final year of my research has been greatly appreciated.

Special thanks are also due to my colleagues in the Compression Research Group. It has been a pleasure working with all of you and I have enjoyed the numerous discussions we had about research, work and life in general. In particular, I would like to thank Sunil Narang for all of our memorable research collaborations and for teaching me everything I know about spectral graph theory. I also owe thanks to Woo-shik Kim for our collaborations and for teaching me so much about

video compression. I would also like to thank Ivy Tseng, Polin Lai, Insuk Chong and Roger Pique for all of their advice, guidance and support. I also owe a special thanks to Sean McPherson, who has been a great colleague and friend through all of my time spent at USC.

From the Autonomous Networks Research Group, I owe special thanks to Prof. Bhaskar Krishnamachari, Sundeep Pattem and Ying Chen for all of the unforgettable years of collaboration. The time spent in our joint efforts has truly enriched my experience at USC. I would also like to thank Paula Tarrío from Universidad Politécnica de Madrid, Giuseppe Valenzise from Politecnico di Milano, Alfonso Sánchez from Universidad Politécnica de Catalunya and Javier Perez Trufero from Universidad Politécnica de Catalunya for all of the wonderful collaborations we have undertaken.

I would also like to thank my mother Elena Shen and father Jen-Chi Kung for their endless love, patience, guidance and support throughout my life. Special thanks also to my brother Ernest Shen and to the rest of my family who have always given me so much love and support. Finally, I would like to thank Vanessa Hadikusumah for all of her love and support.

Table of Contents

List Of Tables

List Of Figures

Abstract

There are many scenarios in which data can be organized onto a graph or tree. Data may also be similar across neighbors in the graph, e.g., data across neighboring sample points may be spatially correlated. It would therefore be useful to apply some form of transform across neighboring sample points in the graph to exploit this correlation in order to achieve more compact representations. To this end, we describe a general class of de-correlating lifting transforms that can be applied to any graph or tree, and propose a variety of transform optimizations. We mainly focus on the design of tree-based lifting transform designs. Extensions to graph-based lifting transforms are also discussed. As a first application, we develop distributed graph-based transforms for efficient data gathering in wireless sensor networks (WSNs), where the goal is to transmit data from every node in the network to a collection (or sink) node along a routing tree. In particular, we (i) propose a general class of *unidirectional transforms* that can be computed in a distributed manner as data is routed toward the sink, and (ii) provide conditions for their invertibility. Moreover, we show that any unidirectional lifting transform is invertible, and propose a variety of tree-based lifting transform designs. By using these transforms to de-correlate data in the network, the total communication cost for data gathering is significantly reduced. We also extend these tree-based lifting transforms to incorporate broadcast communication links. This leads to

a set of graph-based lifting transforms for WSNs. In particular, nodes incorporate data received from their broadcast neighbors together with data received from their neighbors in the routing tree. By doing so, they are able to achieve more data de-correlation. By exploiting the additional broadcast communication links in this way, these graph-based lifting transforms reduce the total communication cost even further. In addition to the transform designs, we also propose an algorithm that can jointly optimize the choice of routing tree with the transform. As a second application, we also develop graph-based transforms for image compression. In particular, we focus on designing graph-based transforms that avoid filtering across edges in an image. This reduces the number of large magnitude coefficients, which are expensive to code, and ultimately reduces the total bit rate while also preserving better the edge structure in the reconstructed images. To this end, we first discuss how our tree-based lifting transforms generalize existing wavelet transforms proposed for image coding, then propose algorithms to design the trees and transforms. By avoiding filtering across edges, our tree-based lifting transforms yield better coding efficiency than standard transforms (i.e., the total bit rate is reduced for a fixed reconstruction quality). We also develop an edge-adaptive intra prediction scheme that avoids computing predictions across edges in an image/video frame. Since predictions are not computed across edges, our scheme significantly reduces the number of large magnitude coefficients that must be coded. This new scheme is then incorporated with the intra prediction scheme in H.264/AVC, and is shown to increase the overall coding efficiency of H.264/AVC. Moreover, when using this new scheme to code depth map images in a multi-view video coding system (where virtual views are synthesized using video plus depth from existing views), we also see an improvement in the quality of the virtual views.

Chapter 1

Introduction

There are many applications in which data can be organized on a graph $G = (V, E)$ or tree $T = (V, E')$, where the set of tree edges E' is a subset of E. As an example, consider a wireless sensor network (WSN) [4] where nodes are able to communicate with each other (possibly via broadcast) through wireless data transmissions. In this case, nodes serve as vertices, V, and the communication links that connect neighboring nodes form the set of edges E. A concrete example of this is illustrated in Figure 1.1, where nodes are organized onto a routing tree T. Even for standard such as images a graph interpretation is possible, i.e., we can view the pixels in the image grid as vertices in a graph or tree. Edges between pixels can then be defined, for instance, as connections between 4-connected or 8-connected neighbors. As we will soon see, a graph-based representation is useful since it allows us to generalize standard signal processing operations (e.g., filtering, transforms, de-noising) to different types of data. In this chapter, we first provide some motivation and preliminaries for a graph-based representation in Section 1.1. We then give an overview of the work proposed in this thesis in Sections 1.2, 1.3, 1.4 and 1.5.

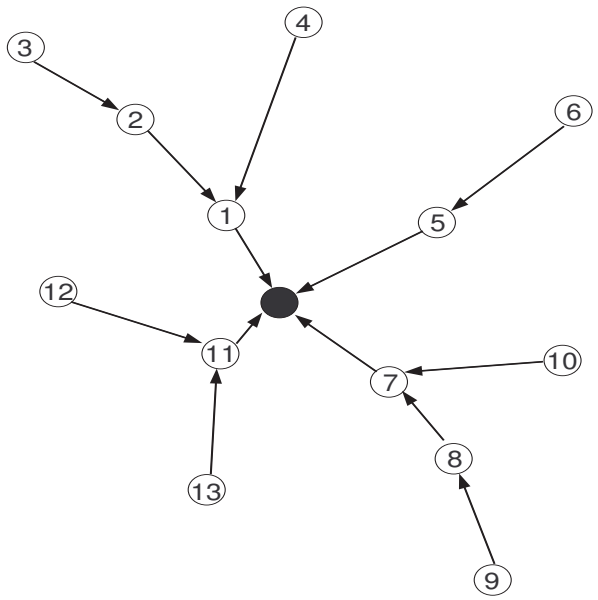


Figure 1.1: Irregularly spaced nodes organized onto a rooted tree.

1.1 Motivation

We now describe the basic graph-based representation and provide some motivation for the work in this thesis. Let $\{x(n)\}_{n=1}^{N}$ denote a finite-dimensional signal and suppose that the N values are taken from some arbitrary sampling grid. The points in the sampling grid (i.e., the *sample points*) can be organized as vertices V in a graph G, with each vertex corresponding to, for example, a point in Euclidean space. The connectivity between vertices can then be represented by edges E in the graph, with the "neighbors" of each vertex defined as other vertices that are connected to it (e.g., its "one-hop" neighbors in the graph). This leads to a *graph-based signal representation* for $x(n)$.

Note that for standard signals (e.g., digital audio signals and images), samples are typically placed on a regularly spaced sampling grid on the real line $\mathbb{R}$ or the real plane $\mathbb{R}^2$. There are many signal processing tools available for regularly spaced data, e.g., filters for processing [39], transforms for compression and analysis [22,28,47,64].

However, *none of these tools can be used for irregularly spaced sampling grids*. Thus, the utility of this graph-based representation is two-fold. First, it allows us to generalize existing signal processing techniques to irregularly spaced data. Since irregularly spaced sampling grids are common to many applications (e.g., WSNs [14], distributed databases [18], and de-noising of scattered data [23, 36]), one overarching theme of this thesis is to develop signal processing tools that can be applied to irregularly spaced data by utilizing this graph-based representation. Secondly, graph-based representations for regularly spaced data can also be used to develop transforms that are adapted to the underlying signal structure, e.g., discontinuities (or edges) in an image. A set of "edge-adaptive" transforms is also described in this thesis.

In standard applications such as image processing and compression, filtering is typically done along 1D paths. For example, in JPEG [42] and JPEG-2000 [67] filtering is performed along the rows and columns of an image. Filtering could also be applied along 1D paths that are oriented diagonally as in the Directional DCT [78], Directionlets [71] and Bandelets [43]. Note that in all of these (and related) techniques *filtering is done along non-overlapping 1D paths*. While these are reasonable strategies for signals on regular sampling grids, they are not easily extended to signals on irregular sampling grids. For irregular grids, it is more sensible to form the sample points onto a graph or tree, then to perform filtering along these graphs or trees. In particular, note that filtering along 1D paths is just a special case of filtering along trees with non-overlapping paths (i.e., every node in the tree has at most one child). On the other hand, more general trees will have have multiple 1D paths that merge together (i.e., nodes can have more than one child). Therefore, in this thesis the primary focus is to develop techniques for filtering along arbitrary trees. This provides a more general framework for filtering

since *data can be filtered along overlapping 1D paths*. We mainly focus on a *tree-based signal representation* and develop algorithms for processing data along these trees, then discuss how to extend these algorithms to more general graphs.

Of the wide variety of signal processing techniques that can be generalized using a tree-based (and graph-based) signal representation, we only focus on designing de-correlating linear transforms for data compression. Linear transforms are important since data across neighboring sample points is typically well correlated, e.g., neighboring pixels in an image tend to have similar intensity, temperature readings tend to be very similar across neighboring nodes in a WSN, etcetera. Thus, applying a de-correlating linear transform (e.g., a DCT or wavelet transform) to the data will allow us to reduce the amount of information (i.e., bits) needed to represent it. We would also like the transforms to be *localized* in the sense that the transform only operates on sets of sample points that are physically close together. Let $\mathbf{x} = [x(1), x(2), \ldots, x(N)]^t$ be a vector containing the measurements of N nodes in a graph. An example of a transform which is localized for the tree shown in Figure 1.1 is

$$\mathbf{y} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_5 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_7 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{A}_{11} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_5 \\ \mathbf{x}_7 \\ \mathbf{x}_{11} \end{bmatrix},$$

where $\mathbf{x}_1 = [x(1), x(2), x(3), x(4)]^t$, $\mathbf{x}_5 = [x(5), x(6)]^t$, $\mathbf{x}_7 = [x(7), x(8), x(9), x(10)]^t$ and $\mathbf{x}_{11} = [x(11), x(12), x(13)]^t$. Note that the transform operations on nodes 1 through 4 only involve data from nodes 1 to 4, hence, the transform is localized to nodes 1 to 4. Similar logic follows for other nodes. These *local transforms* are

also convenient since they can be computed in a distributed manner, thus, they are easily amenable to distributed compression in WSNs. Moreover, we can easily adapt localized transforms to local signal structures such as discontinuities. This is useful for image compression. In this work, we mainly focus on developing tree-based lifting transforms since (i) they have been shown to provide good de-correlation for multiple applications and (ii) they are local transforms.

The transform designs in this thesis are geared toward two applications. The first is distributed data gathering for WSN, where the goal is to gather data from every node in the network at a central collection (or sink) node. In this application, de-correlating linear transforms are often computed in the network in order to reduce the amount of data nodes must transmit to a central collection (or sink) node. This reduces the total communication cost that nodes incur while transporting data to the sink node. Tree-based transforms are developed for these WSNs as well as graph-based extensions that incorporate broadcast wireless communication links. The second application is image compression, where tree-based transforms are developed in which the paths in the tree do not cross over discontinuities (i.e., edges) in the image. Since filtering along these trees avoids filtering across edges, fewer large magnitude coefficients are produced. Thus, fewer bits are needed to represent the transform coefficients.

1.2 Lifting Transforms on Graphs

In terms of transform designs, our primary focus is on the design of wavelet transforms constructed on graphs and trees using lifting [65]. Lifting transforms are invertible by construction, and are fully specified by (i) a *split* step which divides the sample points into an *even* and *odd* set, (ii) a *prediction* step that filters data

from odd sample points with data from even ones (yielding detail coefficients), and (iii) an *update* step which filters data from even sample points with detail coefficients from odd ones. Since arbitrary split, prediction and updates can be used without affecting invertibility, *these transforms are very flexible. They can also be made local* by only allowing odd (resp. even) sample points to use data from their even (resp. odd) neighbors in the graph. Furthermore, these lifting transforms have consistently shown excellent performance in many applications, e.g., in data de-noising [23, 36] and distributed data gathering [72]. In this thesis we describe a framework that encompasses these tree-based and graph-based lifting transforms. We primarily focus on tree-based lifting transform designs, where the splitting is done along a tree and filtering operations are only performed across neighbors in the tree, though we have also proposed graph-based splitting schemes [37]. Similar techniques have also been proposed [23, 36]. Thus, we also compare tree-based and graph-based lifting transforms. In terms of *novel contributions*, we propose new tree-based lifting transforms with (i) a new split design along an arbitrary tree, (ii) an extension of existing adaptive prediction filters to WSNs and (iii) a novel update filter design.

1.3 Transform-based Distributed Data Gathering

In terms of applications, we first apply these lifting transforms to data gathering in WSN. Note that computing the transforms in [23,36,72,73] in a distributed manner will require nodes to make many (local) data transmissions before coefficients can be encoded and transferred to the collection node along an efficient routing tree. An example of this is shown in Figure 1.2. As such, these strategies are not very efficient in terms of energy expenditure. Instead, it would be better to design

transforms that can be computed as data is being routed to the sink node. This will eliminate the need for excessive local data transmissions, thereby leading to distributed implementations that are more energy-efficient.

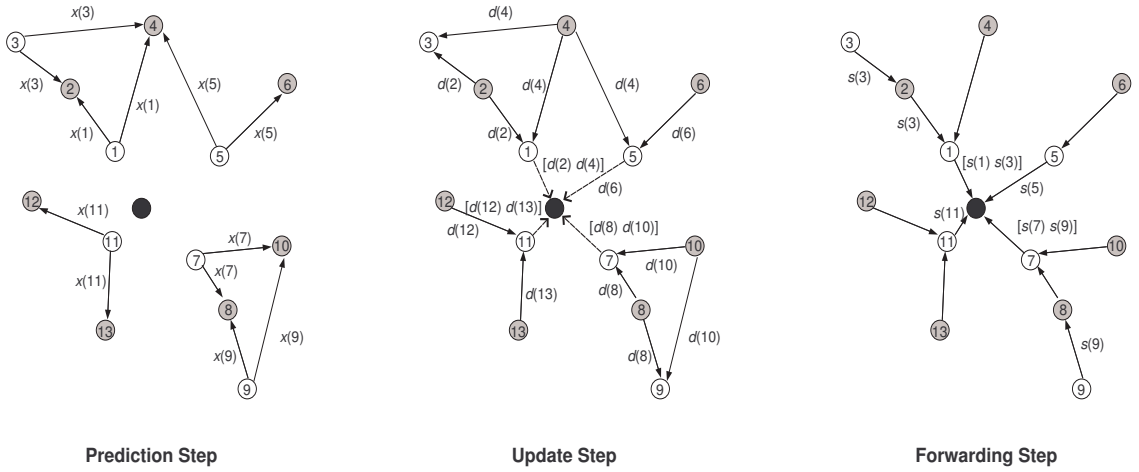


Figure 1.2: Example illustrating the communications required to compute the transforms in [23, 36, 72, 73] in a distributed manner. White nodes are even and gray nodes are odd. First even nodes must transmit data to their odd neighbors. Odd nodes receive even node data, compute transform coefficients, then transmit those coefficients back to their even neighbors (and also to the sink). Even nodes then use these odd node coefficients to compute their own coefficients, then transmit them to the sink. Note that even nodes must transmit their own data twice.

In this thesis we (i) design a general class of transforms (not restricted to lifting) that can be computed in a distributed manner as data is routed to the sink node, then (ii) provide lifting transform designs that fall into this general class. While these designs are specific to WSN, they can be easily extended to other applications. We assume that data is forwarded along the tree according to a transmission scheduling and that for transform computations sensor nodes can only use data that they receive before they transmit. This includes data that sensor nodes receive along the tree, i.e., from descendants, as well as data overheard via broadcast communications. This leads to a general class of transforms which we refer to as *unidirectional transforms*; a general definition and invertibility conditions are given

in Chapter 3. We then describe how to translate existing unidirectional transforms into our framework in order to demonstrate its generality. Finally, novel energy-efficient lifting transforms are proposed that provide superior performance over existing methods.

1.4 Joint Optimization of Transform and Routing

The choice of routing tree also impacts the performance of unidirectional transforms. For example, from a routing perspective the best trees are the well-known shortest path routing trees [12] (SPTs). While SPT routing provides the minimum cost to route a fixed amount of data to the sink, transforms computed along these trees will not always provide the most compact representation of the underlying data. In particular, an SPT will provide the minimum distance from any node to the root, but will not necessarily minimize the distance between each node and its neighbors in the tree. As was pointed out in our previous work [54], if data correlation is inversely proportional to distance, a minimum spanning tree [12] (MST) which minimizes the sum of the distances between neighboring nodes will provide a more compact representation of the data than an SPT. Thus, routing with compression on a shortest path routing tree (SPT) will not necessarily provide the minimum total cost. Another major contribution of this thesis is a joint routing and transform optimization algorithm, as described in Chapter 4.

1.5 Graph-based Transforms for Image Coding

The second application of this thesis is image coding. Correlation across neighboring pixels in an image is typically exploited in one of two ways. First, separable

filtering (i.e., filtering is done first along rows, then along columns) is typically applied, e.g., pixel data is filtered using separable DCT bases [47] as in JPEG [42], or with separable wavelet bases as in JPEG-2000 [67]. These bases can represent smooth images with few horizontal, vertical and diagonal discontinuities very compactly. However, for images with complex discontinuities, these transforms produce many large magnitude high-pass coefficients. Large magnitude high-pass components require many bits to be encoded, i.e., they increase the total bit rate. Furthermore, quantization of these large high-pass components leads to annoying compression artifacts such as ringing. One way to deal with this is to construct transforms along graphs that either avoid discontinuities or perform filtering parallel to them. When doing so, the number of large magnitude high-pass coefficients can be significantly reduced. The structure of the discontinuities will also be well preserved, thereby reducing the amount of ringing artifacts. In Chapter 5 we construct trees that do not have links between pixels that are separated by discontinuities, then we design transforms along these trees. We apply these transforms to natural and depth map images, and see performance that is superior to standard separable transforms.

Another way correlation in images is typically exploited is through block-based intra prediction schemes used in, for example, H.264/AVC and MPEG-4. The prediction is typically done along a fixed set of directions and the "best" direction is chosen as the final "intra prediction mode". While these directional prediction methods can provide accurate predictions of blocks with a single diagonal edge (and therefore, can provide low energy prediction residuals for these blocks), they do not provide accurate for blocks with more complex edge structure such as "L" or "V" shaped edges. Thus, we also develop an edge-adaptive intra prediction scheme that can be easily integrated with existing techniques. When applied to intra predictive

coding of depth map images, we see significant gains with respect to existing intra prediction schemes.

1.6 Outline

This thesis is organized as follows. First we provide an overview of lifting transforms in Chapter 2. We then describe the data gathering problem for WSN and propose a general framework for efficient de-correlating transforms that can be computed in the network in Chapter 3. Various tree-based and graph-based transforms are also proposed in Chapter 3. In Chapter 4 we also propose a joint routing and transform optimization method for WSN. In Chapter 5 we propose a variety of tree-based and graph-based transforms for image compression. Finally, some concluding remarks and interesting directions for future work are discussed in Chapter 6.

Chapter 2

Lifting Transforms on Graphs

We now focus on the construction and optimization of tree-based and graph-based lifting transforms. As a starting point, in Section 2.1 we establish some definitions and notation for lifting transforms [65] and show that these transforms are invertible by construction. These transforms consist of three key components. The first is a *split step* that divides nodes into disjoint sets of even and odd nodes. *Prediction filters* must also be designed with which data at odd nodes is linearly predicted from data at even nodes (yielding detail coefficients). Finally, *update filters* are used to linearly update data at even nodes using detail coefficients from odd nodes (yielding smooth coefficients). Multiple prediction and update steps can be used. We initially make no assumption about the relationships between these nodes, i.e., we do not assume anything about the structure of the graph nor is there any notion of relative position or distance, though some notion of this would be useful when defining the filtering operations used in the transform. Therefore, the lifting transforms presented as such are very general.

In Section 2.2, two split design procedures are discussed that can be applied to an arbitrary rooted tree or to an arbitrary graph. These designs were first introduced by us in [37,55]. We note that data de-correlation occurs in the prediction step, i.e.,

if an appropriate choice of prediction filter is made for each odd node, the prediction made from its neighbors' data will be very close to its own data, hence, the resulting prediction residual (i.e., detail coefficient) will be close to zero. This is useful since small prediction residuals require very few bits to be encoded. Naturally, the choice of prediction filter depends on the properties of the given data. Thus, in Section 2.3 we will discuss prediction filter designs that minimize the average energy in the prediction residuals. This result and an algorithm for learning these prediction filters were described by the author in [52].

When a prediction step is applied without any update step, issues like numerical instability of the inverse transform [23, 67] and propagation of quantization errors [16, 67] will arise. This will reduce the quality of the reconstructed data. Thus, it is also important to include an update step to mitigate these effects. It is also desirable to design update filters that have certain properties such as preserving the average value of coefficients across multiple levels of decomposition or orthogonality between low-pass (i.e., update) and high-pass (i.e., prediction) filters. Various update filter designs are discussed in Section 2.4. In particular, we propose an update filter design that makes the low-pass and high-pass filters orthogonal. This result was first introduced by us in [56]. Furthermore, we show that this choice of update filters also minimizes the reconstruction MSE due to quantization of the transform coefficients.

2.1 Preliminaries

Lifting transforms are computed by splitting (i.e., partitioning) nodes into even and odd sets, filtering data from odd nodes with data from even nodes to produce detail coefficients, and then filtering data from even nodes with details coefficients from

Parameter	Description
N	Number of sample points (nodes)
$\mathcal{I}$	Set of node indices
$\mathbf{e}_i$	i-th identity vector, $\mathbf{e}_i(i) = 1$, $\mathbf{e}_i(j) = 0$ for all $j \neq i$
$\mathbf{I}$	Identity matrix
$\mathbf{0}$	All zero vector
$x(i)$	Data at node $i \in \mathcal{I}$
$\hat{x}(i)$	Prediction of $x(i)$
$\mathbf{x}$	Vector of original data
$\mathbf{T}$	Lifting transform matrix
$\mathbf{y}$	Vector of lifting transform coefficients
$\mathcal{E}$ and $\mathcal{O}$	Set of even and odd nodes ($\mathcal{E} \cap \mathcal{O} = \emptyset$)
$\mathcal{N}_i$	Set of even (odd) neighbors of odd (even) node i
$\mathbf{p}_n$	Prediction vector (filter) for odd node n
$d(n)$	Detail coefficient of odd node n
$\mathbf{P}$	Prediction matrix
$\mathbf{d}$	Vector of detail coefficients
$\mathbf{u}_m$	Update vector (filter) for even node m
$s(m)$	Smooth coefficient of even node m
$\mathbf{U}$	Update matrix

Table 2.1: Table of common notation.

odd nodes. As we will soon show, since data at odd nodes is only filtered using data from even nodes and vice versa, the corresponding transform is guaranteed to be invertible. We first establish some notation that will be throughout the remainder of this chapter. The notation is summarized in Table 2.1.

Suppose that there are N sample points indexed by $\mathcal{I} = \{1, 2, \ldots, N\}$. Let $x(n)$ denote the value at sample point n, and let $\mathbf{x} = [x(1), x(2), \ldots, x(N)]^t$. Let $\mathcal{E}$ and $\mathcal{O}$ be two disjoint sets of even and odd nodes, respectively. For each odd node n, let $\mathcal{N}_n \subset \mathcal{E}$ be the set of even neighbors of node n. The prediction vector $\mathbf{p}_n$ is used to produce a prediction as $\sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m)x(m)$ and this prediction is subtracted from $x(n)$ to yield detail coefficient $d(n) = x(n) - \sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m)x(m)$. No predictions are performed for even node data, thus, $\mathbf{p}_m = \mathbf{0}$ for all $m \in \mathcal{E}$, where $\mathbf{0}$ is the all zero vector. Since data for odd node n is only filtered using even node

data in $\mathcal{N}_n$, we also have that $\mathbf{p}_n(k) = 0$ for all $k \in \mathcal{O} \cup (\mathcal{E} - \mathcal{N}_n)$. For each even node m, let $\mathcal{N}_m \subset \mathcal{O}$ denote the set of odd neighbors of node m. Data from each even node m is then updated using update vector $\mathbf{u}_m$, yielding smooth coefficient $s(m) = x(m) + \sum_{n \in \mathcal{N}_m} \mathbf{u}_m(n) d(n)$. Since odd node data is not updated, $\mathbf{u}_n = \mathbf{0}$ for all $n \in \mathcal{O}$. Since data from even node m is only filtered with odd node data from $\mathcal{N}_m$, we also have that $\mathbf{u}_m(l) = 0$ for all $l \in \mathcal{E} \cup (\mathcal{O} - \mathcal{N}_m)$. This is all summarized in the following definition.

Definition 1 (Single Step Lifting Transform)**.** *Let there be N data points $x(n)$ indexed by $n \in \mathcal{I} = \{1, 2, \ldots, N\}$. Let $\mathcal{I}$ be partitioned into two disjoint sets of* even *and* odd *nodes denoted by $\mathcal{E}$ and $\mathcal{O}$ respectively, i.e., $\mathcal{I} = \mathcal{E} \cup \mathcal{O}$ and $\mathcal{E} \cap \mathcal{O} = \emptyset$. For each $m \in \mathcal{E}$, let $\mathcal{N}_m \subset \mathcal{O}$. Similarly, for each $n \in \mathcal{O}$, $\mathcal{N}_n \subset \mathcal{E}$. A* single step lifting transform *is a linear prediction step followed by a linear update step. Let $\mathbf{p}_n$ denote the $N \times 1$ prediction operator used for node n and let $\mathbf{u}_m$ denote the $N \times 1$ update operator for node m. The lifting transform is computed in the prediction step first, yielding detail coefficients for each $n \in \mathcal{O}$ as*

$$d(n) = x(n) - \sum_{i \in \mathcal{N}_n} \mathbf{p}_n(i) x(i). \tag{2.1}$$

In the update step, smooth coefficients are computed for each $m \in \mathcal{E}$ as

$$s(m) = x(m) + \sum_{j \in \mathcal{N}_m} \mathbf{u}_m(j) d(j). \tag{2.2}$$

Since $\mathcal{N}_n \subset \mathcal{E}$ for all $n \in \mathcal{O}$, we have that $\mathbf{p}_n(j) = 0$, for all $j \in \mathcal{O} \cup (\mathcal{E} - \mathcal{N}_n)$. Similarly, $\mathcal{N}_m \subset \mathcal{O}$ for all $m \in \mathcal{E}$, hence, $\mathbf{u}_m(j) = 0$, for all $j \in \mathcal{E} \cup (\mathcal{O} - \mathcal{N}_m)$. Moreover, $\mathbf{p}_m = \mathbf{0}$, for $m \in \mathcal{E}$ and $\mathbf{u}_n = \mathbf{0}$, for $n \in \mathcal{O}$.

Note that these operations correspond to a set of N vector inner products. For example, if $\mathbf{e}_n$ represents the n-th identity vector (i.e., $\mathbf{e}_n(n) = 1$ and $\mathbf{e}_n(l) = 0$ for all $l \neq n$), then for each $n \in \mathcal{O}$, $x(n) = \mathbf{e}_n^t \cdot \mathbf{x}$ and $\sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m)x(m) = \mathbf{p}_n^t \cdot \mathbf{x}$. Thus, $d(n) = (\mathbf{e}_n - \mathbf{p}_n)^t \cdot \mathbf{x}$. Therefore, we can also express the transform as a single matrix operation $\mathbf{y} = (\mathbf{I} + \mathbf{U}) \cdot (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$ as shown in Proposition 1 (see Appendix A for the proof).

Proposition 1 (Lifting Transform Matrices)**.** *Let the vectors* $\mathbf{u}_n$ *and* $\mathbf{p}_n$ *satisfy Definition 1 for all* n*. Let* $\mathbf{P}$ *be the* $N \times N$ *prediction matrix with* $row_n(\mathbf{P}) = \mathbf{p}_n^t$*. Similarly, let* $\mathbf{U}$ *be the* $N \times N$ *update matrix with* $row_n(\mathbf{U}) = \mathbf{u}_n^t$*. The lifting transform matrix is simply* $\mathbf{T} = (\mathbf{I} + \mathbf{U}) \cdot (\mathbf{I} - \mathbf{P})$ *and we can compute the vector of coefficients as* $\mathbf{y} = \mathbf{T} \cdot \mathbf{x}$*.*

Note that the non-zero filter coefficients are completely unconstrained in Definition 1, thus, this represents a very general class of transforms. The inverse of $\mathbf{I} - \mathbf{P}$ and $\mathbf{I} + \mathbf{U}$ also exist by construction and are easily shown to be $\mathbf{I} + \mathbf{P}$ and $\mathbf{I} - \mathbf{U}$, respectively, as shown in Proposition 2 (the proof is in Appendix A).

Proposition 2 (Inverse Lifting Transform Matrices)**.** *Let* $\mathcal{E}$ *and* $\mathcal{O}$ *satisfy Definition 1, and let* $\mathbf{P}$ *and* $\mathbf{U}$ *satisfy the assumptions in Proposition 1. Then* $(\mathbf{I}-\mathbf{P})^{-1} = \mathbf{I} + \mathbf{P}$ *and* $(\mathbf{I} + \mathbf{U})^{-1} = \mathbf{I} - \mathbf{U}$*.*

We can also introduce non-zero normalization factors into the filters without affecting invertibility. In particular, if the prediction operation at node n is normalized by a factor $c_{n,p}$, this is equivalent to multiplying the prediction matrix $\mathbf{I}-\mathbf{P}$ by a diagonal matrix $\mathbf{D}_p = \text{diag}(c_{1,p}, c_{2,p}, \ldots, c_{N,p})$. The same is true for the update matrix $\mathbf{I} + \mathbf{U}$ for some diagonal matrix $\mathbf{D}_u$. As long as the normalization factors are non-zero (there is no practical reason why they should be zero), the overall transform $\mathbf{y}' = \mathbf{D}_u \cdot (\mathbf{I} + \mathbf{U}) \cdot \mathbf{D}_p \cdot (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$, will be trivially invertible.

Corollary 1 (Lifting Filter Normalization)**.** *Let* $\mathcal{E}$*,* $\mathcal{O}$*,* $\mathbf{P}$ *and* $\mathbf{U}$ *be specified as in Definition 1 and Proposition 1. Let* $c_{1,p}, c_{2,p}, \ldots, c_{N,p}$ *(resp.* $c_{1,u}, c_{2,u}, \ldots, c_{N,u}$*) be a set of normalization factors for the prediction (resp. update) filters, with* $\mathbf{D}_p = diag(c_{1,p}, c_{2,p}, \ldots, c_{N,p})$ *(resp.* $\mathbf{D}_u = diag(c_{1,u}, c_{2,u}, \ldots, c_{N,u})$*). The normalized prediction and update filters are then given by the matrices* $\mathbf{P}' = \mathbf{D}_p \cdot (\mathbf{I} - \mathbf{P})$ *and* $\mathbf{U}' = \mathbf{D}_u \cdot (\mathbf{I} + \mathbf{U})$ *respectively. Moreover,* $(\mathbf{P}')^{-1} = (\mathbf{I} + \mathbf{P}) \cdot \mathbf{D}_p^{-1}$ *and* $(\mathbf{U}')^{-1} = (\mathbf{I} - \mathbf{U}) \cdot \mathbf{D}_u^{-1}$*.*

This can be easily generalized to multiple lifting (i.e., prediction and update) steps and multiple levels of decomposition. Let $\mathcal{O}_j$ and $\mathcal{E}_j$ be odd and even sets of nodes, respectively, for $j = 0, 1, 2, \ldots, J$ and some positive integer J. We assume that $\mathcal{E}_0 = \mathcal{I}$ and $\mathcal{O}_0 = \emptyset$. Suppose that $\mathcal{E}_{j-1} = \mathcal{E}_j \cup \mathcal{O}_j$ and $\mathcal{E}_j \cap \mathcal{O}_j = \emptyset$ for all j. This provides a direct analogy to the standard dyadic decomposition. At each level j, let K_j denote the number of lifting steps and let the k-th prediction and update filters be respectively denoted by the vectors $\mathbf{p}_{n,j}^k$ and $\mathbf{u}_{m,j}^k$ for all $m, n \in \mathcal{I}$. Of course each of these should satisfy Definition 1. Moreover, let $\mathbf{P}_{j,k}$ and $\mathbf{U}_{j,k}$ satisfy the assumptions in Proposition 1 for each j, let $d_j(n)$ denote the detail coefficient of each $n \in \mathcal{O}_j$ and let $s_j(m)$ denote the smooth coefficient of each $m \in \mathcal{E}_j$. By convention, we let $\mathbf{P}_0 = \mathbf{U}_0 = \mathbf{0}$ and $s_0(n) = x(n)$ for all n. This represents a multi-level transform decomposition on the original data, with the aggregate transform operations defined as $\mathbf{y} = \prod_{j=1}^{J} \prod_{k=1}^{K_j} (\mathbf{I} + \mathbf{U}_{j,k}) \cdot (\mathbf{I} - \mathbf{P}_{j,k}) \cdot \mathbf{x}$. Each $(\mathbf{I} - \mathbf{P}_{j,k})$ and $(\mathbf{I} + \mathbf{U}_{j,k})$ is invertible by Proposition 2. Therefore, the overall transform is invertible. This is formally stated in Corollary 2. Note that this transform is still invertible when filter normalization is introduced. This follows by a simple extension of Corollary 1.

Corollary 2 (Invertible Multi-level Lifting Transforms)**.** *Let $\mathcal{E}_j$ and $\mathcal{O}_j$ satisfy Definition 1 and $\mathbf{P}_{j,k}$ and $\mathbf{U}_{j,k}$ satisfy the assumptions in Proposition 1 for all $j = 0, 1, 2, \ldots, J$ for some positive integer J, and for all $k = 1, 2, \ldots, K_j$. Suppose that $\mathcal{E}_{j-1} = \mathcal{E}_j \cup \mathcal{O}_j$ and $\mathcal{E}_j \cap \mathcal{O}_j = \emptyset$ for all j. Then the transform $\mathbf{y} = \prod_{j=1}^{J} \prod_{k=1}^{K_j} (\mathbf{I} + \mathbf{U}_{j,k}) \cdot (\mathbf{I} - \mathbf{P}_{j,k}) \cdot \mathbf{x}$ is invertible with $(\mathbf{I} - \mathbf{P}_{j,k})^{-1} = \mathbf{I} + \mathbf{P}_{j,k}$ and $(\mathbf{I} + \mathbf{U}_{j,k})^{-1} = \mathbf{I} - \mathbf{U}_{j,k}$ for all j and k.*

2.2 Even/Odd Split Design

Assume that nodes are organized on some graph G. This graph could naturally arise from a routing tree as in a WSN, or could be defined based on some additional information as we shall see in the case of images.. We can now investigate exactly how nodes should be split into even and odd sets. An even/odd splitting strategy on trees is described in Section 2.2.1 and a set of strategies for graphs are described in Section 2.2.2. Both the tree-based and graph-based splitting methods are used for the WSN application in Chapter 3 and an experimental evaluation is provided in Section 3.6.3.

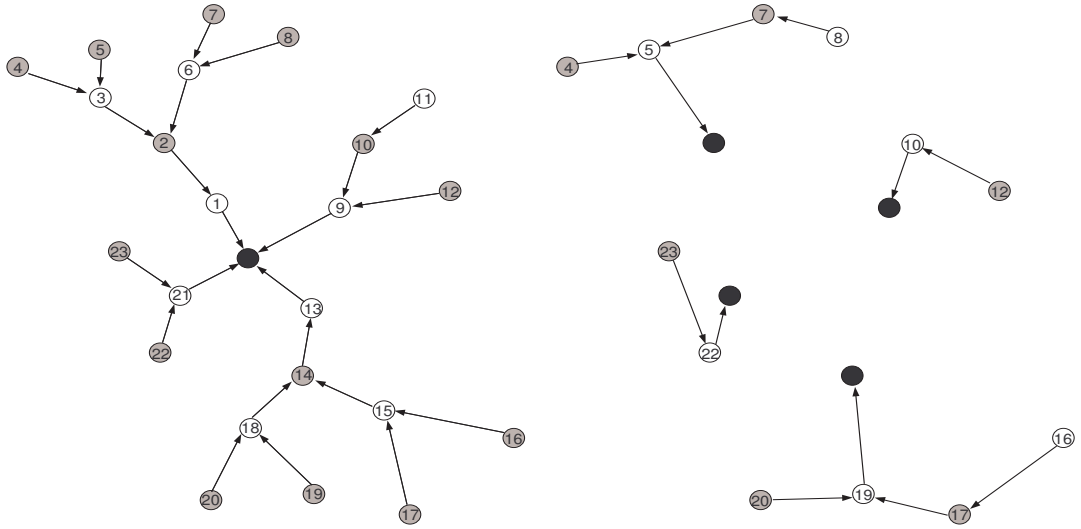
2.2.1 Tree-based Even/Odd Split

Let T denote a rooted tree with root node indexed by $N + 1$. This provides some notion of relative position in T. In particular, every node n will have a parent $\rho(n)$, children $\mathcal{C}_n$, descendants $\mathcal{D}_n$, ancestors $\mathcal{A}_n$, and will be $h(n)$ hops away from the root node. $h(n)$ can also be thought of as the depth of n in T. We can use this information to define the splitting in a manner analogous to the even/odd splitting done on 1D data, e.g., where in $\mathbb{Z}$, samples occurring at even integers are even and those occurring at odd integers are odd.

One natural way of doing an even/odd splitting along T (analogous to even/odd splitting in 1D) is to use the parity of the depth of each node. This splitting method was introduced by us in [55], where each node n for which $h(n)$ is odd fall into the odd set $\mathcal{O}$, i.e., $\mathcal{O} = \{n : h(n) \mod 2 = 1\}$. Similarly, each node m such that $h(m)$ is even fall into the even set $\mathcal{E}$, i.e., $\mathcal{E} = \{m : h(m) \mod 2 = 0\}$. An example of this split design is shown in Figure 2.1(a). Clearly $\mathcal{O} \cap \mathcal{E} = \emptyset$, and so any prediction and update operations satisfying Definition 1 for this choice of $\mathcal{E}$ and $\mathcal{O}$ will yield an invertible transform. In general, multiple trees T_j can be defined for some $j = 1, 2, \ldots, J$ and positive integer J, with corresponding even sets $\mathcal{E}_j$ and odd sets $\mathcal{O}_j$. Lifting transforms can then be defined on each T_j and, by Corollary 2, each of these transforms will be invertible. Therefore, any multi-level transform constructed in this way will also be invertible. An example of splitting over multiple trees is shown in Figure 2.1. This split design is adopted later in Chapter 3 for the WSN application.

2.2.2 Graph-based Even/Odd Split

While even/odd splitting on a tree is rather simple, it has its disadvantages since it will not exploit links that exists between nodes that are not directly connected in the tree. For example, in the WSN application a routing tree is typically given, but due to the broadcast nature of wireless communication [10, 77], multiple nodes will be able to overhear a single data transmission. This induces additional communication links on top of those along the routing tree, thus, a graph arises. Since it may be possible to achieve more de-correlation by doing an even/odd splitting on this graph (since nodes will typically have more neighbors on a graph than along a tree), it would generally be better to do a graph-based splitting whenever possible.



(a) Split on 1^{st} tree (b) Split on 2^{nd} trees

Figure 2.1: Examples of splitting on multiple trees. Black center node is the sink, gray nodes are even and white nodes are odd. The first level tree is shown in (a). In the second level tree (b), the even nodes from the first level are again split and another level of transform decomposition is performed.

Moreover, there are other applications [23,36] where only the connectivity between nodes is given (e.g., no tree is provided); clearly a graph-based even/odd splitting is needed in these cases. Thus, we also summarize results on graph-based splitting methods for lifting transforms.

Various graph-based even/odd splitting methods have been proposed in the literature [3, 23, 36, 72, 73]. The techniques in [3, 72, 73] are used for distributed compression in WSNs, where nodes are assumed to be randomly distributed on some subset of $\mathbb{R}^2$. In this case, roughly speaking, the nodes with the largest number of neighbors within a certain distance R are chosen as odd and the rest are chosen as even. This is done over multiple splitting stages $j = 1, 2, \ldots, J$ until nodes can no longer be split, and it induces a series of graphs G_j, each of which is used to determine the j-th level splitting. This is one example of splitting nodes along a graph G. Note that under this even/odd split design, each odd node will

have many even neighbors. Thus, this even/odd split is particularly useful since a very accurate prediction $\hat{x}(n)$ can be generated for each odd node n. Therefore, $d(n) = x(n) - \hat{x}(n)$ will tend to be small on average. The graph-based split design in [23] uses similar ideas (i.e., each odd node should have many even neighbors), though it was developed and used specifically for the de-noising of irregularly spaced data.

Alternatively, the even/odd split in [36] attempts to find an even/odd splitting in which the number of links between different even (resp. odd) nodes is minimum. The motivation in that work is to utilize as many links in the graph as possible when computing a lifting transform. Since even (resp. odd) node data is not processed using other even (resp. odd) node data, the links between different even (resp. odd) nodes will not be used in the transforms. Thus, the goal in that work is to find a split that minimizes the number of links between different even (resp. odd) nodes. This is done by searching for a bi-partite graph (which partitions nodes into disjoint even and odd subsets) that minimizes the "conflict fraction", i.e., the number of links between even nodes plus the number of links between odd nodes divided by the total number of links in the graph. Since odd nodes will have (on average) more even neighbors in graph-based splits than in tree-based splits, the predictions are likely to be more accurate. Thus, there will be less energy in the detail coefficients; this reduction in energy will generally lead to more efficient signal representations (e.g., better energy compaction and coding performance).

The graph-based even/odd split we proposed in [37] attempts to optimize the total energy consumption in a WSN. This is done by minimizing the number of even nodes under the constraint that every odd node has at least one even neighbor. This will tend to produce splits for which odd nodes have relatively few even neighbors, so it will not be good from a data de-correlation standpoint. However, in the context of

WSN where the goal is to minimize the total energy consumption, using distributed lifting transforms with very few even nodes turns out to be very beneficial. To summarize [37], the main observations are that (i) any form of transform-based distributed data gathering requires some nodes to transmit raw data (i.e., there must be some *raw data nodes*), (ii) nodes that receive raw data can perform some aggregation to remove correlation as to reduce the amount of data they need to transmit (i.e., there are some *aggregating nodes*), and (iii) the cost to transmit raw data is typically much higher than the cost to transmit aggregated data. Thus, the main goal in that work is to minimize the number of raw data nodes (or rather, to minimize the total cost incurred by raw data nodes) under some constraints. This is a rather general framework (i.e., assignment of raw and aggregating nodes), and lifting transforms are just one example where even nodes serve as raw nodes and odd nodes (which compute residual detail coefficients) act as aggregating nodes. Some comparisons of tree-based and graph-based splits will be presented in Section 3.6.3.

In summary, the graph-based even/odd splitting techniques proposed in [3, 23, 36, 72, 73] are very general and can be applied to any graph. Thus, these graph-based even/odd split designs (along with the tree-based split designs) could also be applied to other applications such as image coding. On the other hand, the graph-based split proposed in [37] was designed specifically for distributed compression in WSNs, and may not provide good performance for other applications such as image coding.

2.3 Prediction Filter Design

Assume (as in Section 2.2) that a graph $G = (V, E)$ is given, nodes are placed in $\mathbb{Z}^2$ and the position of each node n is given by (i_n, j_n). As discussed before, data

de-correlation occurs in the prediction step, i.e., for each odd node n a prediction $\hat{x}(n) = \sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m)x(m)$ and detail coefficient $d(n) = x(n) - \hat{x}(n)$ is computed and encoded. If $\hat{x}(n) \approx x(n)$, then $d(n) \approx 0$, so that it can be encoded using significantly fewer bits than would be needed to encode $x(n)$. Thus, the goal in this section is to design a prediction filter $\mathbf{p}_n$ that produces very accurate predictions of $x(n)$ using data from an arbitrary set of neighbors $\mathcal{N}_n$, i.e., we want to design $\mathbf{p}_n$ such that $\hat{x}(n) = \sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m)x(m) \approx x(n)$. The proper choice of $\mathbf{p}_n$ ultimately depends on how data is correlated across nodes.

In this section we present two methods for computing "good" prediction filters (e.g., "good" in the sense that $|d(n)|^2$ is minimized). In the first method, we assume that data is locally very smooth, so that the data across neighboring nodes is well approximated by a polynomial function. More specifically, for each odd node n, the data from its neighbors in $\mathcal{N}_n$ can be well approximated by a K-degree polynomial $P(i, j|\mathcal{N}_n)$, in which case we can accurately predict $x(n)$ by $\hat{x}(n) = P(i_n, j_n|\mathcal{N}_n)$, i.e., $d(n) = x(n) - \hat{x}(n) \approx 0$. This will be discussed in Section 2.3.1.

If the data is not piece-wise polynomial but is spatially stationary, with the correlation between any node n and m following correlation function $R_{XX}(n, m)$, we can still compute good prediction filters. Thus, in the second method we describe how to compute prediction filter $\mathbf{p}_n$ that minimizes the mean squared error of detail coefficient $d(n)$. They are simply the well known linear minimum mean squared error (LMMSE) prediction filters [20], and are optimal in the sense that they minimize $\mathbb{E}[|d(n)|^2]$, where $\mathbb{E}[\cdot]$ denotes the expected value operation. If the data is stationary but the correlation function $R_{XX}(n, m)$ is unknown, we can use an adaptive filter [20] to estimate the optimal filters. This will be discussed in Section 2.3.2 in the context of distributed compression for WSNs and was initially introduced by us in [52].

2.3.1 Polynomial Prediction Filters

If the data is nearly piece-wise polynomial, then for each odd node n, we can fit data from even neighbors $\mathcal{N}_n$ to a 2D polynomial $P(i,j|\mathcal{N}_n)$, and can predict $x(n)$ by $\hat{x}(n) = P(i_n, j_n|\mathcal{N}_n)$ with great accuracy, i.e., $d(n) = x(n) - \hat{x}(n) \approx 0$. As an example of this type of design, consider a planar prediction of $x(n)$ from $\{x(m)\}_{\mathcal{N}_n}$ as was proposed in [3,72,73]. Suppose that $\mathcal{N}_n = \{m_1, m_2, \ldots, m_{|\mathcal{N}_n|}\}$. The best-fit plane (in the least squares sense [63]) $P(i,j|\mathcal{N}_n) = A \cdot i + B \cdot j + C$ of the data $\{x(m_i)\}_{m_i \in \mathcal{N}_n}$ can be found by solving

$$\begin{bmatrix} x(m_1) \\ \vdots \\ x(m_{k(n)}) \end{bmatrix} = \begin{bmatrix} i_{m_1} & j_{m_1} & 1 \\ \vdots & & \\ i_{m_{|\mathcal{N}_n|}} & j_{m_{|\mathcal{N}_n|}} & 1 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \end{bmatrix}. \tag{2.3}$$

Let $\mathbf{x}_{\mathcal{N}_n} = \left[x(m_1), x(m_2), \ldots, x(m_{|\mathcal{N}_n|})\right]^t$, $\mathbf{c} = (A, B, C)^t$ and

$$\mathbf{A}_n = \begin{bmatrix} i_{m_1} & j_{m_1} & 1 \\ \vdots & & \\ i_{m_{|\mathcal{N}_n|}} & j_{m_{|\mathcal{N}_n|}} & 1 \end{bmatrix}.$$

Then a best-fit plane exists as long as rank $(\mathbf{A}_n^t \mathbf{A}_n) = 3$, since then $\mathbf{c} = (\mathbf{A}_n^t \mathbf{A}_n)^{-1} \mathbf{A}_n^t \mathbf{x}(\mathcal{N}_n)$. Note that rank $(\mathbf{A}_n^t \mathbf{A}_n)$ = rank $(\mathbf{A}_n)$ [63], so a best fit plane exists if and only if rank $(\mathbf{A}_n) = 3$. Since $\hat{x}(n) = A \cdot i_n + B \cdot j_n + C = [i_n \;\; j_n \;\; 1] \cdot \mathbf{c}$, $\hat{x}(n) = \left([i_n \;\; j_n \;\; 1] \cdot (\mathbf{A}_n^t \mathbf{A}_n)^{-1} \cdot \mathbf{A}_n^t\right) \cdot \mathbf{x}_{\mathcal{N}_n}$. Thus, $\mathbf{p}_n(\mathcal{N}_n) = \left([i_n \;\; j_n \;\; 1] \cdot (\mathbf{A}_n^t \mathbf{A}_n)^{-1} \cdot \mathbf{A}_n^t\right)^t$, and is only a function of the positions of n and $\mathcal{N}_n$.

For a general K-degree polynomial we have $P(i,j|\mathcal{N}_n) = \sum_{k=0}^{K} \sum_{l=0}^{K} C_{k,l} \cdot i^k \cdot j^l$. In this case, we must estimate $(K+1)^2$ of the $C_{k,l}$ parameters, i.e., to approximate $x(n)$ with a K-degree polynomial fitted to $\{x(m_i)\}_{m_i \in \mathcal{N}_n}$, n must have at least

$(K+1)^2$ neighbors. For many practical applications, sets of neighbors whose size scales as $(K+1)^2$ may not be feasible for large K, so using high degree polynomials to compute predictions is not very practical.

2.3.2 Data-adaptive Prediction Filters

If the data is spatially stationary with correlation between samples at node n and m given by $R_{XX}(n,m)$, we can still compute good prediction filters. In this case, for each odd node n, we want to find a prediction vector for $x(n)$, that minimizes the energy in residual prediction error $d(n) = x(n) - \sum_{i\in\mathcal{N}_n} \mathbf{p}_n(i)x(i)$, i.e., we want to find

$$\mathbf{p}_n^* = \arg\min_{\mathbf{p}_n} \mathbb{E}[|x(n) - \sum_{i\in\mathcal{N}_n} \mathbf{p}_n(i)x(i)|^2]. \tag{2.4}$$

The solution is the well-known Wiener-Hopf solution [20], and is a function of the correlation $R_{XX}(i,j) = \mathbb{E}[x(i)x(j)]$ between nodes $i,j \in \mathcal{N}_n$.

2.3.2.1 Optimal Prediction Filters

We derive the optimal solution to (2.4) for the sake of completeness. Note that $x^*(n) = \sum_{i\in\mathcal{N}_n} \mathbf{p}_n^*(i)x(i)$ is the LMMSE estimate of $x(n)$ only if the orthogonality principle [62] is satisfied. Thus,

$$\mathbb{E}\{[x(n) - x^*(n)]x(j)\} = \mathbb{E}\{[x(n) - \sum_{i\in\mathcal{N}_n} \mathbf{p}_n^*(i)x(i)]x(j)\} = 0, \text{ for all } j \in \mathcal{N}_n \tag{2.5}$$

This implies that, for all $j \in \mathcal{N}_n$, $\mathbb{E}[x(n)x(j)] = \sum_{i\in\mathcal{N}_n} \mathbf{p}_n^*(i)\mathbb{E}[x(i)x(j)]$. Since $R_X(i,j) = \mathbb{E}[x(i)x(j)]$, we can simplify (2.5) as

$$\sum_{i\in\mathcal{N}_n} \mathbf{p}_n^*(i)R_X(i,j) = R_X(n,j), \text{ for all } j \in \mathcal{N}_n. \tag{2.6}$$

Let $\mathcal{N}_n = \{i_1, i_2, \ldots, i_{|\mathcal{N}_n|}\}$. Let the $|\mathcal{N}_n| \times |\mathcal{N}_n|$ matrix $\mathbf{R}_n$ be defined by $\mathbf{R}_n(k,l) = R_X(i_k, i_l)$ and let the $|\mathcal{N}_n| \times 1$ vector $\mathbf{r}_n$ be defined by $\mathbf{r}_n(k) = R_X(n, i_k)$. We can now express (2.6) above as $\mathbf{R}_n \mathbf{p}_n^*(\mathcal{N}_n) = \mathbf{r}_n$. So as long as $\mathbf{R}_n$ is invertible, we have an optimal solution for node n. If $\mathbf{R}_n$ is positive definite, then $\mathbf{p}_n^*(\mathcal{N}_n) = \mathbf{R}_n^{-1}\mathbf{r}_n$.

2.3.2.2 Approximating Optimal Prediction Filters

We now describe an algorithm to estimate the optimal prediction filters in the context of WSN as was introduced by us in [52]. Note that estimating these statistics in a WSN will be costly in terms of delay, computation and communication. Moreover, a large amount of data is generally needed to reliably estimate correlation matrices. Alternatively, we can use adaptive filters to estimate the optimal spatial prediction filters over time with no learning cost since (i) they converge to the optimal filters for stationary data, (ii) they do not require estimates of data statistics and (iii) the filtering done at one node can be replicated at any other node (e.g., the sink) given the same prediction errors and initial prediction filters. Note that if quantization is used, then both nodes must use the same quantized prediction errors to update the filters. More specifically, in order for the sink to re-produce the same prediction filters used at an odd node n, it must use (i) the same initial prediction filter as node n, (ii) the same prediction errors that were generated by node n, and (iii) the same data that node n used to compute the prediction errors. Conditions (i) and (ii) are easily met. In the context of lifting, (iii) is also met since the update step is always inverted before the prediction step is inverted; thus, the sink can always recover data that node n used to compute prediction errors. In this way, we can *apply an adaptive filter at each odd node* to estimate the optimal prediction filters *without specifying any additional information to the sink*. Note that it still takes time for the filters to adapt to the data well enough to produce

good predictions. Thus, there will be a small learning cost for nodes to initially "train" their filters and also to "re-train" their filters when data statistics change (i.e., the overall encoding rate will be higher during training periods, during which filters have not yet converged to a state that matches current data statistics.)

There are many adaptive filters that we can choose from, but the step-size parameter μ often must be chosen based on data dependent parameters to ensure filter convergence. We generally will not know those parameters, thus, the most suitable choice is a normalized least mean squares adaptive filter [20] since μ need not be specified but is instead adapted as the filter is adapted. Some notation is now established. Suppose nodes measure data at times $t_1, t_2, \ldots, t_M$. Let $x(n, m)$ denote the data at node n captured at time step t_m. The $N \times M$ prediction coefficient matrix for node n is given by $\mathbf{p}_n$, where column i, i.e., $\mathbf{p}_n(:, i)$, is the prediction vector at the i-th time step at node n. The *adaptive filter at each odd node* n is then computed, from $m = 1$ to $m = M$, as $d(n, m) = x(n, m) - \mathbf{p}_n^t(\mathcal{N}_n, m)x(\mathcal{N}_n, m)$ and the update equation $\mathbf{p}_n(\mathcal{N}_n, m+1) = \mathbf{p}_n(\mathcal{N}_n, m) + \tilde{\mu}\frac{x(\mathcal{N}_n,m)d(n,m)}{x^t(\mathcal{N}_n,m)x(\mathcal{N}_n,m)}$, where $\tilde{\mu}$ is a parameter that can be used to speed up (or slow down) the rate of convergence. For correlated Gaussian data, the optimal value of $\tilde{\mu} = 1$ [20].

2.4 Update Filter Design

It is also necessary to provide some form of update step after prediction in order to reduce the effects of numerical instability and propagation of quantization errors. We now present two update filter designs. The first one, as presented in Section 2.4.1, was proposed in [72, 73]. It essentially preserves the average value of smooth coefficients across multiple levels of decomposition. This reduces the harmful effects caused by numerical instability [23], but the resulting low-pass (LP)

filters may not be orthogonal to the resulting high-pass (HP) filters. Therefore, any quantization error introduced in the HP and LP subbands will propagate through the inverse transform into both subbands. Instead, it would be better to design update filters that force LP filters to be orthogonal to HP filters. We provide such an update filter design in Section 2.4.2 and show that it always exists. This design was initially proposed by the author in [56]. Moreover, we show that this choice of update filter (for a fixed prediction filter design) minimizes the reconstruction mean squared error due to quantization of transform coefficients. Comparisons of these various update filter designs are given in Section 3.6.3 and in Section 5.3.2. As we will see in Section 5.3.2 the orthogonalizing update filter design provides a modest increase in coding efficiency.

2.4.1 Mean-preserving Update Filters

A method to preserve the average value of coefficients across multiple levels of decomposition on a finite number of irregularly spaced data points was proposed in [72, 73]. For every even node $m \in \mathcal{E}$ with prediction filters $\mathbf{p}_n$ from neighbors $n \in \mathcal{N}_m$, the update filter which preserves the average value of the smooth coefficients is computed as a function of each $\mathbf{p}_n$. While this does provide some smoothing properties, the proposed design does not yield LP and HP filters that are mutually orthogonal. As pointed out in [16], this is problematic when quantization is introduced since it will cause the quantization errors from one subband to propagate into the reconstructed samples from other subbands. This will increase the total quantization error in the reconstructed data. Therefore, it is more desirable to design a lifting transform that forces the low-pass component to be orthogonal to the high-pass component.

2.4.2 Orthogonalizing Update Filters

We now describe the orthogonalizing update filter design as was first introduced by the author in [56]. As has been discussed, it is desirable to design update filters that makes the LP signal component orthogonal to the HP signal component after each lifting step. More specifically, we would like to decompose the signal vector as $\mathbf{x} = \mathbf{x}_e + \mathbf{x}_o$, with $< \mathbf{x}_e, \mathbf{x}_o >= 0$. This should increase the overall energy compaction of the transform. It will also be useful when performing quantization since it essentially isolates quantization noise into each sub-band. Note that after each lifting step, the even samples correspond to the "low-pass" component and the odd samples to the "high-pass" component. Furthermore, since $\mathbf{I} + \mathbf{U}$ and $\mathbf{I} - \mathbf{P}$ are invertible, $\mathbf{T} = (\mathbf{I} + \mathbf{U})(\mathbf{I} - \mathbf{P})$ is also invertible. Let $\text{row}_i(\mathbf{T}) = \mathbf{t}_i^t$ and let $\mathbf{T}^{-1} = \begin{bmatrix}\tilde{\mathbf{t}}_1 & \tilde{\mathbf{t}}_2 & \dots & \tilde{\mathbf{t}}_N\end{bmatrix}$. Since $\{\mathbf{t}_i\}_{i\in\mathcal{I}}$ and $\{\tilde{\mathbf{t}}_i\}_{i\in\mathcal{I}}$ form a pair of dual bases, we can represent our signal as $\mathbf{x} = \sum_{i=1}^{N} < \mathbf{t}_i, \mathbf{x} > \tilde{\mathbf{t}}_i$. An orthogonal decomposition will then be obtained if we can construct lifting filters that force $< \tilde{\mathbf{t}}_i, \tilde{\mathbf{t}}_j >= 0$ for any $i \in \mathcal{E}$ and $j \in \mathcal{O}$, since then we have $\mathbf{x} = \mathbf{x}_e + \mathbf{x}_o$ with $\mathbf{x}_e = \sum_{i\in\mathcal{E}} < \mathbf{t}_i, \mathbf{x} > \tilde{\mathbf{t}}_i$, $\mathbf{x}_o = \sum_{j\in\mathcal{O}} < \mathbf{t}_j, \mathbf{x} > \tilde{\mathbf{t}}_j$ and $< \mathbf{x}_e, \mathbf{x}_o >= 0$.

We would like to design update filters that provide the desired orthogonality of the dual basis vectors $\{\tilde{\mathbf{t}}_i\}_{i\in\mathcal{I}}$. To achieve this, we assume a fixed prediction filter design, then construct update filters such that the "equivalent filter" of any even node ($\mathbf{t}_n$ for $n \in \mathcal{E}$) is orthogonal to the "equivalent filter" of every odd node ($\mathbf{t}_m$ for $m \in \mathcal{O}$). Suppose there are N_e and N_o even and odd nodes, respectively, and let $\mathcal{E} = \{j_1, j_2, \dots, j_{N_e}\}$ and $\mathcal{O} = \{i_1, i_2, \dots, i_{N_o}\}$ be the sets of even and odd indices, respectively. The equivalent filter for every even node n is $\mathbf{t}_n = \mathbf{e}_n + \sum_{k=1}^{N_o} \mathbf{u}_n(i_k)d(i_k)$ and the equivalent filter for every odd node m is $\mathbf{t}_m = \mathbf{e}_m - \mathbf{p}_m$. What we are seeking is an "orthogonalizing" update filter design for which

$< \mathbf{t}_n, \mathbf{t}_m >= 0$ for all $n \in \mathcal{E}$ and $m \in \mathcal{O}$. The orthogonalizing update filter design is presented in Proposition 3. This design is also sufficient to provide the desired orthogonality result as stated in Proposition 4 (the proof is in Appendix A), that is, our proposed lifting filter design ensures that $< \mathbf{t}_n, \mathbf{t}_m >= 0$ for any $n \in \mathcal{E}$ and $m \in \mathcal{O}$, and this implies that $< \tilde{\mathbf{t}}_n, \tilde{\mathbf{t}}_m >= 0$ for any $n \in \mathcal{E}$ and $m \in \mathcal{O}$.

Proposition 3 (Orthogonalizing Update Filters)**.** *Let the prediction filter for every odd node be fixed and let $\mathcal{N}_n = \mathcal{O}$ for all $n \in \mathcal{E}$. Let $\mathbf{t}_n$ be the* equivalent filter *of node n, i.e., it is the filter resulting from application of both the prediction and update step. Then the equivalent filter of an even node n (e.g., every LP filter) is orthogonal to the equivalent filter of every odd node i_k (e.g., every HP filter), i.e.,*

$$(\mathbf{e}_{i_k} - \mathbf{p}_{i_k})^t \mathbf{t}_n = 0, \ \forall k = 1, 2, \ldots, N_o \tag{2.7}$$

if and only if

$$\mathbf{u}_n(\mathcal{O}) = \left(\mathbf{I}_{N_o} + \tilde{\mathbf{P}}^t \tilde{\mathbf{P}}\right)^{-1} \tilde{\mathbf{P}}^t \mathbf{e}_n. \tag{2.8}$$

Since $\left(\mathbf{I}_{N_o} + \tilde{\mathbf{P}}^t \tilde{\mathbf{P}}\right)^{-1}$ always exists, we always have update filters for which $\mathbf{t}_n^t \mathbf{t}_m = 0, \ \forall m \in \mathcal{O}, n \in \mathcal{E}$.

Proof. Since $x(n) = \mathbf{e}_n^t \mathbf{x}$ and $d(i_k) = (\mathbf{e}_{i_k} - \mathbf{p}_{i_k})^t \mathbf{x}$, we have that

$$\begin{aligned} s(n) &= x(n) + \sum_{k=1}^{N_o} \mathbf{u}_n(i_k) d(i_k) \\ &= \mathbf{e}_n^t \cdot \mathbf{x} + \sum_{k=1}^{N_o} \mathbf{u}_n(i_k) \left(\mathbf{e}_{i_k} - \mathbf{p}_{i_k}\right)^t \cdot \mathbf{x} \\ &= \left(\mathbf{e}_n + \sum_{k=1}^{N_o} \mathbf{u}_n(i_k) \left(\mathbf{e}_{i_k} - \mathbf{p}_{i_k}\right)\right)^t \cdot \mathbf{x} \\ &= \mathbf{t}_n^t \mathbf{x} \end{aligned}$$

(2.7) is satisfied if and only if

$$(\mathbf{e}_{i_l} - \mathbf{p}_{i_l})^t \left(\mathbf{e}_n + \sum_{k=1}^{N_o} \mathbf{u}_n(i_k) \left(\mathbf{e}_{i_k} - \mathbf{p}_{i_k} \right) \right) = 0, \ \forall i_l \in \mathcal{O}. \tag{2.9}$$

Since $n \in \mathcal{E}$, $n \neq i_l$. Thus, $\mathbf{e}_{i_l}(n) = 0$, and so $\mathbf{e}_{i_l}^t \mathbf{e}_n = 0$. Since $\mathbf{p}_{i_k}(i_l) = 0$ for all k by Definition 1, $\mathbf{e}_{i_l}^t \mathbf{p}_{i_k} = 0$ for all k. Similarly, $\mathbf{p}_{i_l}(i_k) = 0$ for all k, so $\mathbf{p}_{i_l}^t \mathbf{e}_{i_k} = 0$ for all k. Therefore, (2.9) becomes

$$\mathbf{e}_{i_l}^t \sum_{k=1}^{N_o} \mathbf{e}_{i_k} \mathbf{u}_n(i_k) + \mathbf{p}_{i_l}^t \sum_{k=1}^{N_o} \mathbf{p}_{i_k} \mathbf{u}_n(i_k) = \mathbf{p}_{i_l}^t \mathbf{e}_n, \ \forall i_l \in \mathcal{O}. \tag{2.10}$$

If we define $\mathbf{u}_n(\mathcal{O}) = [\mathbf{u}_n(i_1), \ldots, \mathbf{u}_n(i_{N_o})]^t$ and $\tilde{\mathbf{I}} = \left[\mathbf{e}_{i_1} \ldots \mathbf{e}_{i_{N_o}} \right]$, we have that $\sum_{k=1}^{N_o} \mathbf{e}_{i_k} \mathbf{u}_n(i_k) = \tilde{\mathbf{I}} \cdot \mathbf{u}_n(\mathcal{O})$ and $\sum_{k=1}^{N_o} \mathbf{p}_{i_k} \mathbf{u}_n(i_k) = \tilde{\mathbf{P}} \cdot \mathbf{u}_n(\mathcal{O})$. Thus, (2.10) becomes

$$\mathbf{e}_{i_l}^t \cdot \tilde{\mathbf{I}} \cdot \mathbf{u}_n(\mathcal{O}) + \mathbf{p}_{i_l}^t \cdot \tilde{\mathbf{P}} \cdot \mathbf{u}_n(\mathcal{O}) = \mathbf{p}_{i_l}^t \cdot \mathbf{e}_n, \ \forall i_l \in \mathcal{O}. \tag{2.11}$$

This provides a set of N_o linear equations in N_o unknowns, and we can express (2.7) as

$$\left[\tilde{\mathbf{I}}^t \tilde{\mathbf{I}} + \tilde{\mathbf{P}}^t \tilde{\mathbf{P}} \right] \cdot \mathbf{u}_n(\mathcal{O}) = \tilde{\mathbf{P}}^t \cdot \mathbf{e}_n. \tag{2.12}$$

Note that $\tilde{\mathbf{I}}^t \tilde{\mathbf{I}} = \mathbf{I}_{N_o}$, the $N_o \times N_o$ identity matrix. Moreover, for any $\mathbf{x} \neq \mathbf{0}$, $\mathbf{x}^t \left[\tilde{\mathbf{I}}^t \tilde{\mathbf{I}} + \tilde{\mathbf{P}}^t \tilde{\mathbf{P}} \right] \mathbf{x} = ||\mathbf{x}||^2 + ||\tilde{\mathbf{P}}\mathbf{x}||^2 > 0$. Thus, $\left[\mathbf{I}_{N_o} + \tilde{\mathbf{P}}^t \tilde{\mathbf{P}} \right]$ is positive definite and (2.8) follows. Since $\mathbf{t}_m = \mathbf{e}_m - \mathbf{p}_m$ for all $m \in \mathcal{O}$, $\mathbf{t}_n^t \mathbf{t}_m = 0, \ \forall m \in \mathcal{O}, n \in \mathcal{E}$. □

Proposition 3 yields a filter design in which the equivalent filter of every even node is orthogonal to the filter of every odd node. Intuitively, this provides a decomposition of the signal into two separate subbands and one should expect orthogonality between the signal components in each subband. In fact, orthogonality between the filters of even and odd nodes is sufficient to guarantee an *orthogonal*

decomposition as $\mathbf{x} = \mathbf{x}_e + \mathbf{x}_o$ with $\mathbf{x}_e^t \mathbf{x}_o = 0$. This is formally stated in Proposition 4 and is proven in the Appendix. Moreover, under a fixed prediction design and some mild assumptions about quantization noise, the filter design proposed in [16] is equivalent to the design in Proposition 3, where in [16] it was shown that this design minimizes the mean-squared reconstruction error due to quantization of the transform coefficients. Thus, our proposed update filters are also useful in coding applications.

Proposition 4 (Orthogonal Decomposition)**.** *Let there be N nodes with $\mathcal{E}$, $\mathcal{O}$, $\mathbf{P}$ and $\mathbf{U}$ specified as in Definition 1 and Proposition 1. Let $\mathbf{T} = (\mathbf{I} + \mathbf{U})(\mathbf{I} - \mathbf{P}) = [\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_N]^t$ and $\mathbf{T}^{-1} = \left[\tilde{\mathbf{t}}_1 \ \tilde{\mathbf{t}}_2 \ \dots \ \tilde{\mathbf{t}}_N\right]$. Suppose the lifting filters have been designed as in Proposition 3. Then for any vector $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{x} = \mathbf{x}_e + \mathbf{x}_o$, with $\mathbf{x}_e = \sum_{i \in \mathcal{E}} < \mathbf{t}_i, \mathbf{x} > \tilde{\mathbf{t}}_i$, $\mathbf{x}_o = \sum_{j \in \mathcal{O}} < \mathbf{t}_j, \mathbf{x} > \tilde{\mathbf{t}}_j$, and $< \mathbf{x}_e, \mathbf{x}_o >= 0$.*

2.4.3 Discussion

Some remarks are now in order. Note that the work initially proposed by the author in [56] only provided the result in Proposition 3. In this thesis, we have also proven Proposition 4, which shows that the update design proposed in Proposition 3 (from [56]) provides an orthogonal decomposition of $\mathbf{x} = \mathbf{x}_e + \mathbf{x}_o$, with $< \mathbf{x}_e, \mathbf{x}_o >= 0$. This is useful from a coding perspective since the quantization errors of the even (LP) components are also orthogonal to the quantization errors of the odd (HP) components, i.e., it essentially isolates the quantization errors made in one subband from the other. This isolation of quantization errors should intuitively lead to minimum reconstruction error. This is in fact the case since we have arrived at the same solution as in [16] (which aims at minimizing the reconstruction error due to quantization), and the connection with the work in [16] shows why our proposed

update filters (and in particular, orthogonal decompositions) are useful in coding applications.

2.5 Conclusions

In this chapter we have shown that lifting transforms are invertible by construction (Proposition 2, Corollary 2), have proposed various methods for even/odd splitting, proposed optimized prediction filter designs, and have developed optimal update filter designs. The even/odd splitting methods described have been optimized with de-correlation in mind [3, 23, 36, 72, 73], and have also been optimized with the goal of minimizing total energy-consumption in WSN [37]. The proposed prediction filters are optimal in the sense that the average energy in prediction residuals is minimized [20, 52]. This is useful from a coding perspective since lower energy in the prediction residuals generally leads to fewer bits needed to represent them. Since these optimal filters depend on the correlation structure in the data, which is not always known, an adaptive prediction filter method was also developed [52]. These adaptive filters converge to the optimal filters under some stationarity assumptions [20]. Finally, orthogonalizing update filters were also proposed [56] (Proposition 3). It was shown that these filters provide an orthogonal decomposition of the input signal (Proposition 4), and moreover, this choice of filters also minimizes the reconstruction MSE. These designs and optimizations are used throughout the remainder of this thesis, with the goal of (i) minimizing total energy consumption in WSN, and (ii) providing efficient image representations for image coding.

Chapter 3

Transform-based Distributed Data Gathering

We now describe how to apply these tree-based and graph-based lifting transforms to WSNs. Most of the work described in this chapter was proposed by us in [37, 52, 55, 57, 58]. We focus on the data gathering problem where the goal is to collect data from every node in a WSN at a central collection (or sink) node. In particular, we assume that the nodes in the WSN are organized onto a routing tree. The application is introduced in detail in Section 3.1. We then develop a general framework for computing "unidirectional" transforms along routing trees (i.e., transforms that are computed as data is routed toward the sink node along the tree) and provide a set of conditions under which these transforms are invertible. This framework was initially introduced by us in [58], and was fully formalized in [57]. It is described in detail in Section 3.2. Once the basic framework is established, we then show how existing transforms fit into this framework. This was also described in [57], and is described in this thesis in Section 3.3. Finally, we discuss how to design unidirectional tree-based and graph-based lifting transforms, then compare the performance against existing work. Again, this work was initially proposed by the author in [57], and is described in detail in Section 3.3.4, 3.3.5 and 3.4.

3.1 Introduction

In networks such as wireless sensor networks (WSNs), one major challenge is to gather data from a set of nodes and transfer it to a collection (or sink) node as efficiently as possible. Efficiency can be measured in terms of bandwidth utilization, energy consumption, etc. We refer to this as the *data gathering problem*. The gathering is typically done in data gathering rounds or *epochs* along a collection of routing paths to the sink, i.e., in every epoch each node forwards data that it has measured along a multi-hop path to the sink. A simple gathering strategy is to have each node route raw data to the sink in a way that minimizes some cost metric, e.g., number of hops to the sink, energy consumption. This minimizes the amount of resources nodes use to transfer raw data to the sink and is the basis for many practical systems used in WSN such as the Collection Tree Protocol (CTP) [68]. However, it has been recognized in the literature [2, 4] that, in a WSN, (i) spatial data correlation may exist across neighboring nodes and (ii) nodes that are not adjacent to each other in a routing path can still communicate due to broadcasted wireless transmissions[1]. Raw data forwarding does not make use of these two facts, thus, it will not be the most efficient data gathering method in general.

When spatial data correlation exists, it may be useful to apply *in-network compression* distributed across the nodes to reduce this data redundancy [4]. More specifically, nodes can exchange data with their neighbors in order to remove spatial data correlation. This will lead to *a representation requiring fewer bits per measurement as compared to a raw data representation*, also leading to reduced energy consumption, bandwidth usage, delay, etc. Since nodes in a WSN are severely energy-constrained [2, 4, 74], some form of in-network processing that removes data

[1]Data transmissions in a WSN are typically broadcast [10, 77], so multiple nodes can receive a single data transmission.

redundancy will help reduce the amount of energy nodes consume in transmitting data to the sink. In this way the lifetime of a WSN can be extended. This could also be useful in other bandwidth-limited applications such as underwater acoustic networks [46] and structural health monitoring [34].

Generally speaking, distributed spatial compression schemes require some form of data exchange between nodes. Therefore, one needs to select both *a routing strategy* and *a processing strategy*. The routing strategy defines what data communications nodes need to make and the processing strategy defines how each node processes data. There are a variety of approaches available, e.g., distributed source coding (DSC) techniques [13, 45], transform-based methods like Distributed KLT [15], Ken [5], PAQ [69], and wavelet-based approaches [1, 3, 6, 7, 9, 55, 72, 73]. Note that DSC techniques do not require nodes to exchange data in order to achieve compression. Instead, each node can compress its own data using some statistical correlation model. Note, however, that an estimate of these models must be known at every node, so nodes will still need to do some initial data exchange in order to learn the models (after which compression can be done independently at each node). Our work only considers transform-based methods, which use linear transforms to decorrelate data while distributing transform computations across different nodes. While we do not consider DSC approaches, our algorithms could be useful in the training phase of these methods to estimate correlation. Ken and PAQ are examples of approaches we consider, where data at each node is predicted using a linear combination of measurements from the node and measurements received from its neighbors. Similarly, the Distributed KLT, wavelet-based methods and many other related methods also use linear transforms to decorrelate data. Therefore, we can restrict ourselves to linear in-network transforms while still encompassing a general class of techniques.

Many of the existing transform-based methods *propose a specific transform first*, then design routing and processing strategies that allow the transform to be computed in the network. Some examples are the wavelet transforms proposed in [3,9,72,73], the Distributed KLT, Ken and PAQ. While these methods are good from a data decorrelation standpoint, *the routing and processing strategies that are used to facilitate distributed processing may not always be efficient in terms of data transport cost*. In particular, nodes may have to transmit their own data multiple times [72,73], nodes may need to transmit multiple copies of the same coefficients [9], or nodes may even need to transmit data away from the sink [15,72,73]. As discussed in [57, 72], this sort of strategy can outperform raw data gathering for very dense networks, but it can lead to significant communication overhead for small to medium sized ones (less than 200 nodes). Other related methods may also suffer from such inefficiencies.

The results of our previous work [54, 55] and of [72] demonstrate why transport costs cannot be ignored. One simple way to work around these issues is to first design an efficient routing tree (e.g., a shortest path routing tree, or SPT), then allow the transform computations to occur only along the routing paths in the tree. We call these types of schemes *en-route in-network transforms*. These transforms (e.g., the wavelet transforms in [1, 6–9, 55]) will typically be more efficient since they are computed as data is routed to the sink along efficient routing paths. In addition to overall efficiency, these transforms can be easily integrated on top of existing routing protocols, i.e., a routing tree can be given by a protocol, then the transform can be constructed along the tree. This allows such schemes to be easily usable in a WSN - as demonstrated by the SenZip [41] compression tool, which includes an implementation of our algorithm in [55] - as well as other types of data gathering networks [34, 46].

We note that all existing en-route transforms start from well-known transforms, then modify them to work on routing trees. Instead, in this work we start from a routing tree T and additional links given by broadcast (e.g., Figure 3.1). We then pose the following questions: (i) what is the full set of transforms that can be computed as data is routed toward the sink along T and (ii) what are conditions for invertibility of these transforms? *The main goal of this work is to determine this general set of invertible, en-route in-network transforms.* Note that in many transform-based compression systems, design or selection of a transform is considered separately from the design of a quantization and encoding strategy. This is done in practice in order to simplify the system design (e.g., [67]). In general certain properties of the transform (energy compaction, orthogonality) can serve as indicators of achievable performance in the lossy case. We adopt a similar approach in our work, choosing to only focus on the transform design. Simple quantization and encoding schemes can then be applied to the transform coefficients, as demonstrated in our experimental results. Joint optimization of routing and compression is also possible, as in [40, 48] and in Chapter 4 [54], but this is beyond the scope of this section. Here we only focus on the design of transforms for a fixed routing tree such as, e.g., an SPT.

In order to formulate this problem, we first note that the data gathering process consists of data measurement at each node and routing of data to the sink along T done in accordance with some transmission scheduling, i.e., nodes transmit data along T in a certain order. Also note that data is only transmitted along T in the direction of the sink, i.e., data transmissions are *unidirectional* toward the sink. Moreover, each node can only process its own data with data received from other nodes that transmit before it, i.e., processing of data must be *causal* in accordance with the transmission schedule. In particular, before each node transmits it will

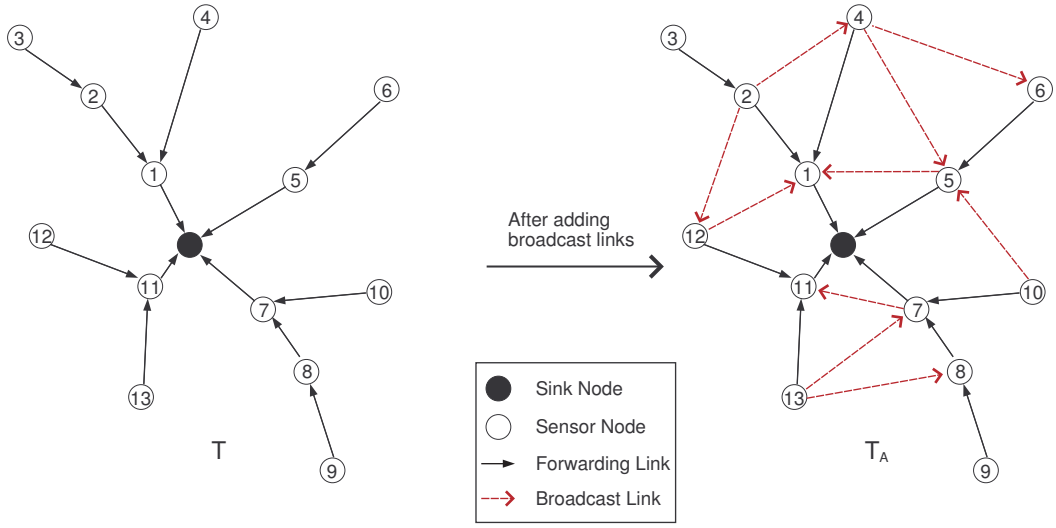


Figure 3.1: Example of routing tree and a tree augmented with broadcasts. Solid arrows denote forwarding links along the tree and dashed arrows denote broadcast links.

only have access to data received from nodes that use it as a relay in a multi-hop path to the sink (i.e., "descendants") and nodes whose data it receives but is not responsible for forwarding to the sink (i.e., "broadcast" neighbors). Whenever broadcast is used, data from a single node will often be available at multiple nodes. While this can help to decorrelate data even further (since more data will be available for transform computations at each node), it would be undesirable to transmit this same piece of data through multiple paths since this would increase the overall communication cost. Thus, in addition to causality and unidirectionality, the transform should also be *critically sampled*, i.e., the number of transform coefficients that are computed and routed to the sink is equal to the number of nodes in the network. We refer to causal, critically-sampled transforms that are computed in a unidirectional manner as *unidirectional transforms*.

As we will show, unidirectional transforms can be defined in terms of the routing tree, the broadcast links induced by the routing and the transmission schedule.

Thus, given a tree and transmission schedule, the main problem we address in this work is to determine a set of necessary and sufficient conditions under which an arbitrary unidirectional transform is invertible. While unidirectional transforms have been proposed, to the best of our knowledge, none of the existing works have attempted to define the most general set of unidirectional transforms, nor has any attempt been made to find conditions under which such transforms are invertible. Our proposed theory also incorporates the use of broadcast data in a general setting. This leads us to develop transforms that use broadcasts in a manner not previously considered. This contribution is discussed in detail in Section 3.2, and was initially proposed by us in [57].

In the context of wavelet transforms for WSNs, early work [1, 6, 7, 9] developed unidirectional wavelet transforms on 1D routing paths in WSNs. Extensions to 2D routing paths on arbitrary routing trees were made by the authors in [54, 55]. The superiority of these 1D [9] and 2D [55] transforms over the method in [72] (which requires a great deal of backward communication) was demonstrated in [55]. General unidirectional transforms were initially proposed by us in [58], in the context of lifting transforms [65], and conditions for single-level invertible unidirectional lifting transforms were initially proposed there. However, no invertibility conditions were provided for general unidirectional transforms, nor were any conditions given for invertible multi-level unidirectional lifting transforms. We provide such conditions here (Section 3.2 and 3.3.4) as well as new transform designs (Section 3.4) that outperform previously proposed transforms.

General unidirectional transforms with a set of necessary and sufficient invertibility conditions are presented in Section 3.2. In order to demonstrate the generality of our proposed theory, Section 3.3 shows how existing unidirectional transforms (e.g., the tree-based KLT [52], tree-based differential pulse code modulation

(T-DPCM) [41, 52] and lifting transforms [52, 58]) can be mapped into our framework. Moreover, our proposed formalism is used to construct general unidirectional lifting transforms. Some of the inefficiencies of existing lifting transforms are then discussed. In order to address these inefficiencies, we define a new Haar-like wavelet transform in Section 3.4 which is analogous to the standard Haar wavelet when applied to 1D paths. As is shown in Section 3.4, our formalization guarantees invertibility of these Haar-like transforms, and also leads to an extension which incorporates broadcast. Section 3.6 provides experimental results that demonstrate the benefits of using our proposed transforms.

3.2 En-route In-network Transforms

In this section, assuming a fixed routing tree T and schedule $t(n)$ are given, we provide a definition of unidirectional transforms and determine conditions for their invertibility. Some notation is established in Section 3.2.1. Unidirectional transforms are then defined in Section 3.2.2. Section 3.2.3 presents a set of conditions under which these transforms are invertible. Throughout this discussion, the configuration of the network in terms routing and scheduling is assumed to be known. Section 3.2.4 addresses how this can be achieved in practice and how our approach can be used with decentralized initialization approaches.

3.2.1 Notation

Assume there are N nodes in the network with a given routing tree $T = (V, E_T)$, where $V = \{1, 2, \ldots, N, N+1\}$, each node is indexed by $n \in \mathcal{I} = \{1, 2, \ldots, N\}$, the sink node is indexed by $N+1$, and $(m, n) \in E_T$ denotes an edge from node m to node n along T. We also assume that there is a graph $G = (V, E)$ which is

defined by the edges in E_T and any additional edges that arise from the broadcast nature of wireless communications. An example graph is shown on the right side of Figure 3.1. We observe that data gathering consists of three key components. The first is *data measurement*, where each node n measures some scalar data $x(n)$ that it must send to the sink in each epoch (these ideas can be easily generalized to non-scalar data[2]). Additionally, node n must route its data to the sink along T. The tree T is defined by assigning to every node n a parent $\rho(n)$. We assume that these trees are provided by a standard routing protocol such as CTP. Finally, we assume that data transmissions are scheduled $[10, 60]$ in some manner, i.e., node n will transmit data to its parent $\rho(n)$ at time $t(n)$ according to a *transmission schedule* (see Definition 2). CTP is a practical example that can be viewed in terms of this formalization: nodes are assigned parents in a distributed manner, data is forwarded to the sink along the corresponding routing paths and the times at which nodes transmit serve as an implicit transmission schedule.

Definition 2 (Transmission Schedule)**.** *A transmission schedule is a function* $t : \mathcal{I} \to \{1, 2, \ldots, M_{slot}\}$*, such that* $t(n) = j$ *when node* n *transmits in the* j*-th time slot*[3]*. Moreover, node* n *transmits data before node* m *whenever* $t(n) < t(m)$*.*

Note that, along the tree T, each node has a set of *descendants* $\mathcal{D}_n$ which use node n as a data relay to the sink and a set of *ancestors* $\mathcal{A}_n$ that node n uses for relaying data to the sink. Also let each node n be $h(n)$ hops away from the sink node, i.e., n has depth $h(n)$ in T. We also let $\mathcal{C}_n^k$ denote the descendants of n which

[2]One straightforward extension is to use a "separable" transform, where a transform is first applied in one dimension (e.g., over time or across dimensions of a multivariate input) and then in the other (i.e., spatially).

[3]Note that these time slots are not necessarily of equal length; they simply allow us to describe the order in which communications proceed in the network; before time slot $t(n)$, node n is listening to other nodes, and at time $t(n)$ node n starts transmitting its own data, and potentially data from its descendants in the routing tree.

are exactly k hops away from n, i.e., $\mathcal{C}_n^k = \{m \in \mathcal{D}_n | \rho^k(m) = n\}$, where $\rho^k(m)$ is the k-th ancestor of node m (e.g., $\rho^1(m)$ is the parent of m, $\rho^2(m)$ is the grandparent of m, etc). For instance, $\mathcal{C}_n^1$ is the set of children of n, $\mathcal{C}_n^2$ is the set of grandchildren of n, etc, and for simplicity we let $\mathcal{C}_n = \mathcal{C}_n^1$. Also note that data can be heard via broadcast in WSNs, so we let $\mathcal{B}_n^f$ define the *full set of broadcast neighbors* whose data node n can overhear due to broadcast.

Under this formulation, each node n can process its own data with data received from $\mathcal{D}_n$ and $\mathcal{B}_n^f$. This yields transform coefficients $y(n)$ and $y(m)$ for each descendant $m \in \mathcal{D}_n$. We make an abuse of notation by letting $y(\mathcal{D}_n) = \{y(m) | m \in \mathcal{D}_n\}$. Since node n is only responsible for forwarding $y(n)$ and $y(\mathcal{D}_n)$ to its parent $\rho(n)$, it should not transmit any data received from broadcast neighbors. In particular, we assume that node n transmits the *transform coefficient vector* $\mathbf{y}_n = [y(n)\, y(\mathcal{D}_n)]^t$ to its parent $\rho(n)$ at time $t(n)$. We refer to this as *critical-sampling*, where in each epoch only one transform coefficient per sample per node is generated and then transmitted to the sink. Also note that $y(n)$ and $y(\mathcal{D}_n)$ can be further processed at the ancestors of n. We refer to this as *delayed processing*.

Note that data is only transmitted along T toward the sink, i.e., data relay is *unidirectional* toward the sink. The existence of a transmission schedule - given explicitly or implicitly - also induces a notion of *causality* for transform computations. In particular, the computations performed at each node n can only involve $x(n)$ and any $\mathbf{y}_m$ received from a node m that transmits data before node n. More specifically, nodes can only use data from $m \in \mathcal{B}_n^f$ if $t(m) < t(n)$ (we assume that $t(m') < t(n)$ for all $m' \in \mathcal{D}_n$). These constraints (i.e., causality and unidirectional relay) induce *causal neighborhoods* whose data each node n can use for processing, where we let $\mathcal{B}_n = \{m \in \mathcal{B}_n^f | t(m) < t(n)\}$ denote the *set of causal broadcast neighbors*. These can be abstracted as in Figure 3.2 where $\mathbf{y}_{\mathcal{D}_n} = \left[\mathbf{y}_{\mathcal{C}_n(1)}^t \cdots \mathbf{y}_{\mathcal{C}_n(|\mathcal{C}_n|)}^t\right]^t$

and $\mathbf{y}_{\mathcal{B}_n} = \left[\mathbf{y}^t_{\mathcal{B}_n(1)} \cdots \mathbf{y}^t_{\mathcal{B}_n(|\mathcal{B}_n|)}\right]^t$. These ideas are illustrated in Figure 3.3. For instance, nodes 4 and 12 will not receive data from node 2 before they transmit, thus, they cannot use it for processing. These are formally defined as follows.

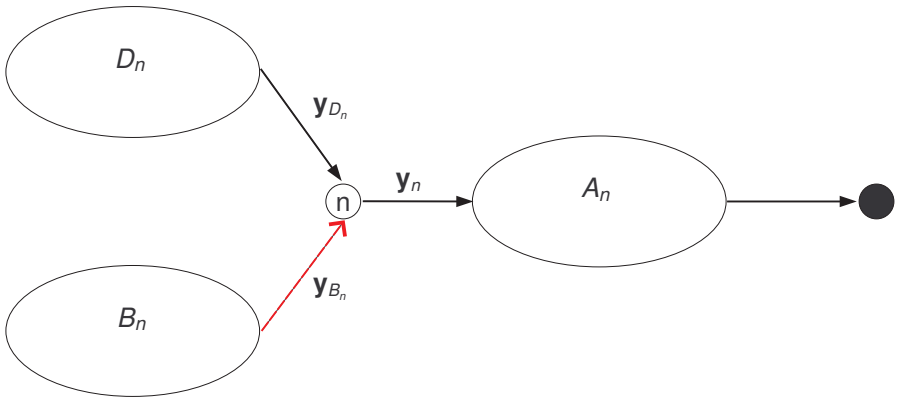


Figure 3.2: Example of causal neighborhoods for each node. Node n receives $\mathbf{y}_{\mathcal{D}_n}$ and $\mathbf{y}_{\mathcal{B}_n}$ from $\mathcal{D}_n$ and $\mathcal{B}_n$, respectively, processes $x(n)$ together with $\mathbf{y}_{\mathcal{D}_n}$ and $\mathbf{y}_{\mathcal{B}_n}$, then forwards its transform coefficient vector $\mathbf{y}_n$ through its ancestors in $\mathcal{A}_n$.

Definition 3 (Causal Neighborhoods)**.** *Given a routing tree T and schedule $t(n)$, the* causal neighborhood *of each node n is the union of the descendants $\mathcal{D}_n$ and the set of causal broadcast neighbors $\mathcal{B}_n = \{m \in \mathcal{B}^f_n | t(m) < t(n)\}$, i.e., $\mathcal{D}_n \cup \mathcal{B}_n$. We also define $\bar{\mathcal{B}}_n = \mathcal{B}_n \cup_{m\in\mathcal{B}_n} \mathcal{D}_m$ for future discussions.*

3.2.2 Definition of Unidirectional Transforms

We define a *unidirectional transform* (not necessarily invertible) as any transform that (i) is computed unidirectionally along a tree T and (ii) satisfies causality and critical sampling. Now we can establish the general algebraic form of unidirectional transforms. Without loss of generality, assume that node indices follow a pre-order numbering [70] on T, i.e., $\mathcal{D}_n = \{n+1, n+2, \ldots, n+|\mathcal{D}_n|\}$ for all n (see Figure 3.3 for an example). A pre-order numbering always exists, and can be found via standard

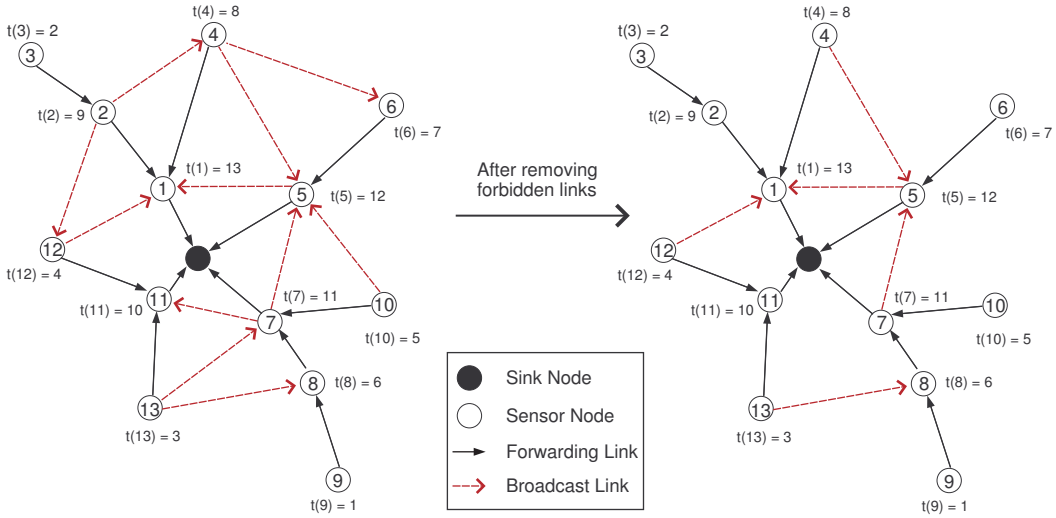


Figure 3.3: Illustration of causal neighborhoods. Node n transmits at time $t(n)$. The left figure shows the full communication graph. The right figure shows the graph after removing broadcast links that violate causality and step by step decoding.

algorithms [70]. For the sake of simplicity, we also assume that the transmission schedule provides a unique time slot to each node[4], i.e., $t(n) \neq t(m)$ for all $n \neq m$.

Recall that each node n receives $\mathbf{y}_{\mathcal{D}_n}$ and $\mathbf{y}_{\mathcal{B}_n}$ from its descendants and (causal) broadcast neighbors, respectively. Thus, in a general unidirectional transform, each node processes its own data $x(n)$ along with $\mathbf{y}_{\mathcal{D}_n}$ and $\mathbf{y}_{\mathcal{B}_n}$. Then, it will transmit coefficient vector $\mathbf{y}_n$ at time $t(n)$. We omit $t(n)$ from the notation of $\mathbf{y}_n$ since the timing is implicit. In order to satisfy critical-sampling, each node should only forward $1 + |\mathcal{D}_n|$ coefficients to the sink. Therefore, $\mathbf{y}_n$ must be a $(1 + |\mathcal{D}_n|) \times 1$ dimensional vector. A unidirectional transform can now be expressed as follows.

Definition 4 (Unidirectional Transform)**.** *Let T be a routing tree with a unique time slot assignment given by $t(n)$, and suppose that the causal neighborhood of*

[4]We note that the time slot assignment need not be unique. However, this assumption significantly simplifies the transform construction and invertibility conditions. It is easy to develop similar transform constructions when multiple nodes are assigned the same time slots, and similar invertibility conditions arise.

each node is given by Definition 3. A unidirectional transform on T is a collection of local transformations done at each node n given by

$$\mathbf{y}_n = \left[\mathbf{A}_n \, \mathbf{B}_n^1 \, \ldots \, \mathbf{B}_n^{|\mathcal{B}_n|}\right] \cdot \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{D}_n} \\ \mathbf{y}_{\mathcal{B}_n} \end{bmatrix}, \tag{3.1}$$

where $\mathbf{y}_n$ has dimension $(1 + |\mathcal{D}_n|) \times 1$, $\mathbf{A}_n$ has dimension $(1 + |\mathcal{D}_n|) \times (1 + |\mathcal{D}_n|)$ and each $\mathbf{B}_n^i$ has dimension $(1 + |\mathcal{D}_n|) \times (1 + |\mathcal{D}_{\mathcal{B}_n(i)}|)$. The transform is computed starting from the node at the first time slot up through the nodes in the remaining time slots $k = 2, 3, \ldots, N$.

3.2.3 Invertibility Conditions for Unidirectional Transforms

We now establish a set of invertibility conditions for unidirectional transforms. Note that these transforms are always computed in a particular order, e.g., starting from nodes furthest from the sink (i.e., "leaf" nodes), up to nodes which are 1-hop from the sink. Some sort of interleaved scheduling (where one set of nodes transmits before the rest) could also be used [37]. Therefore, it would also be desirable to have step by step decoding in the reverse order, since this would simplify the transform constructions. In particular, if the overall transform can be inverted by inverting the computations done at each node in the reverse order, then invertibility will be ensured by designing invertible transforms at each node.

Step by step decoding in the reverse order is trivially guaranteed when no broadcast data is used since the transform at each node n is simply $\mathbf{y}_n = \mathbf{A}_n \cdot \left[x(n) \, \mathbf{y}_{\mathcal{D}_n}^t\right]^t$. Thus, if each $\mathbf{A}_n$ is invertible, we can invert the operations done at node n as $\left[x(n) \, \mathbf{y}_{\mathcal{D}_n}^t\right]^t = (\mathbf{A}_n)^{-1} \cdot \mathbf{y}_n$. This becomes more complicated when broadcast data is

used. By examining (3.1), we observe that $\mathbf{y}_n = \mathbf{A}_n \cdot \left[x(n)\, \mathbf{y}^t_{\mathcal{D}_n}\right]^t + \left[\mathbf{B}^1_n \ \ldots \ \mathbf{B}^{|\mathcal{B}_n|}_n\right] \cdot \mathbf{y}_{\mathcal{B}_n}$, where $\mathbf{y}_{\mathcal{B}_n} = \left[\mathbf{y}^t_{\mathcal{B}_n(1)} \ \cdots \ \mathbf{y}^t_{\mathcal{B}_n(|\mathcal{B}_n|)}\right]^t$. In order to have step by step decodability, we need to be able to recover (for every node n) $x(n)$ and $\mathbf{y}_{\mathcal{D}_n}$ from $\mathbf{y}_n$ and $\mathbf{y}_{\mathcal{B}_n}$. Note that this fails whenever we cannot decode some transform coefficient vector $\mathbf{y}_m$ from broadcast node $m \in \mathcal{B}_n$ before decoding $\mathbf{y}_n$. It will also fail if the matrix operations performed at any given node are not invertible. Thus, in order to guarantee step by step decodability, we need to ensure that (i) the matrix operations at each node are invertible, and (ii) it is possible to decode each $\mathbf{y}_m$ before decoding $\mathbf{y}_n$. As we now show, (i) is guaranteed by ensuring that each $\mathbf{A}_n$ matrix is invertible and (ii) is guaranteed by imposing a timing condition.

Proposition 5 (Step by Step Decodability)**.** *Suppose that we have the transform in Definition 4 and assume that $t(\rho(m)) > t(n)$ for every broadcast node $m \in \mathcal{B}_n$. Then we can recover $x(n)$ and $\mathbf{y}_{\mathcal{D}_n}$ as*

$$\left[x(n)\, \mathbf{y}^t_{\mathcal{D}_n}\right]^t = \mathbf{A}_n^{-1} \cdot \mathbf{y}_n - \mathbf{A}_n^{-1} \cdot \left[\mathbf{B}^1_n \ \ldots \ \mathbf{B}^{|\mathcal{B}_n|}_n\right] \cdot \mathbf{y}_{\mathcal{B}_n} \tag{3.2}$$

if and only if $\mathbf{A}_n^{-1}$ exists.

Proof. Note that the vector transmitted by any broadcast node $m \in \mathcal{B}_n$ will be processed at its parent, node $\rho(m)$, and this processing will occur at time $t(\rho(m))$. Moreover, node n will generate its own transform coefficient vector $\mathbf{y}_n$ at time $t(n)$, and by assumption we have that $t(\rho(m)) > t(n)$. Thus, it is possible to decode $\mathbf{y}_m$ before $\mathbf{y}_n$ for every broadcast neighbor $m \in \mathcal{B}_n$. It follows that, we can always form $\mathbf{y}_{\mathcal{B}_n} = \left[\mathbf{y}^t_{\mathcal{B}_n(1)} \ \cdots \ \mathbf{y}^t_{\mathcal{B}_n(|\mathcal{B}_n|)}\right]^t$ before decoding $\mathbf{y}_n$. Therefore, we can recover $x(n)$ and $\mathbf{y}_{\mathcal{D}_n}$ as in (3.2) if and only if $\mathbf{A}_n^{-1}$ exists. □

To simplify our transform constructions, we *also assume* that nodes use the latest version of broadcast data that they receive, i.e., $m \in \mathcal{B}_n$ only if $\mathcal{A}_m \cap \mathcal{B}_n = \emptyset$. This *second constraint* precludes the possibility that a node n receives broadcast data from node m and from an ancestor of node m. Removing the broadcast links which violate these constraints gives a simplified communication graph as shown on the right side of Figure 3.3. Removal of these links can be done by local information exchange within the network; examples of how this can be achieved are discussed in Section 3.2.4. Under the constraint of Prop. 5 and this second constraint, we can represent the global transform taking place in the network as follows. Since the time slot assignment is unique, at time $t(n)$ only data from n and its descendants will be modified, i.e., only $x(n)$ and $y(\mathcal{D}_n)$ will be changed at time $t(n)$. Since pre-order indexing is used, we have that $\mathbf{y}_{\mathcal{D}_n} = [y(n+1),\ \ldots,\ y(n+|\mathcal{D}_n|)]^t$. Therefore, the global transform computations done at time $t(n)$ are given by (3.3), where each $\tilde{\mathbf{y}}_i$ corresponds to data which is not processed at time $t(n)$.

$$\begin{bmatrix} \tilde{\mathbf{y}}_1 \\ \mathbf{y}_{\mathcal{B}_n(1)} \\ \vdots \\ \tilde{\mathbf{y}}_k \\ \mathbf{y}_n \\ \tilde{\mathbf{y}}_{k+1} \\ \vdots \\ \mathbf{y}_{\mathcal{B}_n(|\mathcal{B}_n|)} \\ \tilde{\mathbf{y}}_K \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_n^1 & \ldots & \mathbf{0} & \mathbf{A}_n & \mathbf{0} & \ldots & \mathbf{B}_n^{|\mathcal{B}_n|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{I} & \ldots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{y}}_1 \\ \mathbf{y}_{\mathcal{B}_n(1)} \\ \vdots \\ \tilde{\mathbf{y}}_k \\ \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{D}_n} \end{bmatrix} \\ \tilde{\mathbf{y}}_{k+1} \\ \vdots \\ \mathbf{y}_{\mathcal{B}_n(|\mathcal{B}_n|)} \\ \tilde{\mathbf{y}}_K \end{bmatrix} \tag{3.3}$$

The *global transform matrix* $\mathbf{C}_{t(n)}$ at time $t(n)$ is just

$$\mathbf{C}_{t(n)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_n^1 & \dots & \mathbf{0} & \mathbf{A}_n & \mathbf{0} & \dots & \mathbf{B}_n^{|\mathcal{B}_n|} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{3.4}$$

This yields the *global transform coefficient vector*

$$\mathbf{y} = \mathbf{C}_N \cdot \mathbf{C}_{N-1} \cdots \mathbf{C}_1 \cdot \mathbf{x}. \tag{3.5}$$

Figure 3.4 illustrates these computations. Initially, $\mathbf{y} = \mathbf{x} = [x(1)\ x(2)\ \dots\ x(5)]^t$. At times 1 and 2, nodes 3 and 5, respectively, transmit raw data to their parents. Therefore, the global matrices at times 1 and 2 are simply $\mathbf{C}_1 = \mathbf{C}_2 = \mathbf{I}$. At time 3, node 4 produces (3.6), where a_i and b_i represent arbitrary values of the transform matrix used at node 4. Then at time 4, node 2 produces transform coefficients $y(2)$ and $y(3)$ (and coefficient vector $\mathbf{y}_2$) as in (3.7), where a_i' and b_i' are the values of the matrix used at node 2.

$$\mathbf{y}_4 = \begin{bmatrix} y(4) \\ y(5) \end{bmatrix} = \begin{bmatrix} b_1 & a_1 & a_2 \\ b_2 & a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} x(3) \\ x(4) \\ x(5) \end{bmatrix} \tag{3.6}$$

$$\mathbf{y}_2 = \begin{bmatrix} y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} a_1' & a_2' & b_1' & b_2' \\ a_3' & a_4' & b_3' & b_4' \end{bmatrix} \cdot \begin{bmatrix} x(2) \\ x(3) \\ \mathbf{y}_4 \end{bmatrix} \tag{3.7}$$

Node 1 then computes $\mathbf{y}_1$ at time 5. The global transform is given by

$$\mathbf{y} = \mathbf{A}_1 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & a_1' & a_2' & b_1' & b_2' \\ 0 & a_3' & a_4' & b_3' & b_4' \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & b_1 & a_1 & a_2 \\ 0 & 0 & b_2 & a_3 & a_4 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \end{bmatrix}. \tag{3.8}$$

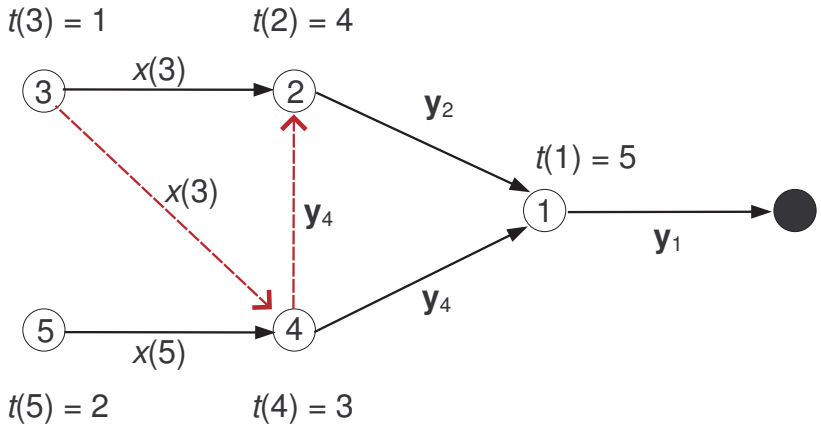


Figure 3.4: Example to illustrate unidirectional computations. Nodes generate and transmit transform coefficients in the order specified by the transmission schedule.

It is now simple to show that the transform is invertible if each $\mathbf{A}_n$ is invertible.

Proposition 6 (Invertible Unidirectional Transforms)**.** *Suppose that we have the transform in Def. 4, the second timing constraint ($m \in \mathcal{B}_n$ only if $\mathcal{A}_m \cap \mathcal{B}_n = \emptyset$) is met, and Prop. 5 is satisfied for every node n. Then the overall transform given by (3.5) is invertible.*

Proof. Under the two broadcast timing assumptions, the global transform is given by (3.5). (3.5) is invertible if and only if every $\mathbf{C}_{t(n)}$ in (3.4) is invertible. $\mathbf{C}_{t(n)}$ is invertible if and only if $\det\left(\mathbf{C}_{t(n)}\right) \neq 0$. Recall that adding a multiple of one row to another does not change the determinant [63]. Given the structure of the $\mathbf{C}_{t(n)}$ matrices, using such row operations to eliminate each $\mathbf{B}_n^i$ matrix, it is easy to show that $\det\left(\mathbf{C}_{t(n)}\right) = \det\left(\mathbf{A}_n\right)$. Moreover, Prop. 5 implies that $\mathbf{A}_n$ is invertible. □

Proposition 6 shows that locally invertible transforms provide globally invertible transforms. Moreover, under our stated timing constraints, broadcast data does not affect invertibility. Therefore, broadcast data at each node n can be used in an arbitrary manner without affecting invertibility. So in order to design an invertible unidirectional transform, all that one must do is design invertible matrices $\mathbf{A}_n$. This is an encouraging result since it essentially means that broadcast data can be used in any way a node chooses. In particular, broadcast data can always be used to achieve more data decorrelation.

3.2.4 Discussion

The theory presented thus far assumes that the routing and transmission scheduling are known, and that all of the transform matrices are known both at the nodes and at the sink. In practice, the routing, scheduling and transforms must be initialized. Moreover, the network may need to re-configure itself if, for example, nodes die or link conditions change drastically. In addition, packet losses will often occur. Nodes typically deal with this (as in CTP) by re-transmitting a packet until an acknowledgment (ACK) is received from the intended recipient. While these three issues pose no significant problems for routing, they all have an impact on our

proposed transform due to the assumptions we make about timing. We now provide some discussion of how this affects our theory and how it can be handled.

We first address the impact that initialization and reconfiguration have on the routing and scheduling, as well as what can be done to address it. We assume that routing is initialized and reconfigured in a distributed manner using standard protocols such as CTP. Distributed scheduling protocols for WSNs also exist [60,61]. However, the resulting schedules may not be consistent with Definition 3 (i.e., they may not provide timings for which $t(m) < t(n)$ for all $m \in \mathcal{D}_n$), so in practice we would need to enforce such timings. One way to achieve this is to force nodes to suppress transmission (in a given epoch) until they have received data from all of their descendants. Another alternative would be to determine such a transmission schedule at the sink, then to disseminate the timing information to the nodes.

Whenever timing and routing information is established (or re-established due to re-configuration), it is also necessary to check our main broadcast timing constraint, i.e., $m \in \mathcal{B}_n$ only if $t(n) < t(\rho(m))$. We describe one way in which this information can be disseminated to each node in a distributed manner. First, whenever the time $t(n)$ at node n is initialized or changes, it broadcasts a small packet (i.e., a *beacon*) which contains $t(n)$ to its children. Then, any child of n which broadcasts data will send the same beacon to all of its neighbors. This requires a total of 2 messages for each broadcasting node. Note that protocols such as CTP already use control beacons (in addition to data packets) to update stale routing information. Thus, nodes could potentially piggyback timing information on these control beacons whenever they are generated, or otherwise use separate control beacons to disseminate timing information. This will incur an additional cost, although (as was shown in [68]) the per packet cost for control beacons is typically much smaller than the cost for data forwarding.

Initialization and re-configuration also impacts the transform matrices that are used. Each node could transmit the values of its matrix to the sink, or vice versa, but this may be very costly. Instead, the construction of each transform matrix should be based on a small amount of information which is made common to the nodes and to the sink. For example, the values in each transform matrix could be based on the number of 1-hop neighbors that each node has [55] or the relative node positions [73]. In this way each matrix can be constructed at each node and at the sink without explicitly communicating the matrix values. However, additional information (e.g., node positions, number of neighbors) would need to be communicated to the sink whenever the network is initialized or re-configured. For example, each node could construct a transform using only the number of nodes that it receives data from (as in [55, 73]) and would send the set of nodes whose data it used as overhead to the sink. Then, assuming that the nodes and the sink construct the matrices according to the same rules, the sink can re-construct the matrix used at each node.

Packet loss is the last practical issue which impacts our proposed transforms. We do not consider the effects of channel noise on the data since these can be handled using a wide variety of existing techniques. Moreover, packet losses and channel noise will impact other data gathering schemes (e.g., CTP), and we expect that the penalty due to packet losses will be similar in our scheme and in other data gathering schemes. Packet losses are typically handled (as in CTP) by re-transmitting a packet until an ACK is received from the desired destination. Thus, if node n does not receive data from descendant $n+k$ by the time that it transmits, due to packet re-transmissions for $n+k$, the data from node $n+k$ cannot be combined with data available at node n. This is equivalent to not using the data from node $n+k$ in the transform computation (i.e., $\mathbf{A}_n(j, k+1) = \mathbf{A}_n(k+1, j) = 0$

for all $j \neq k+1$ and $\mathbf{A}_n(k+1, k+1) = 1$) and does not affect our proposed theory. However, this change must be signaled to the sink so that it knows how to adjust $\mathbf{A}_n$ accordingly. This can be done by including some additional information in the packet headers for node n and $n+k$ to signify this change.

Packet losses also have an impact on the use of broadcast data. Suppose that node n does not receive a data packet from broadcast neighbor b_k but the packet from b_k does reach the intended recipient $\rho(b_k)$. In this case, node $\rho(b_k)$ will send an ACK back to node b_k and node b_k will no longer re-transmit (note that node b_k will not expect an ACK from node n). Thus, data from node b_k can not be combined with data available at node n. This is equivalent to not using data from node b_k in the transform computation (i.e., $\mathbf{B}_n^k = \mathbf{0}$) and our proposed theory is not affected. However, node n must signal this change to the sink node so that it knows to set $\mathbf{B}_n^k = \mathbf{0}$.

One way to work around these issues (initialization, re-configuration and packet losses) is to design transforms that can work under arbitrary timing and with arbitrary uses of broadcast data. However, under arbitrary timing and use of broadcast data, it is no longer possible to guarantee global transform invertibility by designing invertible transforms at each node. More specifically, we must ensure that the transform computations done at different nodes are jointly invertible. This leads to a set of complex conditions. The cost to determine such conditions and to coordinate nodes so that they satisfy these conditions could be very high, perhaps even much higher than the additional coordination needed to implement our proposed transforms. However, it is still possible to design simple versions of such transforms by using constructions such as lifting. The transforms described in Section 2.2.2 [37] is one particular example. Given this high degree of complexity to ensure an invertible transform when using broadcast, broadcast data should probably only be used

with our proposed transforms if (i) it is possible to fix the timing in the network in accordance with Definition 3, and, (ii) the timing is very stable.

3.3 Unidirectional Transform Designs

Proposition 6 provides simple conditions for invertible transform design, i.e., $\mathbf{A}_n$ is invertible for every node n. This is a simple design constraint that *unifies many existing unidirectional transforms*. In this section, we demonstrate how existing unidirectional transforms can be mapped to our formulation. In particular, we focus on the tree-based Karhunen-Loève Transform (T-KLT) [52], T-DPCM [41, 52] and early forms of tree-based wavelet transforms [1,7,9,55] constructed using lifting [65].

In order to exploit spatial correlation to achieve reduction in the number of bits per measurement, nodes must first exchange data. Therefore, some nodes must transmit raw data to their neighbors before any form of spatial compression can be performed. Since raw data typically requires many more bits than encoded transform coefficients, it would be desirable to minimize the number of raw data transmissions that nodes must make to facilitate distributed transform computation. Therefore, our *main design consideration* is to minimize the number of raw data transmissions that are required to compute the transform.

3.3.1 Tree-based Karhunen-Loève Transform

Since transforms that achieve data decorrelation potentially lead to better coding efficiency [17], we consider now the design of unidirectional transforms that achieve the maximum amount of data decorrelation. This can be achieved by applying, at each node n, a transform $\mathbf{A}_n$ that makes all of the coefficients in $\mathbf{y}_n$ statistically uncorrelated (or "whitened"), e.g., by using a Karhunen-Loève transform (KLT)

at each node, leading to the T-KLT described in our previous work [52]. In this transform, each node n computes and transmits a set of "whitened" coefficients $\mathbf{y}_n$, which will then have to be "unwhitened" and then re-whitened at $\rho(n)$ to produce a new set of whitened coefficients. Whitening can be done using a KLT and unwhitening can be achieved using an inverse KLT. More specifically, this is done at each node n by (i) finding the whitening transform $\mathbf{H}_n$ and unwhitening transforms of each child $\mathbf{G}_{\mathcal{C}_n(i)}$, (ii) applying an unwhitening transform to each child to recover the original measurements as $\mathbf{x}_{\mathcal{C}_n(i)} = \mathbf{G}_{\mathcal{C}_n(i)} \cdot \mathbf{y}_{\mathcal{C}_n(i)}$, and then (iii) rewhitening these measurements as $\mathbf{y}_n = \mathbf{H}_n \cdot \left[x(n)\, \mathbf{x}^t_{\mathcal{C}_n(1)} \ \ldots \mathbf{x}^t_{\mathcal{C}_n(|\mathcal{C}_n|)}\right]^t$. Thus,

$$\mathbf{y}_n = \mathbf{H}_n \cdot \begin{bmatrix} 1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{G}_{\mathcal{C}_n(1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{G}_{\mathcal{C}_n(|\mathcal{C}_n|)} \end{bmatrix} \cdot \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{C}_n(1)} \\ \vdots \\ \mathbf{y}_{\mathcal{C}_n(|\mathcal{C}_n|)} \end{bmatrix}, \tag{3.9}$$

with $\mathbf{A}_n = \mathbf{H}_n \cdot \mathrm{diag}\left(1, \mathbf{G}_{\mathcal{C}_n(1)}, \ldots, \mathbf{G}_{\mathcal{C}_n(|\mathcal{C}_n|)}\right)$. Each $\mathbf{A}_n$ is trivially invertible since $\mathbf{H}_n$ and each $\mathbf{G}_{\mathcal{C}_n(i)}$ are invertible by construction.

3.3.2 Orthogonal Unidirectional Transforms

It may also be desirable to construct orthogonal transforms on an arbitrary tree T. Given the assumptions in Section 3.2, we have that the transform $\mathbf{C}_{t(n)}$ is orthogonal if and only if $\left(\mathbf{C}_{t(n)}\right)^t \cdot \mathbf{C}_{t(n)} = \mathbf{C}_{t(n)} \cdot \left(\mathbf{C}_{t(n)}\right)^t = \mathbf{I}$, which holds if and only if $\mathbf{A}_n^t = \mathbf{A}_n^{-1}$ and $\mathbf{B}_n^i = \mathbf{0}$. Thus, under the formulation in Section 3.2, a unidirectional transform is orthogonal only if broadcast data is not used.

3.3.3 Tree-based DPCM

A simpler alternative to the T-KLT is T-DPCM [41, 52]. A related DPCM based method was proposed in [31]. The method in [31] is not designed for any particular communication structure, but it can easily be adapted to take the form of a uni-directional transform. In contrast to the method in [31], the T-DPCM methods in our previous work [41, 52] compute differentials directly on a tree such as an SPT.

In the T-DPCM method of [52], each node n computes its difference with respect to a weighted average of its children's data, i.e., $y(n) = x(n) - \sum_{m \in \mathcal{C}_n} \mathbf{a}_n(m)x(m)$. For this to be possible, one of two things must happen: either every node n must decode the differentials received from its children to recover $x(m)$ for each $m \in \mathcal{C}_n$, or, every node n must transmit raw data two hops forward to its grandparent (at which point $y(n)$ can be computed) to avoid decoding data at every node. In order to avoid each node having to forward raw data two hops, at each node n, the inverse transform on the data of each child $\mathcal{C}_n(i)$ must be computed first using the inverse matrix $\left(\mathbf{A}_{\mathcal{C}_n(i)}\right)^{-1}$ of each child. The forward transform is then designed accordingly. We can express this version of T-DPCM as:

$$\mathbf{y}_n = \begin{bmatrix} 1 & -\mathbf{a}_n(\mathcal{D}_n) \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 1 & \left(\mathbf{A}_{\mathcal{C}_n(1)}\right)^{-1} & \cdots & \left(\mathbf{A}_{\mathcal{C}_n(|\mathcal{C}_n|)}\right)^{-1} \\ \mathbf{0} & \mathbf{I} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I} \end{bmatrix} \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{C}_n(1)} \\ \vdots \\ \mathbf{y}_{\mathcal{C}_n(|\mathcal{C}_n|)} \end{bmatrix}. \tag{3.10}$$

Moreover, only leaf nodes need to forward raw data and the rest transmit only transform coefficients.

Alternatively, in the T-DPCM scheme of [41], each node n first forwards raw data $x(n)$ to its parent $\rho(n)$, then node $\rho(n)$ computes a differential for n and

forwards it to the sink, i.e., node $\rho(n)$ computes $y(n) = x(n) - \mathbf{a}_n(\rho(n))x(\rho(n))$. This transform can also be mapped to our formalism as

$$\mathbf{y}_n = \begin{bmatrix} 1 & \mathbf{0} \\ -\mathbf{a}_{\mathcal{D}_n}(n) & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{D}_n} \end{bmatrix}. \tag{3.11}$$

This eliminates the computational complexity of the previous T-DPCM method since no decoding of children data is required. However, every node must now forward raw data one hop. Moreover, it will not decorrelate the data as well as the first method since only data from one neighbor is used.

3.3.4 Unidirectional Lifting-based Wavelets

We now describe how unidirectional wavelet transforms can be constructed under our framework as was initially proposed by us in [57]. This can be done using lifting [65]. Lifting transforms are constructed by *splitting* nodes into disjoint sets of *even* and *odd* nodes, by designing *prediction filters*, which alter odd data using even data, and *update filters*, which alter even data based on odd data. They are invertible by construction [65].

Recall from Chapter 2 that nodes are split into odd and even sets $\mathcal{O}$ and $\mathcal{E}$, respectively. This can be done completely arbitrarily. One example from Chapter 2.2.1 [55] is to split according to the depth in the tree, e.g., as illustrated in Figure 3.5. Data at each odd node $n \in \mathcal{O}$ is then predicted using data from even neighbors $\mathcal{N}_n \subset \mathcal{E}$, yielding detail coefficient $d(n) = x(n) - \sum_{i \in \mathcal{N}_n} \mathbf{p}_n(i)x(i)$. Incorporating some broadcast data into the prediction is also useful since it allows odd nodes to achieve even further decorrelation. After the prediction step, data at each

even node $m \in \mathcal{E}$ is updated using details from odd neighbors $\mathcal{N}_m \subset \mathcal{O}$, yielding smooth coefficient $s(m) = x(m) + \sum_{j \in \mathcal{N}_m} \mathbf{u}_m(j) d(j)$.

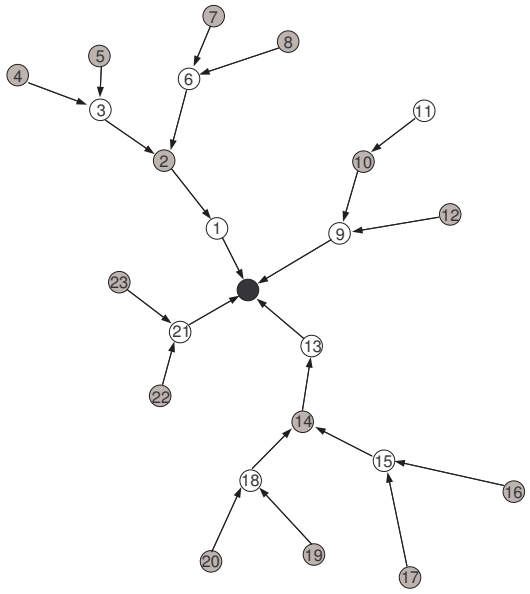


Figure 3.5: Example of splitting based on the depth of the routing tree. White (odd depth) nodes are odd, gray (even depth) nodes are even and the black center node is the sink.

As was shown in Section 2.1, invertibility will be guaranteed as long as (i) odd node data is only predicted using even node data, and (ii) even node data is only updated using details from odd nodes. So if $\mathcal{E}$ and $\mathcal{O}$ is an arbitrary even and odd split, the transform computed at each node will be invertible as long as the computations satisfy (i) and (ii). We are particularly interested in designing *unidirectional* lifting transforms, thus, we must constrain the set of neighbors for each node to its descendants $\mathcal{D}_n$ and its causal broadcast neighbors $\mathcal{B}_n$. More formally, let $\mathcal{O}_n = (n \cup \mathcal{D}_n) \cap \mathcal{O}$ be the set of odd nodes whose data is available at n from its subtree. Let $\mathcal{E}_n = (n \cup \mathcal{D}_n) \cap \mathcal{E}$ be defined similarly. Moreover, let $\mathcal{O}_n^{\mathcal{B}} = \bar{\mathcal{B}}_n \cap \mathcal{O}$ denote the set of odd nodes whose data n receives via broadcast. Similarly, let $\mathcal{E}_n^{\mathcal{B}} = \bar{\mathcal{B}}_n \cap \mathcal{E}$. Then the computations at n will be invertible as long

as it only predicts $y(\mathcal{O}_n)$ from $y(\mathcal{E}_n)$ and $y(\mathcal{E}_n^{\mathcal{B}})$ and only updates $y(\mathcal{E}_n)$ from $y(\mathcal{O}_n)$ and $y(\mathcal{O}_n^{\mathcal{B}})$. Let $\mathbf{M}_n$ and $\mathbf{M}_n^{\mathcal{B}}$ be permutation matrices such that

$$\begin{bmatrix} y(\mathcal{O}_n) \\ y(\mathcal{E}_n) \\ y(\mathcal{O}_n^{\mathcal{B}}) \\ y(\mathcal{E}_n^{\mathcal{B}}) \end{bmatrix} = \begin{bmatrix} \mathbf{M}_n & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_n^{\mathcal{B}} \end{bmatrix} \cdot \begin{bmatrix} x(n) \\ y(\mathcal{D}_n) \\ y(\bar{\mathcal{B}}_n) \end{bmatrix}. \tag{3.12}$$

Then node n can compute transform coefficients as

$$\mathbf{y}_n = (\mathbf{M}_n)^t \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{U}_n & \mathbf{I} & \mathbf{U}_n^{\mathcal{B}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{P}_n & \mathbf{0} & \mathbf{P}_n^{\mathcal{B}} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} y(\mathcal{O}_n) \\ y(\mathcal{E}_n) \\ y(\mathcal{O}_n^{\mathcal{B}}) \\ y(\mathcal{E}_n^{\mathcal{B}}) \end{bmatrix}. \tag{3.13}$$

By multiplying these matrices together, we get $\mathbf{y}_n = [\mathbf{A}_n \, \mathbf{B}_n] \cdot \left[x(n) \, \mathbf{y}_{\mathcal{D}_n}^t \, \mathbf{y}_{\mathcal{B}_n}^t\right]^t$, with

$$\mathbf{A}_n = (\mathbf{M}_n)^t \cdot \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{U}_n & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \mathbf{P}_n \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \mathbf{M}_n,$$

$$\mathbf{B}_n = (\mathbf{M}_n)^t \cdot \begin{bmatrix} \mathbf{0} & \mathbf{P}_n^{\mathcal{B}} \\ \mathbf{U}_n^{\mathcal{B}} & \mathbf{U}_n \mathbf{P}_n^{\mathcal{B}} \end{bmatrix} \cdot \mathbf{M}_n^{\mathcal{B}}.$$

Since $\det(\mathbf{A}_n) = 1$, single-level unidirectional lifting transforms are invertible.

The transform given by (3.13) corresponds to only one level of decomposition. In particular, at each node n the transform of (3.13) will yield a set of smooth (or low-pass) coefficients $\{y(k)\}_{k \in \mathcal{E}_n}$ and a set of detail (or high-pass) coefficients $\{y(l)\}_{l \in \mathcal{O}_n}$. The high-pass coefficients will typically have low energy if the original

data is smooth, so these can be encoded using very few bits and forwarded to the sink without any further processing. However, there will still be some correlation between low-pass coefficients. It would therefore be useful to apply additional levels of transform to the low-pass coefficients at node n to achieve more decorrelation. This will reduce the number of bits needed to encode these low-pass coefficients, and will ultimately reduce the number of bits each node must transmit to the sink.

Suppose each node performs an additional J levels of lifting transform on the low-pass coefficients $\{y(k)\}_{k\in\mathcal{E}_n}$. At each level $j = 2, 3, \ldots, J+1$, suppose that nodes in $\mathcal{E}_n^{j-1}$ are split into even and odd sets $\mathcal{E}_n^j$ and $\mathcal{O}_n^j$, respectively. We assume that $\mathcal{E}_n^1 = \mathcal{E}_n$. For each odd node $l \in \mathcal{O}_n^j$, we predict $y(l)$ using even coefficients from some set of even neighbors $\mathcal{N}_l^j \subset \mathcal{E}_n^j$, i.e., $y(l) = y(l) - \sum_{k\in\mathcal{N}_l^j} \mathbf{p}_{l,j}(k)y(k)$. Then for each even node $k \in \mathcal{E}_n^j$, we update $y(k)$ using odd coefficients from some set of odd neighbors $\mathcal{N}_k^j \subset \mathcal{O}_n^j$, i.e., $y(k) = y(k) + \sum_{l\in\mathcal{N}_k^j} \mathbf{u}_{k,j}(l)y(l)$. This decomposition is done starting from level $j = 2$ up to level $j = J+1$. For all $j = 2, 3, \ldots, J+1$, let $\mathbf{M}_n^j$ be a permutation matrix such that

$$\begin{bmatrix} y(\mathcal{O}_n^j) \\ y(\mathcal{E}_n^j) \\ y(\mathcal{R}_n^j) \end{bmatrix} = \mathbf{M}_n^j \cdot \mathbf{y}_n, \tag{3.14}$$

where $\mathcal{R}_n^j = (n \cup \mathcal{D}_n) - (\mathcal{O}_n^j \cup \mathcal{E}_n^j)$ is the set of nodes whose coefficients are not modified at level j. Then we can express the level j transform computations in matrix form as

$$\mathbf{y}_n = \left(\mathbf{M}_n^j\right)^t \cdot \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{U}_n^j & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \mathbf{P}_n^j & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} y(\mathcal{O}_n^j) \\ y(\mathcal{E}_n^j) \\ y(\mathcal{R}_n^j) \end{bmatrix}, \tag{3.15}$$

where $\mathbf{P}_n^j$ and $\mathbf{U}_n^j$ represent the prediction and update operations used at level j, respectively.

By combining (3.12), (3.13), (3.14) and (3.15), we finally get that $\mathbf{y}_n = [\mathbf{A}_n\ \mathbf{B}_n] \cdot \left[x(n)\ \mathbf{y}_{\mathcal{D}_n}^t\ \mathbf{y}_{\mathcal{B}_n}^t\right]^t$, with $\mathbf{A}_n$ and $\mathbf{B}_n$ defined in (3.16) and (3.17).

$$\mathbf{A}_n = \prod_{j=2}^{J+1} \mathbf{M}_n^{jt} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{U}_n^j & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{P}_n^j & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{M}_n^j \mathbf{M}_n^t \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{U}_n & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{P}_n \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{M}_n \tag{3.16}$$

$$\mathbf{B}_n = \prod_{j=2}^{J+1} \mathbf{M}_n^{jt} \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{U}_n^j & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{P}_n^j & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \mathbf{M}_n^j \mathbf{M}_n^t \begin{bmatrix} \mathbf{0} & \mathbf{P}_n^{\mathcal{B}} \\ \mathbf{U}_n^{\mathcal{B}} & \mathbf{U}_n \mathbf{P}_n^{\mathcal{B}} \end{bmatrix} \mathbf{M}_n^{\mathcal{B}} \tag{3.17}$$

Prop. 6 implies that the overall transform is invertible if $\mathbf{A}_n$ given in (3.16) is invertible. Since each $\mathbf{M}_n^j$ is a permutation matrix, $|\det(\mathbf{M}_n^j)| = 1$. Moreover, the remaining matrices are triangular. Thus, it easily follows that $\det(\mathbf{A}_n) = 1$. Therefore, unidirectional, multi-level lifting transforms are always invertible.

3.3.5 Unidirectional 5/3-like Wavelets

This section describes the *5/3-like transform* on a tree initially proposed in the author's previous work [55]. First, nodes are split into odd and even sets $\mathcal{O}$ and $\mathcal{E}$, respectively, by assigning nodes of odd depth as odd and nodes of even depth as even (as done in Section 2.2.1). This is illustrated in Figure 3.5. The transform neighbors of each node are simply $\mathcal{N}_n = \{\rho(n)\} \cup \mathcal{C}_n$ for every node n. This provides a *5/3-like* wavelet transform on a tree since whenever predictions and updates are used along a 1D path, the transform reduces to the 5/3 wavelet transform [33].

This transform can be computed in a unidirectional manner, but doing so requires that some nodes forward raw data 1 or 2 hops. This is illustrated in Figure 3.6.

Figure 3.6: Raw data example. Nodes 3 and 6 need $x(2)$ to compute details $d(3)$ and $d(6)$, so they must forward raw data over 1-hop to node 2. Nodes 4 and 5 need $d(3)$ to compute $s(4)$ and $s(5)$, so they must forward raw data over 2-hops.

Data from each odd node n is predicted using data $x(\mathcal{C}_n)$ (from children $\mathcal{C}_n$) and $x(\rho(n))$ (from parent $\rho(n)$). However, odd node n will not have $x(\rho(n))$ locally available for processing. Therefore, we require that each odd node n transmit raw data $x(n)$ one hop forward to its parent $\rho(n)$, at which point node $\rho(n)$ can compute the detail coefficient of n. Each even node m will then compute detail $d(j) = x(j) - \sum_{i \in \mathcal{C}_j} \mathbf{p}_l(j)x(j) - \mathbf{p}_j(m)x(m)$ for every child $j \in \mathcal{C}_m$. Similarly, the smooth coefficient of each even node m requires details from its parent $\rho(m)$ and children $\mathcal{C}_m$, so it can not be locally computed either. Moreover, detail $d(\rho(m))$ can only be computed at node $\rho^2(m)$, i.e., at the grandparent of m. Therefore, we require that even node m transmit raw data $x(m)$ two hops forward to $\rho^2(m)$, at which point $d(\rho(m))$ will be available and $\rho^2(m)$ can compute $s(m) = x(m) + \sum_{j \in \{\rho(m)\} \cup \mathcal{C}_m} \mathbf{u}_m(j)d(j)$. Note that each of these operations are trivially invertible, and easily lead to local transform matrices $\mathbf{A}_n$ which are invertible by construction. However, the number of raw data transmissions is relatively high, i.e., 1-hop for odd nodes and 2-hops for even nodes. We address this inefficiency in the next section.

3.4 Unidirectional Haar-like Wavelets

We now construct a transform that addresses the inefficiency of the transform proposed in Section 3.3.5. For the transform in Section 3.3.5, raw data from even and odd nodes must be forwarded over 2-hops and 1-hop, respectively. This can be inefficient in terms of transport costs. Instead, it would be better to construct a lifting transform that directly minimizes the number of raw data transmissions each node must make. We use the splitting method in Section 3.3.5. Note that some form of data exchange must occur before the transform can be computed, i.e., evens must transmit raw data to odds, or vice versa. Suppose that even nodes forward raw data to their parents. In this case, the best we can do is to design a transform for which even nodes transmit raw data over only 1-hop, and odd nodes do not transmit any raw data. This will minimize the number of raw data transmissions that nodes need to make, leading to transforms which are more efficient than the 5/3-like transform in terms of transport costs. We note that minimizing raw data only serves as a simple proxy for the optimization. A more formal optimization which relies on this same intuition is undertaken in recent work [37].

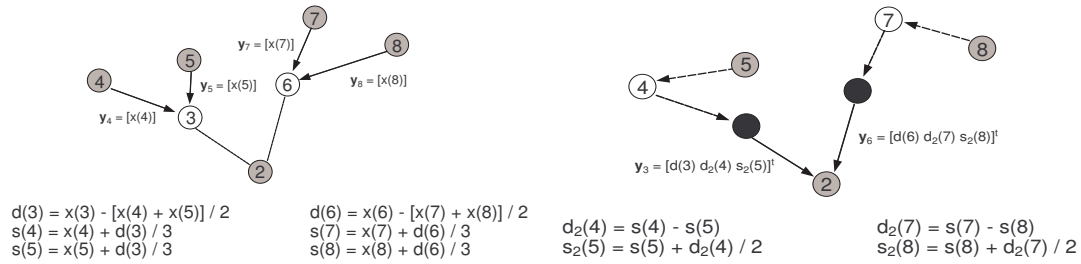
3.4.1 Transform Construction

A design that is more efficient than the 5/3-like transform can be achieved as follows. Note that an odd node n has data from its children $\mathcal{C}_n$ and/or even broadcast neighbors $\mathcal{B}_n \cap \mathcal{E}$ locally available, so it can directly compute a detail coefficient for itself, i.e., $d(n) = x(n) - \sum_{i \in \mathcal{C}_n} \mathbf{p}_n(i)x(i) - \sum_{j \in \mathcal{B}_n \cap \mathcal{E}} \mathbf{p}_n(j)x(j)$. Thus, the detail $d(n)$ is computed directly at n, is encoded, and then is transmitted to the sink. These details require fewer bits for encoding than raw data, hence, this reduces the number of bits that odd nodes must transmit for their own data. Since data

from even node m is only used to predict data at its parent $\rho(m)$, we simply have that $\mathcal{N}_m = \{\rho(m)\}$ and $s(m) = x(m) + \mathbf{u}_m(\rho(m))d(\rho(m))$. Moreover, these smooth coefficients can be computed at each odd node n. Therefore, even nodes only need to forward raw data over one hop, after which their smooth coefficients can be computed. Note that not all odd nodes will have children or even broadcast neighbors, i.e., there may exist some odd nodes n such that $\mathcal{C}_n = \emptyset$ and $\mathcal{B}_n \cap \mathcal{E} = \emptyset$. Such odd nodes can simply forward raw data $x(n)$ to their parent $\rho(n)$, then $\rho(n)$ can compute their details as $d(n) = x(n) - \mathbf{p}_n(\rho(n))x(\rho(n))$. Thus, there may be a few odd nodes that must send raw data forward one hop. This leads to a *Haar-like transform* since it is the Haar wavelet transform when applied to 1D paths.

Odd nodes can also perform additional levels of decomposition on the smooth coefficients of their descendants. In particular, every odd node n will locally compute the smooth coefficients of its children. Therefore, it can organize the smooth coefficients $\{s(k)\}_{k\in\mathcal{C}_n}$ onto another tree T_n^2 and perform more levels of transform decomposition along T_n^2. In this work, we assume T_n^2 is a minimum spanning tree. This produces detail coefficients $\{d_2(k)\}_{k\in\mathcal{O}_n^2}$, $\{d_3(k)\}_{k\in\mathcal{O}_n^3}$, ..., $\{d_{J+1}(k)\}_{k\in\mathcal{O}_n^{J+1}}$ and smooth coefficients $\{s_{J+1}(k)\}_{k\in\mathcal{E}_n^{J+1}}$ for some $J \geq 0$. In this way, odd nodes can further decorrelate the data of their children before they even transmit. This reduces the resources they consume in transmitting data. An example of this separable transform for $J = 1$ is illustrated in Figure 3.7. By choosing averaging prediction filters and the orthogonalizing update filter design in Section 2.4.2 [56], we get the global equation in (3.18). The coefficient vector $\mathbf{y}_6$ is obtained in a similar manner.

$$\mathbf{y}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(3) \\ x(4) \\ x(5) \end{bmatrix} \quad (3.18)$$



(a) 1^{st}-level along T (b) 2^{nd}-level "orthogonal" to T

Figure 3.7: Unidirectional Computations for Haar-like Transform. In (a), nodes 3 and 6 compute a first level of transform. Then in (b), nodes 3 and 6 compute a second level of transform on smooth coefficients of their children.

3.4.2 Discussion

The transform computations that each node performs can be easily mapped into our standard form $\mathbf{y}_n = [\mathbf{A}_n \; \mathbf{B}_n] \cdot \left[x(n) \; \mathbf{y}_{\mathcal{D}_n}^t \; \mathbf{y}_{\mathcal{B}_n}^t\right]^t$ by appropriately populating the matrices in (3.16) and (3.17). Therefore, they will always yield invertible transforms. For example, since each odd node n predicts its own data $x(n)$ using data from its children $\mathcal{C}_n$ and even broadcast neighbors $\mathcal{B}_n \cap \mathcal{E}$, then updates the data of its children from its own detail, a single level transform can be expressed as

$$\mathbf{y}_n = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{u}_{\mathcal{D}_n}(n) & \mathbf{I} \end{bmatrix} \begin{bmatrix} 1 & -\mathbf{p}_n(\mathcal{D}_n) & -\mathbf{p}_n(\bar{\mathcal{B}}_n) \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x(n) \\ \mathbf{y}_{\mathcal{D}_n} \\ \mathbf{y}_{\mathcal{B}_n} \end{bmatrix}. \tag{3.19}$$

By choosing

$$\mathbf{A}_n = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{u}_{\mathcal{D}_n}(n) & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} 1 & -\mathbf{p}_n(\mathcal{D}_n) \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{3.20}$$

and

$$\mathbf{B}_n = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{u}_{\mathcal{D}_n}(n) & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} -\mathbf{p}_n(\bar{\mathcal{B}}_n) \\ \mathbf{I} \end{bmatrix}, \tag{3.21}$$

we have that $\mathbf{y}_n = [\mathbf{A}_n\, \mathbf{B}_n] \cdot \left[x(n)\, \mathbf{y}_{\mathcal{D}_n}^t\, \mathbf{y}_{\mathcal{B}_n}^t\right]^t$. Note that (3.19) covers all of the cases discussed in Section 3.4.1 for each odd node n, that is to say: (i) $\mathcal{C}_n \neq \emptyset$ and $\mathcal{B}_n \cap \mathcal{E} \neq \emptyset$, (ii) $\mathcal{C}_n = \emptyset$ and $\mathcal{B}_n \cap \mathcal{E} \neq \emptyset$, (iii) $\mathcal{C}_n \neq \emptyset$ and $\mathcal{B}_n \cap \mathcal{E} = \emptyset$, and (iv) $\mathcal{C}_n = \emptyset$ and $\mathcal{B}_n \cap \mathcal{E} = \emptyset$. In particular, whenever $\mathcal{C}_n \neq \emptyset$, $\mathbf{p}_n(\mathcal{D}_n)$ and $\mathbf{u}_{\mathcal{D}_n}(n)$ will have some non-zero entries. Otherwise, n has no descendants and so $\mathbf{p}_n(\mathcal{D}_n)$ and $\mathbf{u}_{\mathcal{D}_n}(n)$ will just be vectors of zeros. Similarly, whenever $\mathcal{B}_n \cap \mathcal{E} \neq \emptyset$, $\mathbf{p}_n(\bar{\mathcal{B}}_n)$ will have some non-zero entries. Otherwise, n has no even broadcast neighbors and $\mathbf{p}_n(\bar{\mathcal{B}}_n)$ will be a vector of zeros.

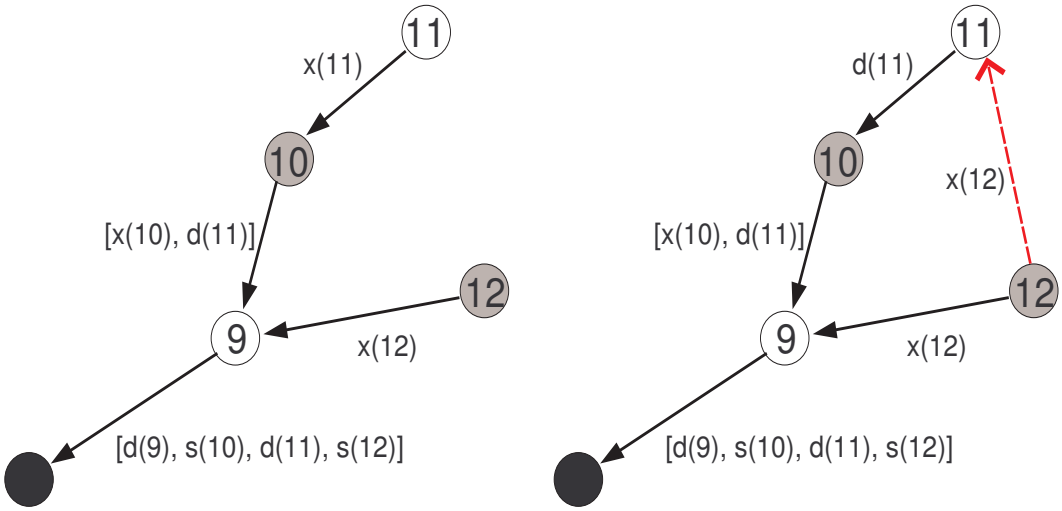
Similarly, each even node m may need to compute predictions for its odd children, so its computations for a single level transform can be expressed as

$$\mathbf{y}_m = \begin{bmatrix} 1 & \mathbf{0} \\ -\mathbf{p}_{\mathcal{D}_m}(m) & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} x(m) \\ \mathbf{y}_{\mathcal{D}_n} \end{bmatrix}. \tag{3.22}$$

Also note that (3.22) covers all of the cases for each even node m discussed in Section 3.4.1, i.e., when m has to compute predictions for children then $\mathbf{p}_{\mathcal{D}_m}(m) \neq \mathbf{0}$, otherwise, $\mathbf{p}_{\mathcal{D}_m}(m) = \mathbf{0}$.

Note that, when broadcast data is used, the decorrelation achieved at odd nodes may still be comparable to the 5/3-like transform since the same number of neighbors (or more) will be used. Moreover, broadcasts are particularly useful for odd nodes n that have no children, i.e., n for which $\mathcal{C}_n = \emptyset$ but $\mathcal{B}_n \cap \mathcal{E} \neq \emptyset$. If broadcast data is not used when it is available, node n will have to transmit $x(n)$ to its parent. Since $x(n)$ requires more bits for encoding than does a detail coefficient $d(n)$, n will consume more resources during data transmission. By using broadcasts, these odd nodes which have no children can still use data overheard from even broadcast neighbors, allowing them to avoid transmitting raw data to their parents. This is

illustrated in Figure 3.8, where node 11 has no children but overhears data from node 12. The example in Figure 3.8(a) will consume more resources at node 11 than will the example in Figure 3.8(b).



(a) Without Broadcasts

(b) With Broadcasts

Figure 3.8: No broadcasts are used in (a), so node 11 consumes more resources when transmitting raw data $x(11)$. Broadcasts are used in (b), so node 11 consumes less resources when transmitting detail $d(11)$.

3.5 Quantization of Transform Coefficients

Quantization is also an important part of the compression process since it allows nodes to provide data to the sink at different bit rates (and correspondingly, different costs and reconstruction qualities), though lossless coding (i.e., without quantization) could always be performed for the wavelet transforms in Section 3.3.4, 3.3.5 and 3.4 or the T-DPCM schemes in Section 3.3.3. In this thesis we consider both lossless and lossy coding. One major problem with quantization in WSNs is that original data will not always be available for computation at every node. Thus, it is possible that some nodes receive data that has already been quantized, process it,

then quantize it again. This creates two major problems. First, it can lead to severe propagation of quantization errors in the inverse transform computations, leading to significant degradation in the reconstructed data. Secondly, if the encoders (i.e., at the nodes) and decoder (i.e., at the sink) operate on different data (unquantized versus quantized) for adaptation of the filters or entropy coders, this will lead to a serious drift problem.

The first problem can be easily handled by only using quantized data for filtering operations at each node, that way we can avoid having cascaded quantization steps. T-DPCM is one scheme that only uses quantized data to compute predictions. In light of these facts, for the lifting transforms we also find it sensible to (i) only use quantized data to compute predictions for odd node data and (ii) only use quantized detail coefficients to compute updates of even node data. A similar strategy would be employed over multiple levels of decomposition. In this way we can avoid this severe propagation of quantization errors. This idea was initially proposed in [8] for the same reason, and was used to quantize the coefficients of a unidirectional 1D, 5/3 wavelet transform. Note that transforms such as the T-KLT do not have this luxury since inverse and forward transforms are constantly being applied at every node, i.e., initially each node transmits quantized raw data, the quantized raw data is used to generate transform coefficients, then the coefficients are quantized again. Thus, propagation of quantization error is inevitable for the T-KLT.

The second problem can be dealt with by using only quantized data in the filter adaptation. For example, in the adaptive prediction filter scheme described in Section 2.3.2, the prediction filters are always updated using quantized data. In this way, nodes and the sink will use the same data for adaptation, so drift problems are completely avoided. Obviously quantized data should always be used when adapting the entropy coders.

3.6 Experimental Results

This section presents experimental results that compare the transforms proposed here against existing methods. Source code used to generate these results can be found on our web page[5]. In particular, we focus on comparing the proposed multi-level Haar-like lifting transforms against the multi-level 5/3-like transform from [55, 58], the T-DPCM scheme in [52] and raw data gathering. We consider the application of distributed data gathering in WSNs. Performance is measured by total energy consumption.

3.6.1 Experimental Setup

For evaluation, we consider simulated data generated from a second order AR model. This data consists of two 600×600 2D processes generated by a second order AR model with low and high spatial data correlation, e.g., nodes that are a certain distance away have higher inter-node correlation for the high correlation data than for the low correlation data. More specifically, we use the second order AR filter $H(z) = \frac{1}{(1-\rho e^{j\omega_0} z^{-1})(1-\rho e^{-j\omega_0} z^{-1})}$, with $\rho = 0.99$ and $\omega_0 = 99$ (resp. $\omega_0 = 359$) to produce data with low (resp. high) spatial correlation. The nodes were placed in a 600×600 grid, with node measurements corresponding to the data value from the associated position in the grid. Each network used in our simulations is generated from a set of random node positions distributed in the 600×600 grid. An SPT is constructed for each set of node positions. We consider two types of networks: (i) *variable radio range* networks in which each node can have a different radio range, and (ii) *fixed radio range* networks in which each node has the same radio range. In the variable radio range case, the radio range that each node n uses for transmission

[5] http://biron.usc.edu/wiki/index.php/Wavelets_on_Trees

is defined by the distance from n to its parent in the SPT. Additional broadcast links induced by the SPT are also included, i.e., a broadcast link between node n and m exists if m is not a neighbor of n in the SPT but is within radio range of n.

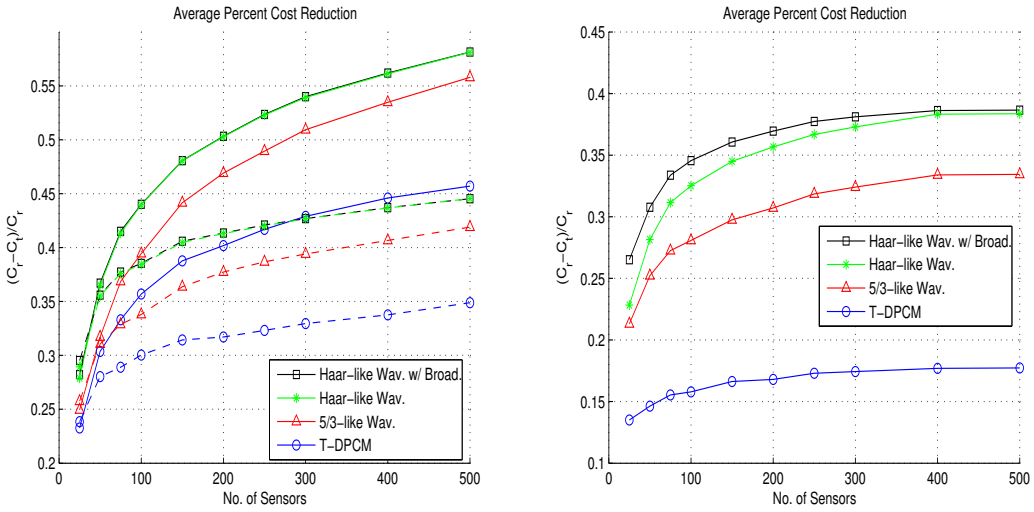
In order to measure energy consumption, we use the cost model for WSN devices proposed in [21, 74], where the energy consumed in transmitting k bits over a distance D is $E_T(k, D) = E_{elec} \cdot k + \varepsilon_{amp} \cdot k \cdot D^2$ Joules and the energy consumed in receiving k bits is $E_R(k) = E_{elec} \cdot k$ Joules. The $E_{elec} \cdot k$ terms capture the energy dissipated by the radio electronics to process k bits. The $\varepsilon_{amp} \cdot k \cdot D^2$ term captures the additional energy for signal amplification needed to ensure reasonable signal power at the receiver. WSN devices also consume energy when performing computations, but these costs are typically very small compared with transmission and reception costs. Therefore, we ignore them in our cost computations. Also note that all data gathering schemes will suffer from channel noise and attenuation, so a no-channel-loss comparison is still valid. Thus, we do not consider these effects in our experiments.

Comparisons are made with the Haar-like transforms of Section 3.4 against the 5/3-like transform with delayed processing proposed in [58] and the T-DPCM scheme proposed in [52]. Predictions for each of these transforms are made using the adaptive prediction filter design in Section 2.3.2 (proposed in [52]). Updates are made using the "orthogonalizing" update filter design in Section 2.4.2 (proposed in [56]). In each epoch, we assume that each node transmits $M = 50$ measurements taken at M different times. Also, each raw measurement is represented using $B_r = 12$ bits. We assume each odd node encodes M detail coefficients together with an adaptive arithmetic coder. Smooth coefficients are treated like raw data, e.g., each one uses B_r bits. Since we only seek to compare the performance of spatial transforms, we do not consider any temporal processing.

3.6.2 Simulation Results

In the case of lossless compression, the average cost reduction ratios taken over multiple uniformly distributed networks are shown in Figure 3.9 for high and low data correlation. These are expressed as the average of multiple values of $(C_r - C_t)/C_r$, where C_t is the cost for joint routing and transform and C_r is the cost for raw data forwarding. Results for variable radio ranges (each node has different radio range) are shown in Figure 3.9(a). Results for fixed radio ranges (each node has the same radio range) are given in Figure 3.9(b). T-DPCM does the worst overall. The 5/3-like transform provides significant improvement over the simple T-DPCM scheme. The Haar-like transforms have the highest average cost reduction ratio, or equivalently, the lowest average cost. Moreover, we note that broadcast is not very helpful (on average) when nodes have variable radio ranges (Figure 3.9(a)), but there is a significant gain when nodes use a fixed radio range (Figure 3.9(b)). This is mainly because, in the fixed radio range case, (i) there are many more opportunities for using broadcast data and (ii) each node has more broadcast neighbors. Thus, broadcast is likely to be most useful when nodes are configured with a fixed radio range.

Note that the amount of raw data forwarding needed to compute the Haar-like transform is significantly reduced compared with the 5/3-like transform. Therefore, the Haar-like transform will do better than the 5/3-like transform in terms of transport costs. Granted, the 5/3-like transform will use data from more neighbors for processing, so the decorrelation given by the 5/3-like transform will be greater than that given by the Haar-like transform. However, in our experiments the average reduction in rate that the 5/3-like transform provides over the Haar-like transform is rather small. The Haar-like transform with broadcast also provides additional



(a) Variable Radio Range

(b) Fixed Radio Range

Figure 3.9: Average percent cost reduction ($\frac{C_r - C_t}{C_r}$). Solid and dashed lines correspond to high and low spatial data correlation, respectively. Best performance achieved by Haar-like transforms, followed by 5/3-like transform and T-DPCM. High correlation data also gives greater reduction than low correlation data.

cost reduction over the Haar-like transform without broadcasts since less raw data forwarding is needed on average. Moreover, the amount of cost reduction achievable is higher for the high correlation data than for the low correlation data.

Lossy coding is also possible and can provide even greater cost reductions while introducing some reconstruction error. In this case, we quantize transform coefficients with a dead-zone uniform scalar quantizer. Performance is measured by the trade-off between total cost and distortion in the reconstructed data, which we express as the signal to quantization noise ratio (SNR). Sample 50 node networks are shown in Figs. 3.10(a) and 3.10(c) and, in the case of high correlation data, the corresponding performance curves are shown in Figs. 3.10(b) and 3.10(d). The Haar-like transforms do the best among all transforms.

When using broadcasts with the Haar-like transform, there is an additional 1 dB (resp. 2.5 dB) gain in SNR for the variable (resp. fixed) radio range network at a

fixed cost, i.e., by using broadcasts we can increase the quality in the reconstructed data for a fixed communication cost. Thus, for these networks, using broadcast is quite helpful. Also note that there are only 2 broadcast links used in the transform for the variable radio range network (Figure 3.10(a)), whereas there are over 10 broadcast links used in the fixed radio range network (Figure 3.10(c)). Thus, broadcast provides even greater gains for the fixed radio range network (2.5 dB versus 1dB) since there are more broadcast links. More generally, broadcast should provide more gains in networks where many broadcast opportunities are available.

In this particular network for the variable radio range case, T-DPCM actually does better than the 5/3-like transform. Note that in T-DPCM, only the leaf nodes forward raw data to the sink; so if there are only a few leaf nodes, the raw data forwarding cost for T-DPCM may not be very high compared with the raw data forwarding cost for the 5/3-like transform. In this particular network, only 19 of the 50 nodes are leaves in the tree. Therefore, the raw data forwarding cost for T-DPCM in this case is lower than that for the 5/3-like transform. However, on average the raw data forwarding cost for T-DPCM will be very high (see Figure 3.9), leading to higher total cost on average as compared with the 5/3-like transform.

3.6.3 Comparison of Filter and Even/Odd Split Designs

This section provides experimental results which compare the various split and filter designs. We again consider the scenario of transform-based distributed data gathering in WSN, and use the same experimental setup as in Section 3.6.1. Figure 3.11(a) shows the same 50 node network used in Section 3.6.2 and the corresponding cost-distortion curves are shown in Figure 3.11(b).

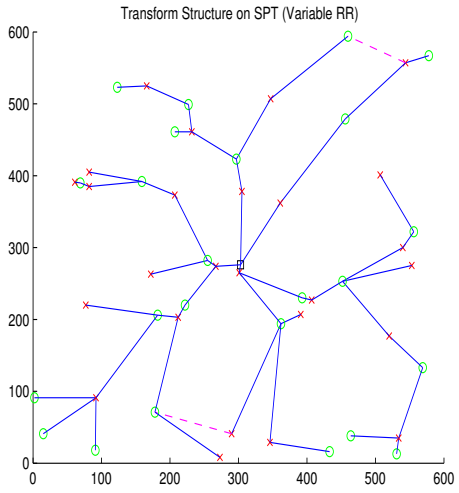
The data adaptive prediction filters from Section 2.3.2 are far superior to simple average and planar-based prediction filters proposed in Section 2.3.1. We also see that the orthogonalizing update filters from Section 2.4.2 do not provide much improvement for this network. This is mainly because not many levels of decomposition are performed. For larger networks, there is more gain when using orthogonalizing updates, but this gain is still not very substantial. In fact, for WSN and our particular transform constructions, we only observe at most 3 levels of decomposition being possible. Thus, this update design is not likely to improve the performance for this application. However, as we will see in Section 5.3, this does provide some significant improvements when used in image coding, mainly because (i) there are many pixels and (ii) many levels of decomposition can be applied.

We also show some comparisons for various split designs in this same WSN context. In particular, we compare the simple tree-based splitting described in Section 2.2.1 with Haar-like transforms from Section 3.4 against the optimized graph-based splitting discussed in 2.2.2 (see [37] for more details). We use the same network shown in Figure 3.11(a), and adaptive prediction filters are used in all cases. The resulting graph-based split is shown in Figure 3.12(a) and the corresponding cost-distortion curves are shown in Figure 3.12(b). The graph-based splits provide some improvements over the tree-based splits, mainly because there is much less raw data forwarding in the network.

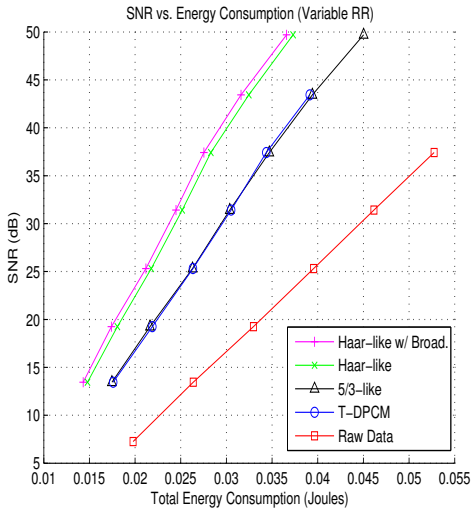
3.7 Conclusions

A general class of en-route in-network (or unidirectional) transforms has been proposed along with a set of conditions for their invertibility. This covers a wide

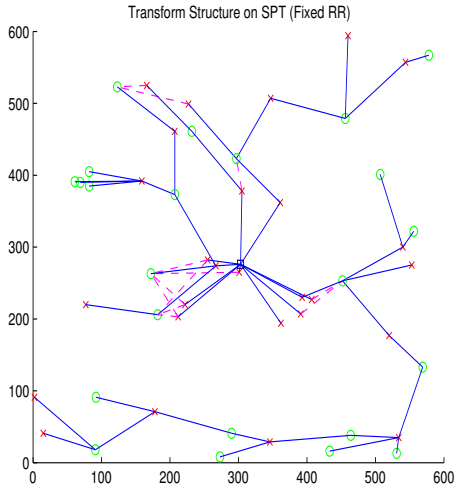
range of existing unidirectional transforms and has also led to new transform designs which outperform the existing transforms in the context of data gathering in wireless sensor networks. In particular, we have used the proposed framework to provide a general class of invertible unidirectional wavelet transforms constructed using lifting. These general wavelet transforms can also take into account broadcast data without affecting invertibility. A unidirectional Haar-like transform was also proposed which significantly reduces the amount of raw data transmissions that nodes need to make. Since raw data requires many more bits than encoded transform coefficients, this leads to a significant reduction in the total cost. Moreover, our proposed framework allows us to easily incorporate broadcasts into the Haar-like transforms without affecting invertibility. This use of broadcast data provides further performance improvements for certain networks.



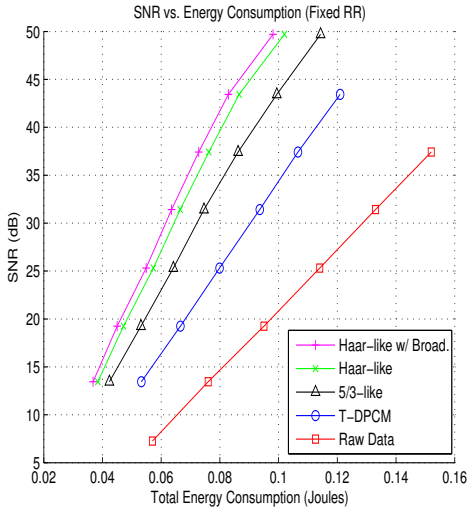
(a) Variable RR Network



(b) Cost-Distortion Curves (Variable RR)

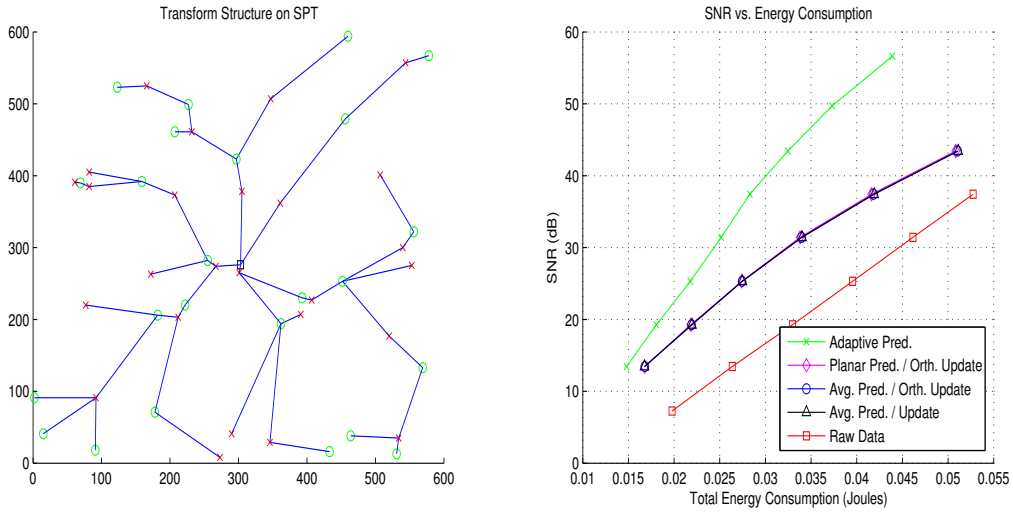


(c) Fixed RR Network



(d) Cost-Distortion Curves (Fixed RR)

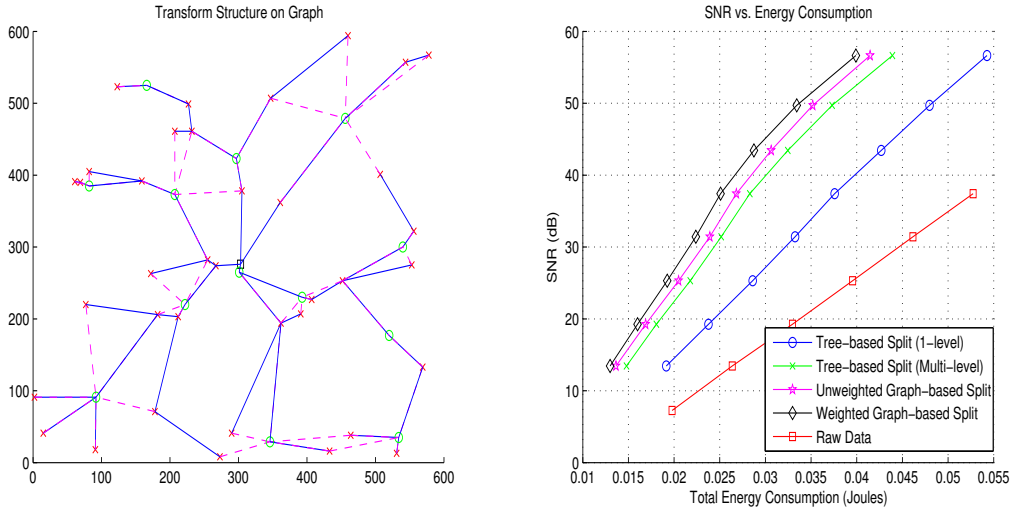
Figure 3.10: Sample networks with corresponding Cost-Distortion curves. In (a) and (c), solid lines denote forwarding links, dashed lines are broadcast links, circles are even nodes, x's are odd nodes, and the square center node is the sink.



(a) Sample Network

(b) Performance Curves

Figure 3.11: Filter design comparison. Circles are even nodes and x's are odd nodes. Adaptive prediction filters do much better than fixed prediction filters. Orthogonalizing updates provide almost no gain.



(a) Graph-based Split

(b) Performance Curves

Figure 3.12: Split design comparison. Circles are even nodes and x's are odd nodes. Dashed lines denote broadcast links. Graph-based splits provide some improvements over tree-based splits.

Chapter 4

Joint Optimization of Transform and Routing

In this chapter we address the problem of joint optimization of transform and routing. Note that the unidirectional transforms proposed in Chapter 3 can be defined along any routing tree, thus, it would be best to choose the routing tree jointly with the transform. We achieve this by proposing two optimization algorithms (initially proposed by us in [54]) that search for the best choice of tree for a fixed choice of transform.

4.1 Introduction

While existing transform-based methods (e.g., those proposed in Chapter 3) are capable of reducing the number of bits to be transferred to the sink, almost all of them separate transform design and routing, i.e., they define transforms first then map those transforms onto efficient routing trees, or vice versa. In the first case, this requires nodes to transmit uncompressed data directly to a cluster head as in [15] or to a certain number of neighbors [72, 73] before transform coefficients can even be computed. If the neighbors (or cluster head) of a node are further away from the sink than the node itself, additional backward transmissions of uncompressed data

will be required that increase the total cost. The unidirectional transforms proposed in Chapter 3 (and [1, 9, 52, 55, 57, 58]) i) can be computed on *arbitrary routing trees* and ii) do not require additional backward transmissions (they are computed in a unidirectional manner as data flows toward the sink). These unidirectional transforms have been shown to be more energy-efficient than the bi-directional transform in [72, 73]. However, note that these unidirectional transforms consider the design of the transform and routing separately, e.g., a shortest path routing tree is chosen first and then a transform is performed over that tree. Instead, the techniques we propose in this chapter attempt to exploit the inherent interaction between different trees and the unidirectional transforms proposed in Chapter 3. This leads us to a practical approach for jointly optimizing compression and routing, i.e., we can aim at designing a tree with good transport cost *and* data correlation properties, knowing that no matter what tree is chosen a unidirectional transform can be implemented.

A shortest path routing tree (SPT), guarantees that the path from a given node to the sink is most efficient for routing, but obviously does not guarantee that consecutive nodes in a path contain highly correlated data. Of course one could simply optimize the transform along this SPT, as was done in work by us in [37]. However, that optimization problem is quite different since (i) the routing is assumed fixed, (ii) any additional broadcast links that arise are added to the links on the tree in order to form a more general graph and (iii) a greedy heuristic is used to search for the minimum cost even / odd splitting on this graph. This leads to a more general transform than the tree-based wavelet transforms proposed in Sections 3.3.4, 3.3.5 and 3.4. On the other hand, the optimization algorithm proposed in this chapter assumes that a tree-based transform is given (such as, e.g., the Haar-like tree-based wavelet transform proposed in Section 3.4), then searches

for the routing tree that minimizes the total energy consumption for this choice of transform.

Again, note that the SPT does not guarantee that consecutive nodes along a routing path contain highly correlated data. For example, if data correlation is inversely proportional to distance between nodes, one would always have to route through the nearest neighbor in order to achieve maximal inter-node data correlation. Clearly SPT routing does not guarantee this, since this design aims to minimize distance to sink, not inter-node distance. The results in [40] corroborate this basic intuition, where it was shown that a network with high data correlation benefits most from routing with compression along shorter hops with longer overall paths. As an alternative, we could consider trees that link together nodes with high inter-node data correlation. Such trees can provide greater compression efficiency than an SPT. However, aggregating along these types of trees may force nodes to transmit data away from the sink, so that gains provided by the increase in de-correlation are offset by increased transmission cost. Since aggregation will occur along routing trees, there is a trade-off between trees that result in energy-efficient routing and ones that allow a transform to de-correlate data effectively. Thus, the main goal in this chapter is to find trees that effectively exploit this trade-off.

In order to achieve jointly optimized routing and transform we search exhaustively for the lowest cost tree among a set of possible trees, for a fixed distortion level $\mathcal{D}$. For a given tree T, we use cost model proposed in Section 3.6 which specifies the cost for each node n to route and compress data to the sink along T. We let $C_T(\mathcal{D})$ denote the total cost to route and compress data along tree T for some fixed distortion level $\mathcal{D}$. Since any of the transforms in Chapter 3 can be computed along arbitrary trees, a natural optimization problem is to find a tree T that minimizes the total cost $C_T(\mathcal{D})$ for a fixed distortion level $\mathcal{D}$ and fixed transform.

While one could consider the full set of possible trees for a given communication graph, this set can be extremely large. The well-known matrix-tree theorem [25] (which provides the number of spanning trees for a given graph) shows that a complete graph with n nodes has n^{n-2} possible trees. Even if the graph is not complete, the matrix-tree theorem may still imply a very large solution space. Thus, it is not computationally feasible to consider a full solution set. To make the optimization problem tractable, we choose only to explore trees that can be obtained by combining links from an SPT computed with edges defined by physical inter-node distances (to minimize distance to the sink) with links from a minimum spanning tree (MST) computed with edge weights defined by inter-node data correlation (to maximize pair-wise inter-node correlation). More specifically, we design an MST using edge weights $w(m,n) = 1 - r_{m,n}$ with $r_{m,n}$ the correlation coefficient between nodes m and n so that an MST corresponding to these edge weights will have a link between each node n and the neighbor of n that has maximum inter-node data correlation with n. Clearly, such an MST is "best" in the sense of maximizing pair-wise data correlation along the tree, which should help achieve improved compression efficiency for our transform. Since the SPT will minimize the cost to route any amount of data from a node to the sink, we can use combinations of such an MST with an SPT to provide a direct trade-off between high compression performance and low routing cost.

To illustrate this point, consider the real network in Figure 4.1 taken from [38], where a combination of an SPT and MST is used for joint routing and compression. The SPT provides the shortest route to the sink from any node, but fails to link some nodes to their closest neighbors. This can reduce compression efficiency. The MST links those nodes to their closest neighbors, but also has some longer paths that push data away from the sink. Clearly, neither alone is sufficient to achieve

the best joint routing and compression performance. Instead, the trees obtained by our proposed optimization methods tend to link nodes to their closest neighbor, but in a way that preserves short paths to the sink, resulting in improved overall performance.

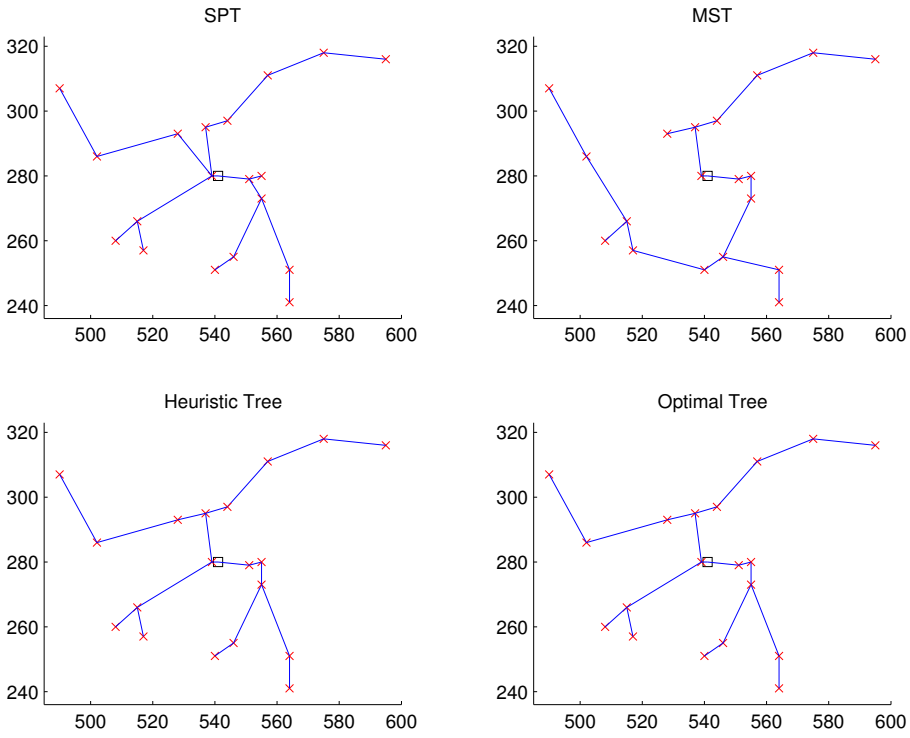


Figure 4.1: SPT, MST, and Combined Tree

In summary, in this chapter we address the joint optimization of routing and compression by using the unidirectional transform framework proposed in Chapter 3. In particular, we propose a two heuristic algorithms for selecting routing and transform jointly by accounting for both data correlation and routing costs.

4.2 Joint Routing and Transform Optimization

Our proposed optimization method is inspired by the "foreign coding" technique developed in [48], where an MST is constructed with edge weights that are a function of data correlation and where data is encoded along this MST and forwarded along an SPT. Note that an MST constructed from edge weights that are inversely proportional to pair-wise data correlation will always connect nodes to the neighbors with which they share the highest data correlation. To see why this is so, consider Prim's algorithm [12] for constructing an MST. Given a set of edge weights, Prim's algorithm can construct an MST by starting from any arbitrary node n. So by starting from any node n, the second node added to the MST will be the node m such that $w(n,m) \leq w(n,k)$, for all $k \in V \backslash \{n\}$. Since $w(n,m)$ is minimal over all edges containing n as a vertex, the correlation between n and m is maximal. Therefore, data at n is maximally correlated with data at m. Since this can be done for arbitrary n, it follows that in an MST every node will be connected to the neighbor with which it shares the highest data correlation. It is worth noting here that an MST defined in this way only considers pair-wise correlation, and so is not necessarily optimal from a coding standpoint when using our proposed transforms (where data is filtered over multiple hops). However, it should provide a good approximation to the optimal tree. Thus, we choose to exploit our stated trade-off by searching for a minimum cost tree among a set of trees that combine a distance-based SPT and a correlation-based MST.

In the technique proposed in [48], spatial correlation at each node is only exploited using data from at most one child in the routing tree. If data from a child is used, nodes perform what is called "foreign coding". Otherwise (if no other data is used for compression), nodes perform what is known as "self coding". In the case of

the unidirectional transforms proposed in Chapter 3, nodes always perform foreign coding. Moreover, this foreign coding can incorporate data from more than one child and/or descendant. Since the optimal solution in [48] assumes that *foreign coding uses data from only one child*, it can not be directly extended to our transforms. Furthermore, the unidirectional transforms we consider should also provide more de-correlation since a node can compress its data using data from more than one neighbor.

An MST does have some drawbacks, though. For one, it may not have as many merge points as an SPT. Since the transforms in Chapter 3 only exploit cross-path correlation at and around merge nodes, having fewer merges may actually reduce the efficiency of our transform when performed along an MST. However, an appropriate combination with an SPT should maintain these merges whenever beneficial. In fact, not all of the neighbors of a node in the MST will have high data correlation with it so some merges may actually hurt coding performance. As mentioned above, our MSTs only consider correlation over a single hop and may result in some inefficiency since our proposed transform actually filters data over multiple hops. Furthermore (as will be discussed in Section 4.5), if a predict has more neighbors it will tend to have less residual energy and so should require fewer bits. Similarly, having more neighbors at an update node can produce a smoother approximation of the original data and as such should also require fewer bits. So as an alternative to MSTs, we could develop trees that (1) preserve beneficial merges and (2) keep the number of merges at predict nodes to a minimum. These issues will be explored experimentally in Section 4.5 and could be an interesting area for future work.

In Section 4.2.1, we propose an algorithm that finds the minimum cost combination of an SPT and MST by computing, for every possible combination, the cost

of transform and routing along each tree (overlayed on an SPT) and then selecting the lowest cost combination for a fixed distortion $\mathcal{D}$. The algorithm is general enough to accommodate an arbitrary definition of edge weights used to construct the MST, i.e., $w(m,n) = 1 - r_{m,n}$, or $w(m,n)$ can be physical inter-node distance, or anything else that allows us to quantify the degree of inter-node data correlation. Since the number of combinations grows rapidly with the number of nodes, we also propose a heuristic approximation algorithm that is amenable to larger networks in Section 4.3.

4.2.1 Optimization Algorithm

For a set of N nodes, let T_S denote the SPT and T_M denote an oriented version of the MST. Let T represent the tree which is our desired combination of T_S and T_M. An oriented version of the MST (T_M) is necessary to define a unidirectional transform. Basically, T_M fixes the sink node $N+1$ as the root and directs all edges in the MST toward the sink. We can represent each tree by defining parent functions ρ_n^M and ρ_n^S for T_M and T_S respectively. Under this construction, data at node n is routed to the sink through ρ_n^M in T_M, ρ_n^S in T_S, and ρ_n in T. Thus, we define the edges in each tree by the ordered pairs (n, ρ_n^M) and (n, ρ_n^S) for T_M and T_S.

We construct a minimum cost tree by searching among all feasible combinations of such edges in T_M with such edges in T_S. We first explain how to find the smallest possible set of feasible combinations (i.e., combinations that result in a connected acyclic graph) in Section 4.2.2. We then provide an algorithm that searches over this feasible set to find a minimum cost solution in Section 4.2.3.

4.2.2 Feasible Set Construction

The total number of combinations could be as many as 2^N, but many such edges in T_S and T_M will be the same so we may eliminate those from consideration. Furthermore, not all combinations of such edges will produce a valid tree (i.e., some may result in cycles or may disconnect certain groups of nodes) so the number of combinations can be reduced even further by eliminating invalid trees. We consider an edge (n, m) to be the same in both trees if $m = \rho_n^M = \rho_n^S$ (i.e., the parent of node n is the same in both trees). Thus, we define $V' = \{n | \rho_n^M \neq \rho_n^S\}$ and N' as the number of nodes in V'. We also enumerate this set as $V' = \{n_1, n_2, \ldots, n_{N'}\}$. For each node $n_i \in V'$, let $E'(n_i) = \{(n_i, \rho_{n_i}^M), (n_i, \rho_{n_i}^S)\}$ be the set of edges from n_i to the parent of n_i in either T_S or T_M. Then the full set of combinations of edges we consider in T_M and T_S is given by:

$$\mathcal{E} = E'(n_1) \times E'(n_2) \times \ldots \times E'(n_{N'}).$$

We reduce the search space further by eliminating combinations of edges in $\mathcal{E}$ that do not produce a valid tree (i.e., graphs that are disconnected or have cycles or both).

We check for tree validity as follows. Let $\tilde{\mathcal{E}}_j \in \mathcal{E}$, where j indexes the j-th combination of edges in $\mathcal{E}$. Naturally, $\tilde{\mathcal{E}}_j = \{(n_1, m_{1,j}), \ldots, (n_1, m_{N',j})\}$ for the j-th combination. Let $\tilde{E}$ be the set of edges in T_S that are the same in T_M. Then a combination $\mathcal{E}_j$ will be feasible only if the graph $\tilde{T} = (V, \tilde{E} \cup \tilde{\mathcal{E}}_j)$ is connected and acyclic. This is done by checking that each leaf node has a non-cyclic path to the sink (which is sufficient since this process traverses every node in the network). Otherwise, the graph $\tilde{T}$ does not form a valid tree. We represent the set of feasible

trees by the $N_f \times N$ matrix $\mathbf{T}_f$, where N_f is the number of feasible trees and $\mathbf{T}_f(m, n)$ is the parent of node n in the m-th feasible tree.

4.2.3 Feasible Set Search

Since the full set of feasible trees is given by $\mathbf{T}_f$, we could then find the tree that optimizes routing and transform by i) fixing a target distortion level $\mathcal{D}$ (in our case, distortion is mean squared error (MSE)) , ii) computing the cost C_j for performing routing and transform along every possible tree given in $\mathbf{T}_f$ with distortion level $\mathcal{D}$, and iii) choosing the tree with minimum cost. This is an exhaustive search over our set of feasible combinations of MST and SPT, and should therefore provide the minimum cost combination.

Specifically, this is done as follows. Let C^* be the cost for the best tree found up to row j and initialize it as $C^* = \infty$. Also let i^* index the row in $\mathbf{T}_f$ corresponding to C^* and initialize it as $i^* = 0$. Then for each row j of $\mathbf{T}_f$, with $j = 1, 2, \ldots, N_f$, do the following. Define the parent function $\rho_j(1 : N) = \mathbf{T}_f(j, 1 : N)$ and compute C_j = ComputeCost$(\rho_j, \mathcal{D}, T_S)$ (a function that computes the cost of doing transform and routing along the tree corresponding to ρ_j with SPT T_S overlayed on top). If $C_j < C^*$, then $C^* = C_j$ and $i^* = j$. Once all feasible trees are exhausted, we can construct the tree T (which minimizes the cost for routing and transform over all feasible combinations of MST and SPT) using the parent function defined by $\mathbf{T}_f(i^*)$. The function ComputeCost$(\rho_j, \mathcal{D}, T_S)$ returns the total cost for using the tree defined by ρ_j. This cost is computed using the cost models discussed in Section 3.6.

4.3 Heuristic Approximation Algorithm

For large N, the feasible set from Section 4.2.2 can still be very large. This makes the problem intractable for large N, which motivates the need for a good heuristic algorithm that approximates the minimum cost algorithm in Section 4.2.1.

The main goal of a good heuristic should be to choose links that provide a direct gain in coding efficiency only if the resulting increase in routing cost does not offset the gains achieved, so that a desirable balance of low cost routing and higher compression efficiency can be obtained. This can be done reasonably well by starting from an initial tree and searching one node at a time from nodes of greatest depth (since these nodes will be further from the sink and will benefit more from efficient coding) and decrementing depth at each stage until all nodes are covered. In our case we choose SPT as the initial tree in order to preserve low routing costs, then for each node we simply determine if the cost (C_T) to use the next hop in the MST is lower than the cost to continue along the next hop in the current tree (e.g. SPT). If so, then the next hop of such a node will be the next hop along the MST (rather than the next hop along the SPT). This ensures that, for each node, any direct gains in coding efficiency will not be offset by the resulting increase in routing cost. This is clearly a greedy algorithm, and so can not guarantee that the optimal combination of an MST and SPT will be found. But at the very least, it will guarantee that the resulting tree provides lower cost transform and routing than a transform performed along the SPT.

This algorithm is described formally in Algorithm 1. The final tree we seek is T and initially $T = T_S$. This allows us to greedily choose an edge in T_M over an edge in T_S only if the direct gain in coding efficiency offsets the increase in routing cost. Naturally, the validity of the tree that results from switching to an edge in T_M is

checked before further steps are taken. We also say that a parent function ρ_j yields a feasible tree if the tree defined by ρ_j is a connected, acyclic graph. The algorithm simply searches each resulting tree and returns the lowest cost tree it finds as T.

Algorithm 1 Find Heuristic Tree

```
1: T = T_S and ρ_j = ρ_j^S, ∀j ∈ I
2: k = max(depth) and C = ∞
3: while k ≥ 1 do
4:    I_k = {m ∈ I : depth(m) = k}
5:    for each n ∈ I_k do
6:       ρ_n^t = ρ_n^M and ρ_j^t = ρ_j, ∀j ∈ I\{n}
7:       if ρ_j^t yields a feasible tree then
8:          C_t = ComputeCost(ρ_t, D, T_S)
9:          if C_t < C then
10:            Update T and ρ_j using ρ_j^t
11:            C = C_t
12:         end if
13:      end if
14:   end for
15:   k = k − 1
16: end while
17: return T
```

4.4 Practical Considerations

There are multiple practical issues involved with joint routing and transform optimization, the most prevalent ones being (i) how to estimate the change in correlation

(and the corresponding difference in cost) for a given change in routing for each node, (ii) how to communicate the changes in routing to each node and (iii) how to perform the joint optimization in a distributed manner. The first issue could be solved with some training data, where the differences in correlation (and the corresponding differences in routing costs) for different routing choices are estimated off line. In this case, the routing optimization could also be done off line and the routing information could then be flooded to the nodes from the sink. Depending on the transform, these changes in correlation (and routing costs) could also be estimated in a distributed manner by having nodes exchange data with their immediate neighbors, then the optimal routing decisions could be made at each node.

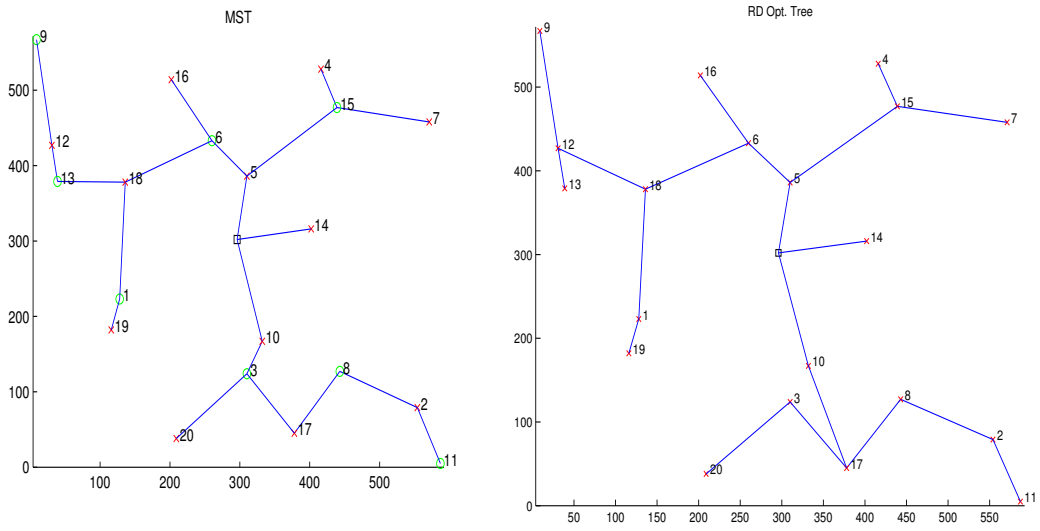
For example, in the T-DPCM scheme proposed in [41], each node computes a difference between its own data and that of its parent in the routing tree. Thus, the total cost for routing and compression for each node is equal to (i) the cost to route raw data to its parent plus (ii) the cost to route the difference between its own data and its parent's data to the sink. Since the cost for each node (cost (i) plus cost (ii)) only depends on the choice of parent, the cost for each node to jointly route and compress its own data is completely independent of any other nodes' cost. Thus, if each node chooses the minimum cost parent for itself, the total cost will also be minimized. Moreover, each node can exchange routing information (e.g., ETX) and some training data (used to estimate bit rate for prediction residuals), and then can choose the parent that minimizes cost (i) plus cost (ii) in a completely distributed manner. Note that care must be taken to ensure that routing loops do not occur (e.g., the best parent of node n is node m, and the best parent for node m is node n, etcetera), so some local consistency checking would also need to be done to ensure that no cycles exist.

For more complex transforms such as the unidirectional wavelet transforms in Chapter 3, data from multiple neighbors (both descendants and ancestors) will be used to compute the transforms. Thus, the optimization will be difficult to perform in a distributed manner in general. Moreover, if topology changes such as link or node failures occur, re-configuration will be needed no matter what transform is chosen. Therefore, the proposed optimization techniques are probably best suited to very stable, static networks in which topology changes are infrequent. These and other related practical issues are an interesting topic for future work.

4.5 Evaluation of MST Performance

As discussed in Section 4.2, the MST is not necessarily the best tree to minimize the distortion (i.e. maximize SNR) for a given rate. This is mainly due to its lack of merges (as compared to an SPT). For the sake of computational feasibility, we consider the small 20 node network shown in Figs. 4.2(a) and 4.2(b) where nodes are indexed by the number next to them. We use the unidirectional 5/3-like transform proposed in Section 3.4 for the comparisons. We compute all of the possible spanning trees using standard algorithms [70] then perform an exhaustive search of all possible spanning trees for the best RD tree. The performance curves are shown in Figure 4.3. In this case, there are only around 1000 spanning trees, so a brute-force search is feasible. However, in general there will be very many spanning trees. For each fixed rate we compute the distortion for each tree and choose the tree with the minimum distortion.

The optimal RD tree shown in Figure 4.2(b) has one more merge (node 12) than the MST shown in Figure 4.2(a) and also has a different merge with different neighbors (node 17), resulting in reconstruction quality shown in Figure 4.3. The



(a) Minimum Spanning Tree (b) RD Optimal Tree

Figure 4.2: Comparison of MST with RD optimal tree.

increase in quality is not so significant in this case, but it may be more significant as the network density grows. The fact that adding a merge improves performance is consistent with our previous discussion.

These results suggest that having more merges at a given node will generally provide better performance. This is reasonable if the data considered is spatially stationary and is highly correlated across space. As discussed before, adding more neighbors to a predict node will tend to produce smaller residual energy. Conversely, an update coefficient (low pass) will provide a smooth approximation to the original data and so adding more residues (i.e. predicts) can increase smoothness which would also reduce the number of bits. All things considered, an MST can provide a good approximation to the optimal RD tree but it is clearly not optimal, mainly because it does not have many merges and because the merges it does have may not occur in the right places in the tree. Finding trees that can eliminate the shortcomings of MSTs is an interesting area for future work.

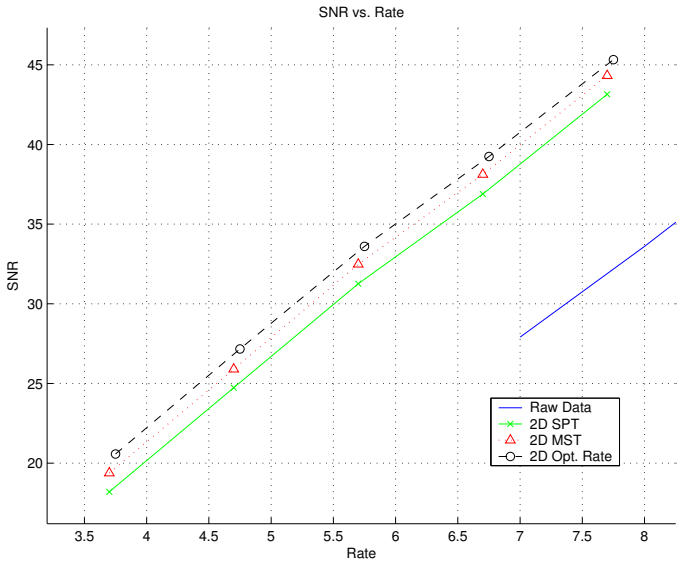
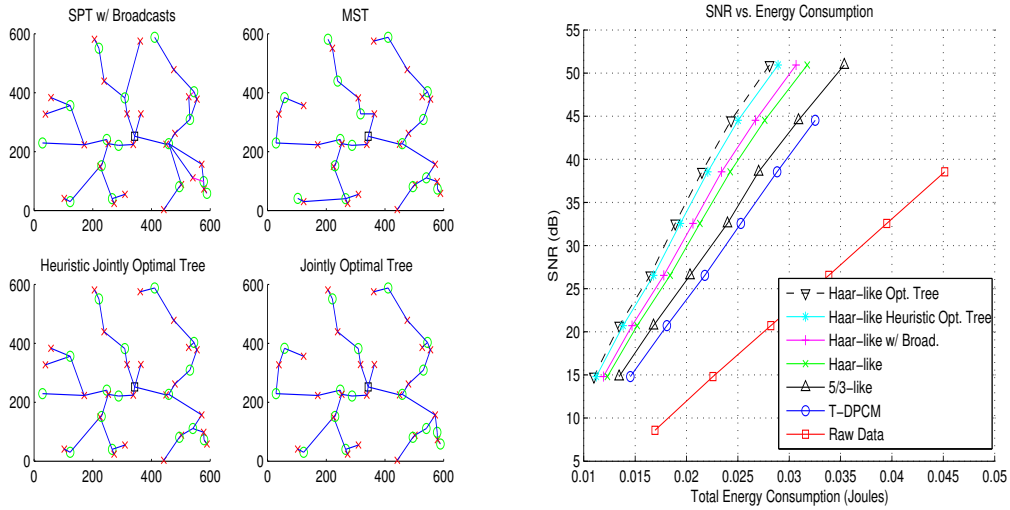


Figure 4.3: Performance Comparison of MST and RD Optimal Tree

4.6 Experimental Results

To evaluate the performance of the proposed optimization algorithm we use the sample 40 node network shown in Figure 4.4(a). We use the same set of sample data, the same cost models and the same entropy coding techniques presented in Section 3.6. We find a jointly optimized transform and routing, with the transform fixed as the Haar-like separable wavelets with and also without broadcasts. We compare the jointly optimized transforms and trees against an SPT using T-DPCM, the 5/3-like separable wavelets and the Haar-like separable wavelets (with and without broadcasts). The jointly optimized network topology is shown in Figure 4.4(a) and the performance curves are shown in Figure 4.4(b).

As we can see, the heuristic optimized tree with Haar-like transform is exactly the same as the full optimized tree, and gives a 2 dB improvement in SNR over the best SPT Haar-like transform (with broadcasts). Thus, in this case we see that the heuristic optimization algorithm presented in Section 4.3 provides essentially



(a) Sample Network

(b) Cost-Distortion Curves

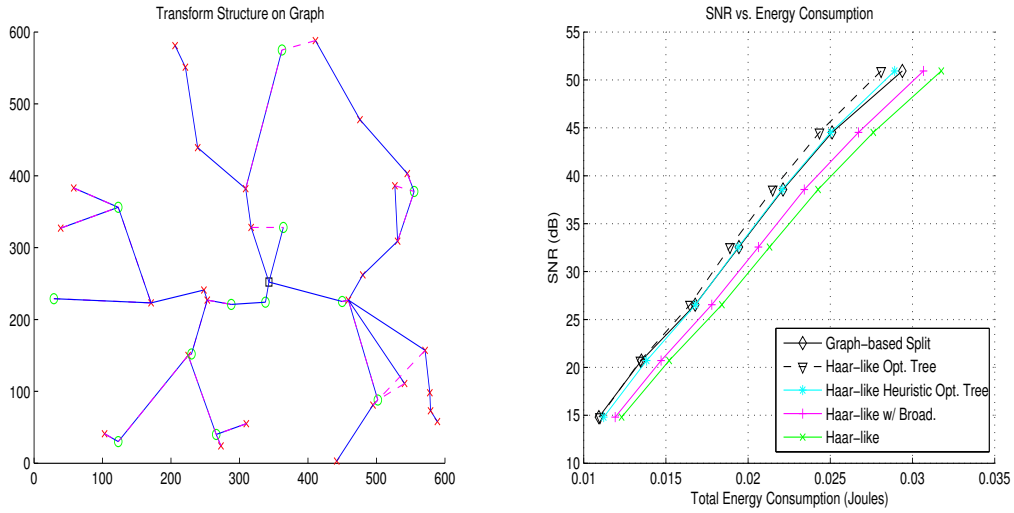
Figure 4.4: Jointly optimized network with corresponding Cost-Distortion curves. In (a), blue lines denote forwarding links, dashed magenta lines denote broadcast links, green circles represent even nodes, red x's represent odd nodes, and the black center node is the sink.

the same performance as the full optimization algorithm in Section 4.2.1. We make similar observations for other networks (which have a few thousand feasible trees) in that the heuristic and full optimized trees are almost exactly the same[1]. This provides a clear example where the total cost can be reduced using joint transform and routing optimization. Naturally, the overall gains that can be achieved will vary from network to network.

We also compare this joint routing and transform optimization algorithm against the graph-based even/odd split optimization described in Section 2.2.2 [37]. The corresponding graph-based even/odd split and the cost-distortion curves are shown in Figure 4.5. In this case, the graph-based even/odd split and the routing tree optimized using the greedy heuristic give very similar performance, while the optimized routing tree using the full search algorithm still gives the best results. We

[1]Of course it is computational infeasible to always compare the full optimization method with our proposed heuristic since the number of possible trees is generally very large.

make similar observations for other networks. As such, the joint optimization of routing and transform should typically provide performance that is competitive with the graph-based even/odd splitting optimization.



(a) Sample Network

(b) Cost-Distortion Curves

Figure 4.5: Comparison of optimized graph-based splitting and optimized routing. In (a), blue lines denote forwarding links, dashed magenta lines denote broadcast links, green circles represent even nodes, red x's represent odd nodes, and the black center node is the sink.

In order to get a better sense of what will happen on average, we also consider the cost reduction for lossless coding averaged over multiple random networks. This is shown in Figure 4.6 for the high correlation data. With routing optimization, we see an average reduction in total cost of around 4% with respect to the Haar-like transform on an SPT. This suggests that routing optimization may not provide significant performance improvements on average. This conclusion is also consistent with previous work [79], which showed that routing with compression along an SPT is nearly optimal for varying degrees of inter-node data correlation. Our results are simply a reflection of that.

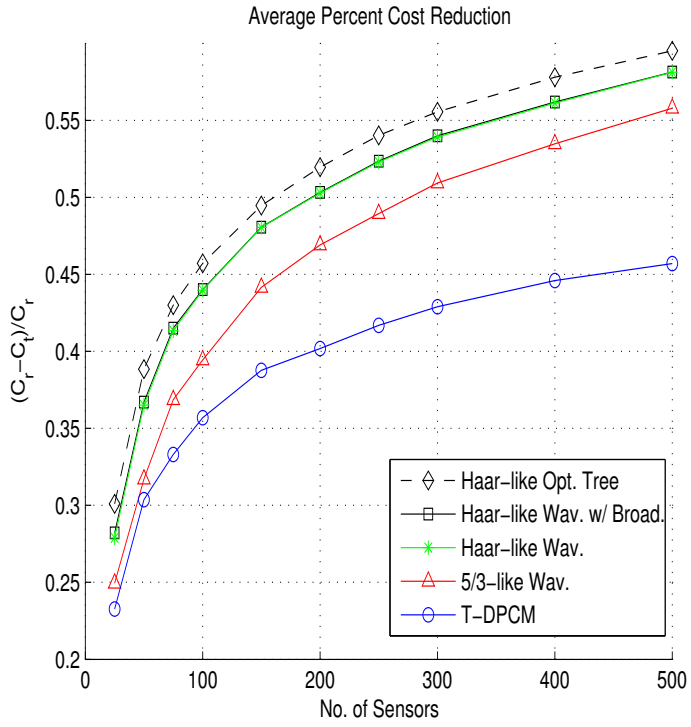


Figure 4.6: Cost reduction ratios with routing optimization.

Chapter 5

Graph-based Transforms for Image Coding

The main focus of this chapter is on developing graph-based transforms for efficient image representations. Efficient representations are typically achieved in one of two ways. The first common method is to apply a separable transform to an image in order to de-correlate data across neighboring pixels. The second method is to divide the image into blocks, to predict the pixel values in each block using pixel values from neighboring blocks, then to compute prediction residuals. Both of these techniques have been adopted in standards such as JPEG, JPEG-2000 and H.264/AVC, and tend provide very efficient representations for smooth images with simple discontinuities (or edges) such as horizontal or vertical edges. However, they do not provide efficient representations for image regions with more complex edge structure. In particular, they tend to produce many large magnitude coefficients in regions with more complex edges, and these require many bits to be encoded. Moreover, quantization of these large magnitude coefficients lead to annoying artifacts near edges in the reconstructed images, e.g., the well-known ringing artifacts. Note that filtering across edges in the image is the main source of inefficiency in both of these schemes. Thus, the main goal of this chapter is to develop graph-based transforms that avoid filtering across edges. This should reduce the number

of large magnitude coefficients, hence reducing the total bit rate needed to represent the transform coefficients. This should also better preserve the edge structure in the reconstructed images. To this end, we make two contributions, summarized as follows.

5.1 Overview

The first part of this chapter discusses a separable, edge-adaptive, tree-based lifting transform. Note that in all of the existing image and video coding standards, a *standard separable transform* (i.e., a transform that consists of row-wise filtering followed by column-wise filtering, or vice versa) is used to de-correlate data across neighboring image pixels. These separable transforms can efficiently represent smooth images regions with only horizontal or vertical edges, i.e., there are only a few large magnitude transform coefficients. This is mainly due to the shape of the corresponding basis functions, i.e., only a few of the basis functions are smooth or are separated into smooth regions separated by only horizontal or vertical edges. However, for regions with more complex edge structure, they tend to produce many large magnitude transform coefficients. In order to achieve an efficient representation of regions with more complex edges, we need to adapt the transforms to the edge structure. In the first part of this chapter (Section 5.3, which summarizes the work proposed by the author in [53, 56]) we focus on developing *edge-adaptive tree-based lifting transforms*, i.e., tree-based transforms that avoid filtering across edges as to avoid creating large magnitude high-pass coefficients. This leads to higher overall coding efficiency (i.e., fewer bits to achieve a fixed reconstruction quality) than the standard separable wavelet transforms used in JPEG-2000 [67].

Another way of efficiently representing images is by dividing the image into blocks, predicting the pixel values in each block using pixel values from neighboring blocks, then computing and transmitting prediction residuals. These types of schemes are referred to as *intra prediction schemes* and are a part of standards such as H.264/AVC and MPEG-4. In these standard schemes, predictions are always computed in a single direction and in order to account for directional edge information in each block (e.g., diagonal edges), a fixed set of prediction directions is used and the "best" one is chosen. These directional predictions will be effective for encoding image blocks where there is only a single diagonal edge, since then a very accurate prediction can be produced and the prediction residuals will be very small. This provides an efficient representation of each block since small prediction residuals will require many fewer bits for encoding than the original pixel values. However, in regions with more complex edge structure such as "L" shaped or "V" shaped edges, these directional prediction modes will produce large prediction residuals at pixels near the edges, and these large residuals will require many bits to be encoded. Note that complex edge structure poses the same problems for the fixed directional predictions as it does for the standard separable transforms (since many large magnitude coefficients are produced), and we can address it in a similar manner by not computing predictions across edges. As such, in the second portion of this chapter (Section 5.4, which describes the work presented by the author in [51]) we propose an *edge-adaptive intra prediction scheme* that can produce accurate predictions for blocks with more complex (non-diagonal) edge structure.

As an additional application, we also consider using the edge-adaptive transform and intra prediction scheme for coding depth map images used in multi-view video coding systems. Recent advances in multi-view video have generated interest in new applications such as 3D-TV [59], bringing a plethora of 3D video services

closer to reality. However, the amount of data captured in such systems is typically very large, making it difficult to store and transmit. We can decrease this data by (i) reducing the number of views, and (ii) using data taken from actual cameras at known positions to interpolate intermediate views. Note that view interpolation techniques such as depth-image-based rendering [76] require accurate position information in 3D space for objects in the scene, i.e., they require *depth map* information in addition to 2D pixel positions. These depth maps are an extra source of data and should therefore be compressed. However, compression artifacts in depth maps (especially around edges) can lead to annoying visual artifacts in the interpolated views [26, 27, 29]. Thus, these edge-adaptive schemes can also be used to efficiently compress depth maps while also preserving the edge information. Also note that the depth maps are actually never displayed and are only used for interpolation, so it is also important to consider the trade-off between depth map rate and interpolation distortion. Therefore, we optimize this trade-off by leveraging results from [26, 27].

We first observe that depth maps typically consist of nearly constant regions separated by edges, i.e., depth maps are nearly piece-wise constant signals. Moreover, preserving edges in depth maps often yields view interpolations that are perceptually better. Thus, using transforms and/or prediction schemes that exploit this piece-wise constant assumption while also preserving edges should lead to better interpolation results. As is shown in work by us [50], using edge-adaptive lifting transforms leads to more efficient depth map coding while also improving the quality of the synthesized views. Similar improvements were also observed for the edge-adaptive intra-prediction scheme proposed by us in [51]. The performance improvements are evaluated experimentally in Section 5.3 for the edge-adaptive lifting transforms and in Section 5.4 for the edge-adaptive intra prediction scheme.

5.2 Preliminaries

We first establish some preliminary concepts and notations that are common to the edge-adaptive tree-based lifting transform and the edge-adaptive intra prediction scheme. Let X denote the original image and denote each pixel by the pair (i, j), with its intensity value given by value $X(i, j)$. Let N_r and N_c denote the number of rows and columns of X, respectively. Note that both of these methods rely on a graph-based signal representation. Therefore, let $G = (V, E)$ denote an undirected graph, where the set V represents the vertices (i.e., the pixels) in the graph and the set E denotes the set of connections between pixels. Also let each $[(i, j), (i', j')] \in E$ denote a connection from pixel (i, j) to (i', j') in V.

Note that the edge-adaptive lifting transform and the edge-adaptive intra prediction scheme both avoid filtering across edges, thus, they both require knowledge of edge locations. We compute edge locations using the edge detector in work done by the author in [50], which is just a modification of the one used in [32]. This technique defines edges to be at fractional locations between pixels. In particular, edges around pixel (i, j) exist at $(i, j \pm 0.5)$, $(i \pm 0.5, j)$ and $(i \pm 0.5, j \pm 0.5)$, and an edge exists whenever $|X(i, j) - X(i, j \pm 1)| > T$, $|X(i, j) - X(i \pm 1, j)| > T$ and $|X(i, j) - X(i \pm 1, j \pm 1)| > T$, respectively, for some threshold T. This leads to a *binary edge map* B which has $2N_r$ rows and $2N_c$ columns and specifies an edge between pixel (i, j) and $(i, j+1)$ if and only if $B(2i, 2j+1) = 1$. Otherwise, there is no edge between them and $B(2i, 2j+1) = 0$. Note that this is equivalent to stating that an edge exists at location $(i, j + 0.5)$, just that we index it using integers by multiplying by 2, e.g., $2 \cdot (i, j + 0.5) = (2i, 2j + 1)$. Similarly, $B(2i + 1, 2j) = 1$ implies that there is an edge between pixel (i, j) and $(i+1, j)$, $B(2i+1, 2j+1) = 1$ implies that there is an edge between pixel (i, j) and $(i + 1, j + 1)$, etcetera. Note

that $B(2i, 2j) = 0$ for all i, j and if $B(2i \pm 1, 2j) = 0$, then there is no edge between pixel (i, j) and $(i \pm 1, j)$. In a similar way, $B(2i, 2j \pm 1) = 0$ implies that there is no edge between pixel (i, j) and $(i, j \pm 1)$, and $B(2i \pm 1, 2j \pm 1) = 0$ implies that there is no edge between pixel (i, j) and $(i \pm 1, j \pm 1)$. Since our transforms are constructed using this edge information, the edge map must be encoded and sent to the decoder so that it can re-produce the same transform used at the encoder.

5.3 Tree-based Lifting Transforms

When using standard separable wavelet transforms in images with complex contours, significant amounts of energy may be present in all high pass sub-bands, mostly concentrated in large coefficients near contours. Encoding these coefficients is costly in terms of rate. Reduction of such high pass sub-band energy can be achieved by applying separable filtering along directions on which pixels exhibit low intensity variation. Directionlets [71] and Bandelets [43] achieve this using (i) a block-based segmentation defining dominant directional flow of pixel intensity in each block and (ii) separable filtering along a grid aligned with each directional flow. In Directionlets integer lattices are used, while in Bandelets the original grid is re-mapped onto a grid aligned with directional flows, on which separable filtering is applied.

In both of these methods a set of "good" paths for traversing the pixels is defined (i.e., one that exhibits low variation in pixel intensity) and then filtering is applied along those paths. Both of these approaches are constrained in that they assume that a set of parallel paths can approximate well the geometric flow of an image in a given region, e.g., a block. Moreover, both of these transforms provide superior performance to standard separable wavelets mainly because they do not

filter across edges. Thus, one can still achieve competitive coding performance to these transforms by performing edge-avoiding filtering as is done in, e.g., the shape-adaptive DWT [30] and the work in [11], where edge-avoiding filtering is done only along rows and columns. The main contribution in this section is to propose techniques that avoid filtering across edges while also loosening the constraint of parallel paths by allowing non-parallel paths for filtering (unlike [11,30,33,71]) and by not restricting flows to be uniform within blocks (unlike [33, 71]). This leads us to a set of edge-adaptive tree-based wavelet transforms, where the main goal is to design trees which do not cross over discontinuities in an image, and then to perform filtering along these trees. This serves to reduce the amount of energy in the high-pass subbands, and ultimately leads to greater coding efficiency.

These types of trees were initially developed by the author in [53] where multiple minimum spanning trees (MSTs) with different orientations were used to provide a separable transform. This approach has two major drawbacks. First, lifting-based prediction and update filters lead to an invertible representation but, given the structures of the trees (e.g., the merge points), the corresponding basis functions tend to be far from orthogonal (unlike similar lifting structures for 1D regularly sampled signals). This results in signal energy being spread evenly across subbands which reduces energy compaction. Second, the MST traversal does not guarantee that downsampled pixel locations will be regularly spaced in successive levels of decomposition. Thus filtering is often applied across pixels that are far apart. Our work in [56] improved upon this by (i) designing orthogonalizing update filters (as detailed in Section 2.4.2), and by (ii) designing trees that provide regular downsampling patterns.

In this section we describe the work in [56]. Section 5.3.1 describes the tree-based transform construction. We then discuss tree constructions that produce

regular downsampling patterns while avoiding crossing over discontinuities in Section 5.3.1.2. Some experimental results are shown in Section 5.3.2 and some concluding remarks are provided in Section 5.5.

5.3.1 Tree-based Transform Design

We now describe how tree-based wavelets such as those proposed in Section 3.3.4 can be applied to images. Assume a tree T has been selected that allows us to traverse the pixels in the image from leaves to the tree root. Suppose we index the root node of T by $(MN+1, MN+1)$. Let $\mathcal{C}_{m,n}$ and $\rho_{m,n}$ denote the set of children and the parent of pixel (m,n) in T, respectively, and let $\mathcal{N}_{m,n} = \mathcal{C}_{m,n} \cup \{\rho_{m,n}\}$. Finally, let $\text{depth}(m,n)$ be the depth of pixel (m,n) in T, with $\text{depth}(MN+1, MN+1) = 0$. A lifting transform can then be developed along T as discussed in Section 2.2.

Let $\mathcal{O}$ and $\mathcal{E}$ denote the sets of odd and even pixels and let $\mathbf{p}_{m,n}$ and $\mathbf{u}_{k,l}$ denote the prediction and update operators at pixels $(m,n) \in \mathcal{O}$ and $(k,l) \in \mathcal{E}$, respectively. For the sake of simplicity, we only allow pixels to use data from their one-hop neighbors, i.e., for all odd pixels (m,n) we require $\mathbf{p}_{m,n}(m,n) = 1$ and $\mathbf{p}_{m,n}(i,j) = 0$ for all $(i,j) \notin \mathcal{C}_{m,n} \cup \{\rho_{m,n}\}$. Similarly, for each $\mathbf{u}_{m,n}$. The transform is computed as follows. For each $(k,l) \in \mathcal{P}$:

$$d(k,l) = X(k,l) - \sum_{(i_t,j_t) \in \mathcal{N}_{k,l}} \mathbf{p}_{k,l}(i_t,j_t) X(i_t,j_t) \tag{5.1}$$

and given every $d(k,l($, for each $(m,n) \in \mathcal{U}$ we have:

$$s(m,n) = X(m,n) + \sum_{(i_t,j_t) \in \mathcal{N}_{m,n}} \mathbf{u}_{m,n}(i_t,j_t) d(i_t,j_t). \tag{5.2}$$

5.3.1.1 Lifting Filter Design

Note that images are locally smooth in that around each pixel (m, n), the values of neighboring pixels tend to be similar to $X(m, n)$. Thus, whenever an odd pixel (m, n) has at least 3 neighbors in the tree the planar-based prediction filter discussed in Section 2.3.1 can be used to provide a good prediction of its data. If an odd pixel (m, n) has at most 2 neighbors, then clearly a planar based prediction cannot be generated, so for these pixels we use a simple average. More specifically, whenever $|\mathcal{N}_{m,n}| > 3$, $\mathbf{p}_{m,n}(\mathcal{N}_{m,n}) = -\left([m \;\; n \;\; 1] \cdot \left(\mathbf{A}_{m,n}^t \mathbf{A}_{m,n}\right)^{-1} \cdot \mathbf{A}_{m,n}^t\right)^t$, where $\mathcal{N}_{m,n} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{|\mathcal{N}_{m,n}|}, j_{|\mathcal{N}_{m,n}|})\}$ and

$$\mathbf{A}_{m,n} = \begin{bmatrix} i_1 & j_1 & 1 \\ \vdots & \vdots & \vdots \\ i_{|\mathcal{N}_{m,n}|} & j_{|\mathcal{N}_{m,n}|} & 1 \end{bmatrix}.$$

Whenever $|\mathcal{N}_{m,n}| = 1$ or $|\mathcal{N}_{m,n}| = 2$, $\mathbf{p}_{m,n}(\mathcal{N}_{m,n}) = -\frac{1}{|\mathcal{N}_{m,n}|}$. For the update filters, we choose the orthogonalizing update filter design proposed in Section 2.4.2.

5.3.1.2 Tree Construction

In this thesis, we compute a set of discontinuities $\mathcal{D}$ is defined using edges in an image. We use the edge detector and the corresponding graph described in Section 5.2. A tree is then constructed this graph (which has no links which cross over edges). In early work by us [53], MSTs were used to avoid filtering across discontinuities. However, when applying multiple levels of decomposition by subsampling the MSTs based on parity of depth, there is no guarantee that pixels will be regularly spaced at each level. This causes the sampling grid at each level to be highly irregular, i.e., the distance (in the image) between pixels that are neighbors in the

tree can vary significantly. As a result, it is possible that a group of pixels that are neighbors in the tree are located far from each other in the image. This loss of "spatial locality" in the filtering operations does not occur when using separable 2D transform (for which the resulting 2D filters in a given subband have the same spatial localization at all positions).

This was addressed in later work by the author [56] by constructing (horizontal and vertical) trees that avoid filtering across major discontinuities while preserving regular sampling grids over multiple levels of decomposition. This tree construction process is now described for one level of decomposition. The binary edge map is encoded using, e.g., JBIG [24] and is sent as side information so that the decoder can construct the same trees. Sample trees are shown in Figure 5.1 with discontinuities shown by red dots.

The horizontal tree is constructed starting from pixels in the right-most column of the image to pixels in the left-most column. Assume N is even and that pixels in the right-most column are even. We want pixels in an even (odd) column to be even (odd) as to preserve column-wise parity. To do so, we just need to specify a parent for each pixel such that the link between itself and its parent does not cross over discontinuities and does not induce cycles. Since construction progresses from right to left, a natural set of parental candidates for pixel (m, n) is a set of pixels to its left given by $\{(k, n-1)|m_l \leq k \leq m_u\}$ for $m_l \leq m \leq m_u$. Let $\text{Cand}_{m,n}$ denote this set of candidates. To avoid filtering across discontinuities, we must eliminate any candidate $(k, n-1)$ such that a discontinuity point exists between (m, n) and $(k, n-1)$. So if $\mathcal{L}_{m,n}^{k,n-1}$ is the set of points on the line segment between (m, n) and $(k, n-1)$, we have

$$\text{Cand}_{m,n} = \{(k, n-1)|m_l \leq k \leq m_u, \mathcal{L}_{m,n}^{k,n-1} \cap \mathcal{D} = \emptyset.\} \tag{5.3}$$

If $\text{Cand}_{m,n} \neq \emptyset$, the parent of (m, n) is chosen as the pixel in $\text{Cand}_{m,n}$ closest to (m, n). Figure 5.1 (b) shows an example of this horizontal tree when $m_l = m - 2$ and $m_u = m + 2$.

If $\text{Cand}_{m,n} = \emptyset$ (no valid parental candidates), we do the following. If n is odd, the parent of (m, n) is the sink. In the example of Figure 5.1 (a), all of the pixels in the first column have no valid parental candidates, thus, their parent is simply the sink. If n is even, then setting the parent of (m, n) as the sink will not preserve the proper parity. Instead, we can set the parent of (m, n) as $(m, n + 1)$, in which case the parent of $(m, n + 1)$ must be set as the sink to avoid creating a cycle. For the same example, pixel $(4, 4)$ is even with $\text{Cand}_{4,4} = \emptyset$ so its parent is $(4, 5)$ and the parent of $(4, 5)$ is the sink.

Each vertical tree is constructed in a similar manner. Note how the horizontal and vertical trees in Figure 5.1 (b), (c) and (d) avoid filtering across discontinuities.

5.3.1.3 Separable Tree-based Transforms

Note that we can compute the transform over *multiple trees, with different orientations*, with a different tree used at each level of decomposition. We now outline a general framework for computing general 2D transforms along multiple trees in a separable manner, with the main goal of exploiting directionality in images. A simple example of such separable trees is shown in Figure 5.1. Suppose we are given an arbitrary method for constructing trees that follow the geometric flow in an image given a prespecified root node (or more generally, set of root nodes). For a one level decomposition (using a tree in one direction followed by 2 trees in another), we would first apply 1 level of decomposition along one tree T_1 (as shown in Fig 5.1(b)) oriented in one direction, then split the set of coefficients in T_1 into even (low pass) and odd (high pass) subsets (according to their depths in T_1) and

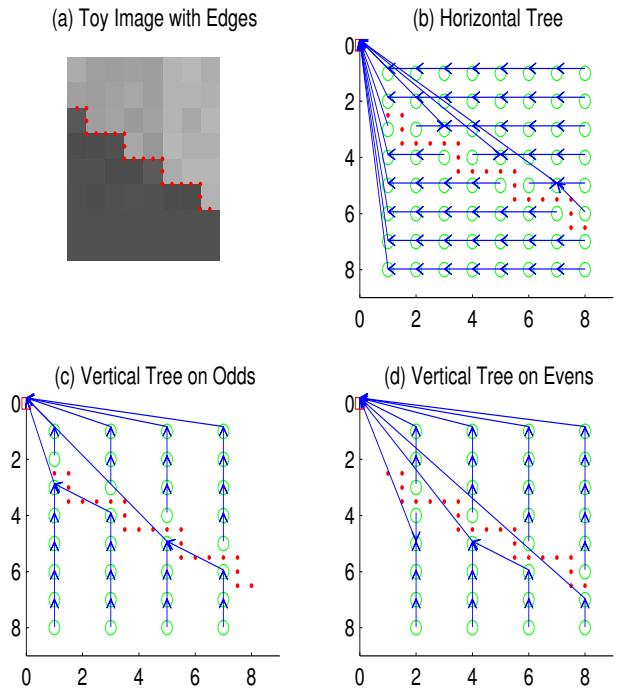


Figure 5.1: Example to illustrate tree construction, where links in the tree (denoted by blue lines between pixels) are not allowed to cross edges in the image (denoted by red dots)

then run a one level transform along a second tree $T_{1,l}$ (as shown in Fig 5.1(c)) and third tree $T_{1,h}$ (as shown in Fig 5.1(d)) over the even and odd subsets respectively.

5.3.2 Experimental Results

We compare the performance of our proposed tree-based lifting transform against the standard 9/7 separable transform [67], the standard 5/3 separable transform [67] and second generation bandelets [44] [1]. All of the transform coefficients are encoded using SPIHT [49][2]. For the tree-based transforms, we also compare mean preserving update filters (Section 2.4.1) against orthogonalizing update filters (Section 2.4.2).

[1] http://www.cmap.polytechnique.fr/peyre/bandelets/
[2] http://www.cipr.rpi.edu/research/SPIHT/

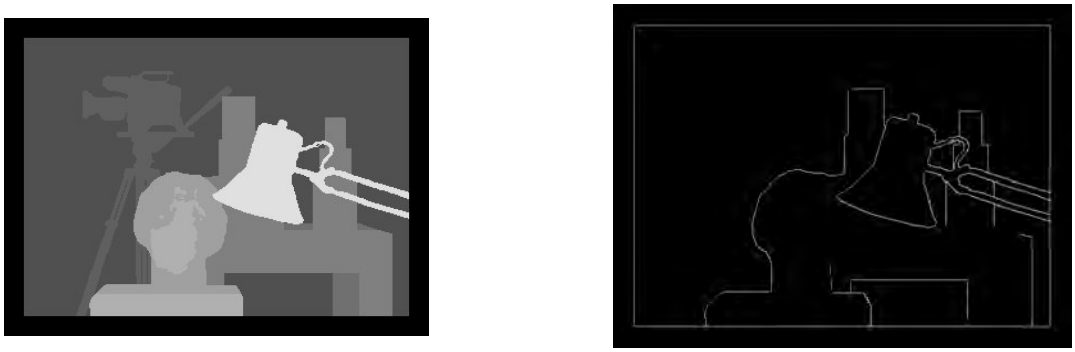
The edge maps are generated using the edge detector described in [32] and are encoded using JBIG [24]. Five levels of decomposition are used in all cases. We evaluate the relative performance on the standard peppers image and also using the ground truth depth map taken from the Middlebury data set[3]. The peppers image and its corresponding edge map are shown in Figure 5.2(a) and 5.2(b), respectively. The Tsukuba depth map and its edge map are shown in Figure 5.3(a) and 5.3(b), respectively.

(a) Peppers image (b) Edge map

Figure 5.2: The Peppers image (a) and its corresponding edge map (b)

Coding performance is shown in Figure 5.4 and 5.5 for the peppers and tsukuba images, respectively. The edge threshold used here is $T = 30$. Despite the additional bits for edge information, the tree-based transforms still gives performance far superior to the standard transforms, with up to 2 dB increase in PSNR for the peppers image and 7 dB increase for the tsukuba image. Most of this gain comes from not filtering across edges. Second generation bandelets [44] only provides up to 0.2 dB improvement over the standard transforms. Note that that this scheme

[3]http://vision.middlebury.edu/stereo/



(a) Tsukuba depth map (b) Edge map

Figure 5.3: The Tsukuba depth map (a) and its corresponding edge map (b)

does processing in the wavelet domain by searching for the "best" mapping of a square block onto a 1D line, then applies an orthogonal 1D wavelet transform on the corresponding 1D signal. However, as was observed in their work, most of the resulting 1D signals still have many sharp transitions (e.g., edges). Thus, it is still likely to have some large wavelet coefficients, though the number of total number of large wavelet coefficients is reduced. On the other hand, our transform operates in the spatial domain by avoiding filtering across known (i.e., detected) edge locations. Thus, for these types of piece-wise smooth images with very little texture, our transforms tend to produce fewer large wavelet coefficients than second generation bandelets, hence the better performance. However, for images with more directional texture information, bandelets is likely to do better than our proposed transforms. We will examine this shortly by using test images that contain a large amount of texture.

The orthogonalizing update filters provide an additional 0.1 dB to 1.5 dB improvement in PSNR over non orthogonalizing update filters, and more gain is seen at lower bit rates. This is not surprising since the orthogonalizing update filters

from Section 2.4.2 showed that this choice of update filters minimizes the reconstruction MSE due to quantization. Thus, we obtain performance improvements by (i) not filtering across edges, (ii) using merges and (iii) using orthogonalizing update filter designs. It is worth noting that using our proposed transform on trees with merges does better than on trees with no merges, but only provides another 0.05 dB gain on average. Thus, using merges may not provide significant improvements in general. The reconstructed depth maps at 0.25 bpp are also shown in Figure 5.6. Clearly the reconstruction using our tree-based transform looks much better and has many fewer ringing artifacts.

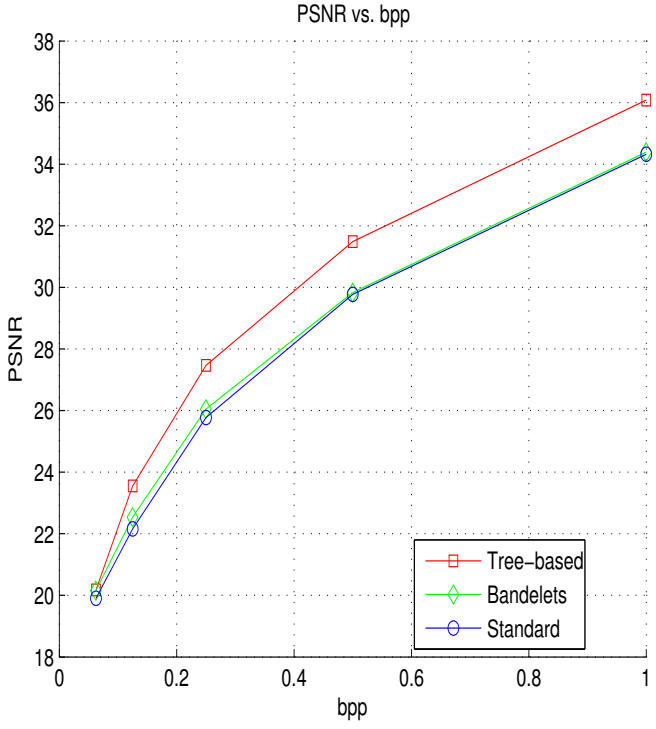


Figure 5.4: Rate-distortion curve for various transforms using peppers image. Tree-based transforms give the best performance, and orthogonalizing update filters provide additional gain over mean-preserving update filters.

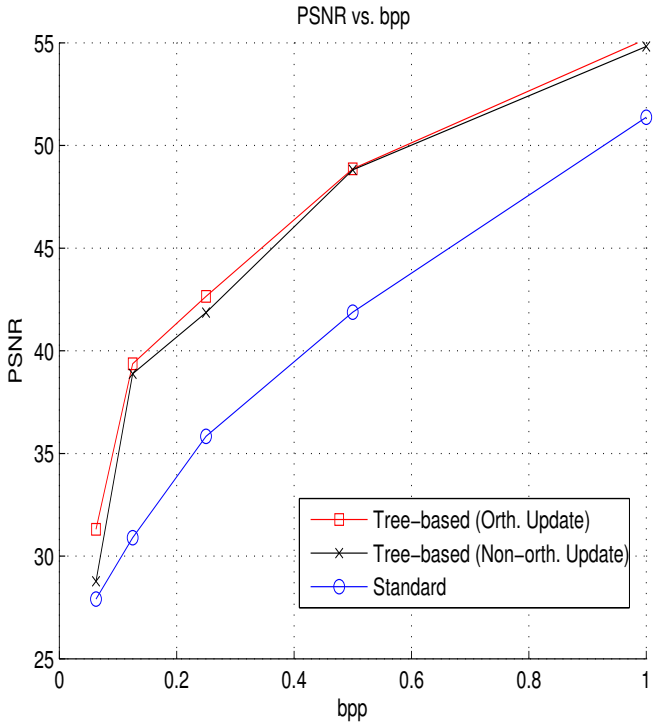


Figure 5.5: Rate-distortion curve for various transforms using depth map image. Tree-based transforms give the best performance, and orthogonalizing update filters provide additional gain over mean-preserving update filters.

For the piece-wise smooth images shown in Figures 5.2(a) and 5.3(a), our transform outperforms the standard wavelet filters and 2nd generation bandelets. However, second generation bandelets will probably outperform our proposed method for images with many textured regions, particularly since edge detection will not be very reliable for textured regions. We examine this using the standard Lena and Barbara images shown in Figures 5.7(a) and 5.7(b), respectively. The corresponding RD curves are shown in Figures 5.8(a) and 5.8(b), respectively. For the Lena image, 2nd generation bandelets does slightly better at lower bitrates and is the same at higher rates. On the other hand, our edge-adaptive lifting transform is worse at lower rates but better at higher rates. The main reason it is worse at lower rates is because of the high edge map bit rate (around 0.018 bpp), thus, much

(a) Tree-based transform (b) Standard 9/7

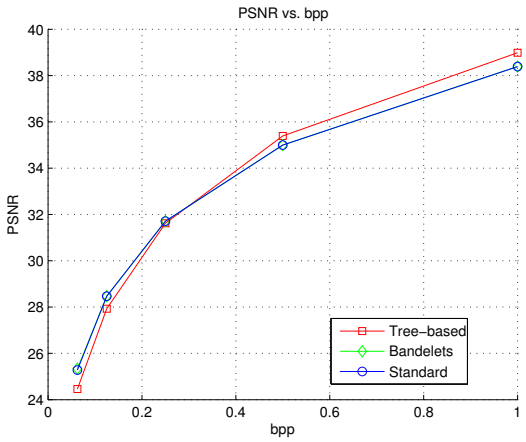
Figure 5.6: Subjective performance comparison at 0.25 bpp. Our proposed method has a PSNR of 42.65 dB whereas the standard 9/7 transform has PSNR of 35.83 dB. This difference is clearly reflected in the reconstructed images

of the bitrate is consumed in coding the edge information. For the Barbara image, our edge-adaptive transform performs about the same as the standard transform and bandelets at lower bitrates, but (just as with the Lena image) does better at higher bitrates. The main cause for worse performance at lower bitrates is the same as with the Lena image. Therefore, our edge-adaptive lifting transform should consistently outperform existing methods for piece-wise smooth images with very little texture, but not necessarily for images with textured regions at lower bitrates.

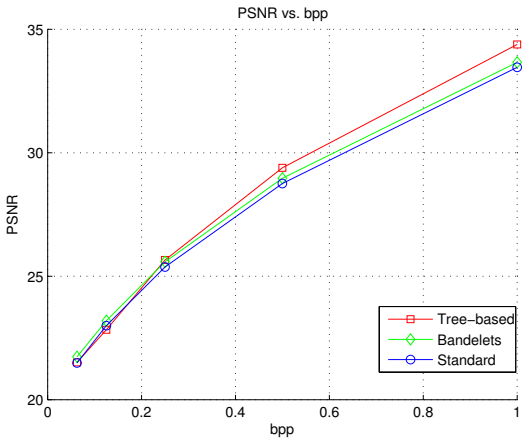
(a) Lena image

(b) Barbara image

Figure 5.7: The Lena image (a) and Barbara image (b)



(a) Lena RD Curve



(b) Barbara RD Curve

Figure 5.8: RD curves for the Lena image (a) and Barbara image (b)

5.4 Edge-Adaptive Intra Prediction

We now introduce our edge-adaptive intra prediction scheme originally proposed in [51]. This follows the same spirit as the wavelet transforms proposed in Section 5.3 in that filtering across edges is avoided. However, the algorithms in this section provide a set of edge-avoiding graphs, whereas in Section 5.3 a set of edge-avoiding trees is provided. Moreover, only edge-avoiding prediction is considered. Edge-adaptive wavelet transforms have been proposed in Section 5.3 as well as in [32, 50]. However, note that these transforms are not easily amenable to block based processing as used in H.264. Platelets [75] have been proposed for efficient representation of piece-wise planar images, and these ideas were extended to depth map encoding in [35]. The coding results of [35] are quite good with respect to JPEG-2000. However, the edges are approximated by lines, hence, they become "smoothed out". This may lead to worse interpolation results. Moreover, only the parameters for the planar approximation are sent without any residue, so there will always be some fixed approximation error. In this section we seek to develop a block-based, edge-aware intra prediction scheme that can preserve the edge information in images while also being easy to integrate with H.264.

The intra prediction modes in H.264 provide good predictions for blocks consisting of flat regions or regions with only horizontal, vertical or diagonal edges, e.g., Figure 5.9(a) and 5.9(b). However, those modes do not provide good predictions for blocks with arbitrary edge shapes, e.g., Figure 5.9(c). Figure 5.10 shows the *predicted pixels* (pixels *a*-*p*) and *predictor pixels* (pixels *A*-*M*) used in H.264 4×4 intra prediction. In blocks such as Figure 5.9(c) there will often be edges between pixels and their predictors. This leads to large prediction residuals which require more bits to be represented. Moreover, when used to encode depth map images

used for view synthesis in a multi-view video coding system, quantization of these large residuals produces ringing artifacts around the depth map edges, and these tend to produce annoying visual artifacts in the synthesized views. To address this inefficiency, we develop an *edge-aware intra prediction scheme* that provides accurate predictions for blocks with arbitrary edge shapes. Our proposed scheme provides an exact representation (up to quantization errors, unlike [35]) and can be easily integrated with H.264 coding tools (unlike [32, 50]). No such method has been developed yet to the best of our knowledge.

(a) Horizontal (b) Diagonal (c) Arbitrary

Figure 5.9: Examples of blocks with different edge structure. Blocks such as those in (a) and (b) can be efficiently represented by existing intra prediction schemes. Blocks such as those in (c) are not efficiently represented.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figure 5.10: Predicted pixels (a-p) and predictor pixels (A-M) used in H.264.

We separate the design of edge-aware intra prediction schemes into three parts: (i) detection of edge locations, (ii) identification of "valid predictors" which do not have an edge between themselves and a given pixel, and (iii) prediction of the intensity of a pixel using the intensities of its "valid predictors". Edge locations are found using the technique described in Section 5.2. Valid predictors for each predicted pixel are then identified by finding paths (in a graph) to predictor pixels that do not cross edges. A method for choosing a prediction among the valid predictors is also proposed in the case of depth map images, though more general prediction choices could be made for other types of images. In fact, the primary focus of this section is on efficient, edge-preserving depth map coding for use in multi-view video coding systems, though the ideas presented here can be easily extended to more general types of images.

Since depth maps typically consist of nearly flat regions separated by edges, *we assume that depth maps are piece-wise constant signals*. In this case, the best prediction for a given pixel is just the intensity value of any of its valid predictors. In order to optimize the trade-off between depth map bitrate and interpolation distortion, we leverage results from previous work [26, 27]. In particular, we use the distortion metric proposed in [27] in the rate-distortion (RD) optimized mode selection. This yields an additional improvement on top of what the new edge-aware intra prediction scheme provides.

This section is organized as follows. Section 5.4.1 describes our proposed edge-adaptive intra prediction scheme. The optimization between depth map bitrate and interpolation distortion is described in Section 5.4.2. Section 5.4.3 shows experimental results demonstrating the gains of our proposed methods. In particular, we are able to reduce the bit rates for coded depth maps by up to 37% for a fixed interpolated PSNR. Finally, some concluding remarks are made in Section 5.5.

5.4.1 Edge-adaptive Intra Prediction

We describe our scheme only in the case of 4×4 intra prediction, but it can be easily extended to other block sizes. We separate the design of an edge-aware intra prediction scheme into three parts. First we must find edge locations using an edge detector as described in Section 5.4.1.1. Once we compute the edge locations, we must then determine the set of "valid predictors" for each pixel in a given block. This is done using a graph-based representation of the pixels in a block, as described in Section 5.4.1.2. Finally, for every pixel in a block, we must determine the prediction value for each valid predictor. This is also described in Section 5.4.1.2 under a piece-wise constant signal model for depth maps.

5.4.1.1 Edge Detection

We compute edge locations using the edge detector in Section 5.2, which is just a modification of the one used in [32]. This edge detector produces a binary edge map that must be encoded and sent to the decoder so that it can produce the same (edge-adaptive) predictions used at the encoder. However, for many test sequences we observe that the number of bits required to encode the edge map for the entire image is very large. Thus, it is necessary to control the amount of edge information that is generated. We describe a block-based coding scheme for the edge information in Section 5.4.1.3.

5.4.1.2 Predictor Selection

In order to find accurate predictors for each pixel, we must first identify valid prediction neighbors. Then, given the set of valid predictors, we must compute prediction values. We identify the "valid predictors" of each pixel as follows. First,

for each block we form an undirected graph $G = (V, E)$ which has as vertices the pixels a-p in the given block and the pixels A-M from previously coded blocks (see Figure 5.10). For each pixel (i, j), we restrict the connections to be only with its left, right, top and bottom neighbors, i.e., pixel (i, j) can only have connections with pixel $(i, j-1)$, $(i, j+1)$, $(i-1, j)$ and $(i+1, j)$. Then for pixel (i, j), we define a connection between (i, j) and $(i, j-1)$ if and only if there is no edge between them, i.e., $[(i, j), (i, j-1)] \in E$ if and only if $B(2i, 2j-1) = 0$. The connections between (i, j) and $(i, j-1)$, $(i+1, j)$ and $(i-1, j)$ are defined in a similar manner. As an example, consider the image in Figure 5.11, where two constant regions are separated by an edge. For the example shown in Figure 5.11 the resulting graph used for searching for valid neighbors is shown in Figure 5.12. Note that pixel m only has connections to pixels n and L. It is not connected to pixel i since there is an edge between them.

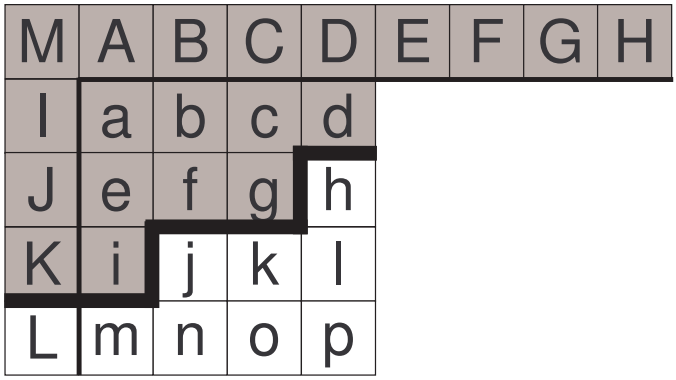


Figure 5.11: Example of valid predictors. This section of the image consists of two flat regions separated by an edge shown by the thick solid line. In this case pixels $A, B, \ldots, K$ and M are all valid predictors for pixels $a, b, \ldots$ and i, but are not valid predictors for pixels h and $j, k, \ldots, p$. On the other hand, pixel L is only a valid predictor for pixels h and $j, k, \ldots, p$.

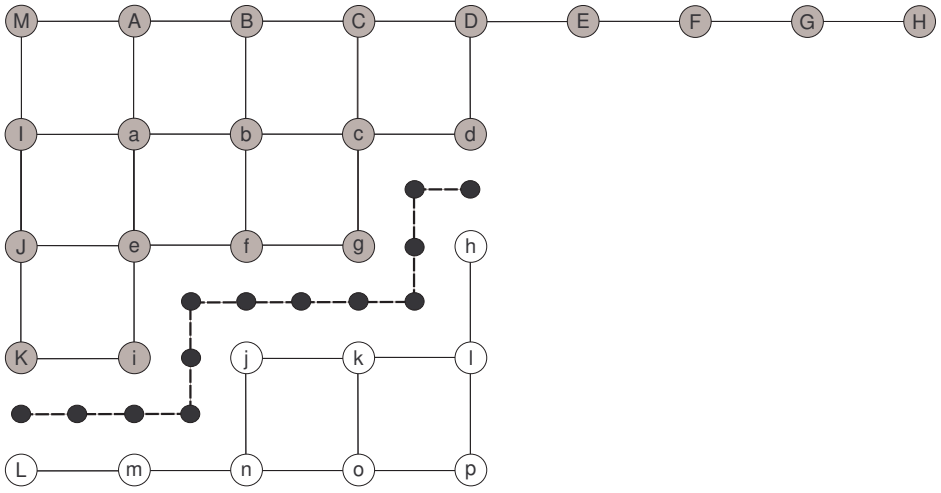


Figure 5.12: Example of graph used to find valid predictors using the same sample from Figure 5.11. The thick dotted line with small black circles denotes the edges. Thin solid lines between pixels represent connections in the graph G. The thick solid line represents the boundary between the current block and previously coded blocks.

We use this graph-based representation to emphasize the generality of our approach, where the goal is to (i) find all the valid prediction neighbors of each pixel at different distances, then to (ii) compute a prediction based on the values of these valid predictors. However, other techniques could be used that would be easier to implement in practice. Using this representation, we can systematically determine if an edge exists between a predicted pixel and a predictor pixel as follows. Note that there is no edge between two pixels if a path between them exists in G. This can be checked as follows. First let $\mathbf{A}(u, v)$ denote the adjacency matrix of graph G, where $\mathbf{A}(u, v) = 1$ if $[u, v] \in E$, and $\mathbf{A}(u, v) = 0$ if $[u, v] \notin E$. For a general graph G and adjacency matrix $\mathbf{A}$, a path of length k exists between vertex u and v if $\mathbf{A}^k(u, v) > 0$ [70]. Thus, for each pixel a-p, we can use this to determine if paths exist to pixels A-M. The set of valid neighbors for a given pixel is simply the set of pixels A-M which have a path to it. This entire process is described in detail in Algorithm 2, where for each pixel x, $\mathcal{V}_x$ denotes the set of valid predictors of x. A

valid predictor for each pixel can then be chosen among those in the set $\mathcal{V}_x$. In the implementation of our algorithm, we set $k_{max} = 8$.

Algorithm 2 Find Valid Predictors

```
Perform edge detection as described in Section 5.2
Form the adjacency matrix A as described in Section 5.2
for each pixel x = a, b, ..., p do
    V_x = ∅
    for each pixel y = A, B, ..., do
        for k = 1, 2, k_max do
            if A^k(x, y) > 0 then
                V_x = V_x ∪ {y}
                Break for loop
            end if
        end for
    end for
end for
```

Figure 5.12 provides an example of this. For instance, pixel p has a path of length 4 to pixel L, but it has no paths to pixels A-K nor to pixel M. Thus, pixel L is the only valid predictor for pixel p. Note that this technique can be used to identify the set of valid prediction neighbors for other block sizes (e.g., it can be used for edge-adaptive intra prediction in 16×16 blocks). If there exists a single pixel that does not have a path to any pixel A-M, then the edge-adaptive intra prediction scheme obviously can not be used. In such cases, only standard prediction modes are used.

Now that we have a way to determine the valid predictors of each pixel, all that remains is to determine which predictors to use and how to compute the predictions. If a pixel has multiple valid prediction neighbors, then *it is possible to predict the intensity at each pixel using the values from multiple prediction neighbors.* However, we note that depth maps typically consist of flat regions separated by edges, i.e., depth maps are nearly piece-wise constant signals. Thus, if pixel (i', j') is a valid predictor of pixel (i, j), then there is no edge between (i', j') and (i, j), hence, $X(i, j) \approx X(i', j')$ and the prediction error is almost zero. As such, one valid prediction neighbor is sufficient to predict each pixel. Thus, if a pixel has multiple valid predictors, we simply choose the one which is the least number of hops away in the graph G as its predictor. For example, in Figure 5.11 and 5.12, pixels c and g will use C as its predictor, pixel d will use pixel D, etc. In the event of a tie, we choose the lowest indexed pixel as the predictor, e.g., pixel a will be predicted by pixel A, pixel f will be predicted from pixel B.

If a given depth map is not actually piece-wise constant (e.g., it is piece-wise planar or piece-wise smooth), we can compute more accurate predictions by using values from multiple valid prediction neighbors. This can be done using a simple average, a linear / planar approximation, or even by using the spectral representation of the graph G [19]. This remains a topic for future work.

5.4.1.3 Discussion

Recall that edge-adaptive prediction is likely to be more useful in blocks with arbitrary edge shapes, e.g., Figure 5.9(c). Since it may only be useful for these types of blocks, it would be better to decide whether edge data should be sent on a block by block basis. This way we only need to encode edge data for such blocks and that choice can be encoded with a simple mode bit. In order to compute these

predictions for a given block, we also need the edge information from neighboring reconstructed blocks. For example, in order to find the valid predictors for pixels *a-p* in Figure 5.11, we also need to know the edge information from pixels *A-M*. However, note that the edge information from neighboring blocks can be derived from the decoded data. Furthermore, these reconstructed blocks will also be available at the decoder. Thus, as long as the encoder (i) encodes edge data for such a block and (ii) derives edge data from neighboring blocks by using the reconstructed values, it will be possible for the decoder to regenerate the same edge information used at the encoder. More importantly, *edge data only needs to be sent for blocks in which it helps*. We can also add our proposed prediction scheme as another mode and use it with the RD-optimized mode selections in H.264. This allows us to further reduce edge information if the RD cost for such blocks (which includes the bits needed to represent the edges) is not the minimum cost among all intra prediction modes. Therefore, we only encode edge information for a block if (i) it has complex edge shapes and (ii) edge-adaptive prediction yields the minimum cost. The corresponding edge maps can be encoded using schemes such as binary arithmetic coding.

5.4.2 RD Optimization

When a depth map is compressed using lossy coding, distortion occurs in the reconstructed depth map. However, depth map decoding errors affect video quality only indirectly, since depth maps are used for interpolation of intermediate views (and the depth maps themselves are not displayed). In previous work [27] it was shown that the depth map error causes translation error in the rendering process, and using this translation error the resulted distortion in the rendered view can be

$$\begin{pmatrix} \Delta x_{im} \\ \Delta y_{im} \\ 1 \end{pmatrix} = \left(\frac{1}{Z_p\left(x_{im}, y_{im}\right) + \Delta Z_p\left(x_{im}, y_{im}\right)} - \frac{1}{Z_p\left(x_{im}, y_{im}\right)} \right) \mathbf{A}_{p'} \mathbf{R}_{p'} \left\{ \mathbf{T}_p - \mathbf{T}_{p'} \right\}$$
$$= \frac{\Delta L_p\left(x_{im}, y_{im}\right)}{255} \left(\frac{1}{Z_{near}} - \frac{1}{Z_{far}} \right) \mathbf{A}_{p'} \mathbf{R}_{p'} \left\{ \mathbf{T}_p - \mathbf{T}_{p'} \right\}. \quad (5.4)$$

estimated by comparing two video pixel values - one at the same location as the depth map value to be coded and the other translated due to depth map error, both in the video belongs to the same view as depth map.

First, the translation error can be found using the intrinsic and extrinsic camera parameters along with nearest and farthest depth values in the scene as in (5.4), where Δx_{im} and Δy_{im} are translation errors (due to distortion in the decoded depth map) in the x and y direction, respectively, at image coordinate (x_{im}, y_{im}). Z denotes the depth value, ΔL is depth map distortion at that pixel, and p and p' indicate view indices. The camera intrinsic parameters are denoted $\mathbf{A}$, and extrinsic parameters are the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{T}$. Z_{near} and Z_{far} indicates the nearest and farthest depth value in the scene, respectively. Note that these parameters can be calculated once for every depth map values in the scene so that they can be used to compute the distortion corresponding to each depth map value. Then, using this translation error, the distortion in the rendered view can be estimated by measuring the distance between pixel at position (x_{im}, y_{im}) and a pixel translated to $(x_{im} + \Delta x_{im}, y_{im} + \Delta y_{im})$, where both pixels belong to the same video frame (refer to [27] for details).

We apply this new distortion metric to the RD optimized mode selection process to decide whether the proposed intra prediction mode will be used for each 4×4 and

16×16 blocks. That is, when the RD cost is calculated, the estimated distortion at the rendered view is used with the bitrate to code the depth map.

5.4.3 Experimental Results

An implementation of our proposed prediction method has been made based on H.264 / AVC (joint model reference software ver. 13.2) and the improvements that our techniques provide are demonstrated by testing with the 'Ballet' and 'Breakdancers' sequences provided by Microsoft Research [80]. We encode 15 video and depth map frames with intra only coding using $QP = 24, 28, 32$ and 36, where the proposed intra prediction applied only to depth map coding. The View Synthesis Reference Software (VSRS) 3.0 [66] is used to generate the rendered view between two coded views. We apply our proposed scheme only to 4×4 and 16×16 blocks which have both horizontal and vertical edges. Standard prediction modes are used for all other blocks. A binary edge map is generated for each depth map frame as described in Section 5.4.1.1. For each 4×4 (or 16×16) block which has horizontal and vertical edges, the corresponding block in the edge map is encoded with a binary arithmetic coder. No edge information is encoded for blocks which have no edges or only horizontal or vertical edges. In order to facilitate a quick implementation, we have simply replaced the horizontal up mode with our proposed prediction mode for 4×4 blocks and replaced the planar prediction mode with our proposed prediction mode for 16×16 blocks. The results for the 'Ballet' and 'Breakdancer' sequences are shown in Figure 5.13. Our proposed intra prediction provides Bjontegaard Delta Bit Rate (BDBR) reduction of 26% and 8% for 'Ballet' and 'Breakdancer' sequences, respectively. The proposed intra prediction with new

distortion metric yields BDBR reduction of 29% and 16% for 'Ballet' and 'Breakdancer' sequences, respectively. The edge map bit rates for these sequences are shown in Table 5.1. Note how the edge coding scheme proposed in Section 5.4.1.3 decreases the amount of edge data as the QP value increases. This is reasonable since at lower (resp. higher) rates the edge map bit rate consumes most of (resp. less of) the total bit rate.

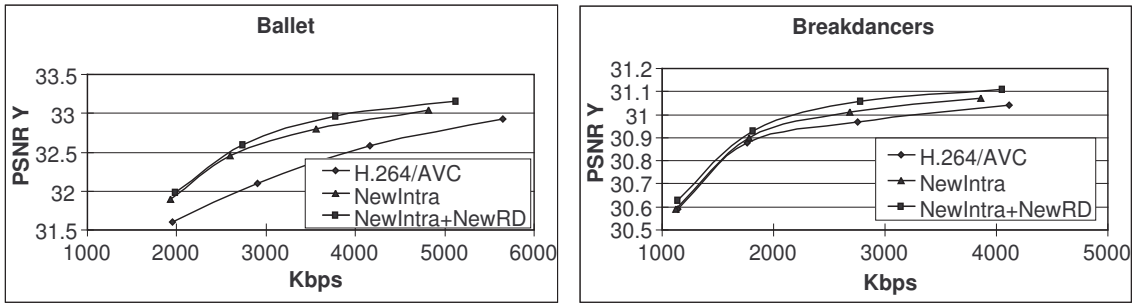


Figure 5.13: Comparison of the rate-distortion curves between the proposed methods and H.264 AVC. x-axis: total bitrate to code two depth maps; y-axis: PSNR of luminance component between the rendered view and the ground truth.

QP	**24**	**28**	**32**	**36**
Edges (Ballet)	483	481	469	383
Edges (Break)	403	351	177	17

Table 5.1: Edge map bit rates (in kbps).

When using our new intra mode with inter frames in an IPPP structure, we see similar gains as shown in Figure 5.14. For this IPPP structure, we see lower bit rates than for all I frame structure (for fixed interpolation quality). Note that in H.264, intra modes are also used in inter (P) frames. Thus, our new intra prediction mode is also utilized to improve the coding efficiency for P frames.

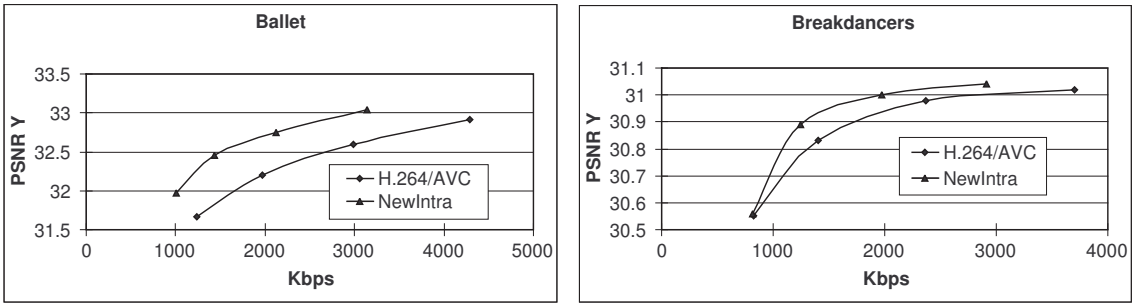


Figure 5.14: Comparison of the rate-distortion curves between the proposed methods and H.264 AVC using IPPP structure. x-axis: total bitrate to code two depth maps; y-axis: PSNR of luminance component between the rendered view and the ground truth.

5.5 Conclusions

A general class of separable edge-adaptive tree-based wavelet transforms has been provided that provides superior RD performance over existing methods. This is due in part to the orthogonalizing update filter design as well as by using trees that avoid filtering across discontinuities. It mainly provides improvements for piecewise smooth images with little to no texture, but it is not always better for images with textured regions. Moreover, merges do provide some improvements, but the gains are not significant as compared with a tree without any path merges. Thus, using trees with merges may not be useful when performing separable, tree-based transforms along rows and columns. However, it may be useful for certain classes of images. This would be an interesting topic for future work.

An edge-adaptive intra prediction scheme for depth map coding was also proposed, and has been integrated with H.264. The edge-adaptive scheme is also constructed from an edge map, and a graph-based representation of the pixels based on the edge information. The edge information is encoded on a per block basis, and is directly accounted for in the optimized mode selection used in H.264. This

new scheme (without the new distortion metric) provides up to a 33% reduction in total bitrate for a fixed PSNR in the interpolated views. An existing distortion metric has also been employed that takes into account the quality of the interpolated views and when used with our proposed can provide up to 37% reduction in total bitrate. This demonstrates the efficacy of the proposed edge-adaptive intra prediction scheme.

Chapter 6

Conclusions

A set of de-correlating tree-based and graph-based transforms have been proposed which can be applied to irregularly (and regularly) spaced data. In particular, a general set of lifting transforms on graphs and trees were proposed in Chapter 2. These transforms are general in that they can be constructed for any graph or any tree. More specifically, general purpose split designs were proposed and other alternative methods were discussed in Section 2.2. Prediction filters were also proposed in Section 2.3 that provide a high degree of data de-correlation for certain classes of signals. Moreover, we proposed a novel update filter design in Section 2.4.2 that makes the LP and HP component orthogonal after each lifting step.

In Chapter 3, general unidirectional transforms and specific transform constructions were then proposed for use in distributed compression WSN. These transforms are able to handle irregularly spaced node locations, and lead to energy-efficient, distributed implementations since they can be computed in a unidirectional manner along a given routing tree, i.e., they can be computed as data is routed toward the sink. It is exactly because of this unidirectionality that the proposed transforms can consistently outperform existing distributed transforms for WSN. Since the transforms are constructed along trees, they provide an easy way into joint optimization

of transform and routing. As such, we also proposed the joint optimization algorithms described in Chapter 4. Note that the lifting transforms we have proposed under this unidirectional framework are only bi-orthogonal. A method for constructing orthogonal unidirectional transforms was provided in Section 3.3.2, but only one particular transform construction was given in Section 3.3.1.

As a final application, edge-adaptive graph-based transforms were proposed for use in image coding. These transforms are edge-adaptive in that they avoid filtering across edges. In particular, edge-avoiding tree-based lifting transforms were constructed for images in Section 5.3. These proved effective tools for coding certain images, e.g., smooth images with very sharp discontinuities, since the number of large high frequency coefficients can be greatly reduced. This led to very efficient image representations which significantly outperform standard transforms. Edge-avoiding graphs were also constructed in Section 5.4 for use in edge-adaptive intra prediction for efficient depth map coding. In particular, we construct graphs that avoid crossing edges, then compute predictions along these graphs. This also allows us to significantly reduce the number of large high frequency coefficients, thereby leading to very efficient representations of depth map images and, ultimately, to higher coding efficiency. Moreover, the compression artifacts around edges (e.g., ringing artifacts) are also significantly reduced, and this leads to better interpolation quality in rendered views.

6.1 Future Work

There are a few interesting directions for future work. One is to design more general types of orthogonal unidirectional transforms. The joint routing and transform optimization algorithm could also be improved by (i) developing methods to perform

the optimization in a distributed manner and (ii) finding better high correlation trees to use in place of the MST for the joint optimization process.

Currently the edge-avoiding tree-based lifting transforms for image coding can only efficiently represent piece-wise smooth images without significant texture. In particular, this method is able to outperform directional transforms for images that have very complex edge structure such as, e.g., depth map images. However, since edge detection typically fails in textured regions, this transform will not provide an efficient representation for textured image regions. In particular, existing directional transforms can efficiently represent oriented textured regions such as stripes. Thus, combining the edge-avoiding lifting transforms with directional transforms could be one new way to achieve gains from both techniques.

The edge-adaptive intra prediction scheme is capable of representing nearly piece-wise constant images. However, since only one predictor is used to predict each pixel, the resulting predictions will not be as accurate for images that have more variation in pixel intensity such as, e.g., piece-wise planar images. This scheme can be improved by using more neighbors for computing predictions. Moreover, both the tree-based lifting transform and the edge-adaptive intra prediction scheme both use edge information to perform edge-avoiding filtering, both could benefit from improved edge coding tools.

Appendix A

Additional Proofs

This appendix contains additional proofs for Chapter 2.

A.1 Proof of Proposition 1

Let $\mathbf{e}_i$ denote the i-th identity vector, i.e., $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0$ for all $j \neq i$. Note that $x(n) = \mathbf{e}_n^t \cdot \mathbf{x}$ and $\sum_{m \in \mathcal{N}_n} \mathbf{p}_n(m) x(m) = \mathbf{p}_n^t \cdot \mathbf{x}$, thus, we can compute the detail coefficient as $d(n) = x(n) - \mathbf{p}_n^t \cdot \mathbf{x} = (\mathbf{e}_n - \mathbf{p}_n)^t \cdot \mathbf{x}$. Thus, if we let $\mathbf{P} = [\mathbf{p}_1\ \mathbf{p}_2\ \ldots\ \mathbf{p}_N]^t$ denote the $N \times N$ prediction matrix and $\mathbf{I}$ be the $N \times N$ identity matrix, we can compute detail coefficients as $\mathbf{d} = (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$. Note that $\mathbf{d}(m) = x(m)$ for $m \in \mathcal{E}$. Furthermore, $x(m) = \mathbf{e}_m^t \cdot \mathbf{x}$ and $\sum_{n \in \mathcal{N}_m} \mathbf{u}_m(n) d(n) = \mathbf{u}_m^t \cdot \mathbf{d} = \mathbf{u}_m^t \cdot (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$, so we can compute the smooth coefficient as $s(m) = x(m) + \sum_{n \in \mathcal{N}_m} \mathbf{u}_m(n) d(n) = (\mathbf{e}_m^t + \mathbf{u}_m^t \cdot (\mathbf{I} - \mathbf{P})) \cdot \mathbf{x}$. Since $m \in \mathcal{E}$, $\text{row}_m(\mathbf{P}) = \mathbf{p}_m^t = \mathbf{0^t}$. Thus, (i) $\mathbf{e}_m^t \cdot (\mathbf{I} - \mathbf{P}) = \mathbf{e}_m^t$, (ii) $\mathbf{e}_m^t + \mathbf{u}_m^t \cdot (\mathbf{I} - \mathbf{P}) = (\mathbf{e}_m + \mathbf{u}_m)^t \cdot (\mathbf{I} - \mathbf{P})$, and (iii) $s(m) = (\mathbf{e}_m + \mathbf{u}_m)^t \cdot (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$. Therefore, if we let $\mathbf{U} = [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_N]^t$ be the $N \times N$ update matrix, the full transform matrix is $\mathbf{T} = (\mathbf{I} + \mathbf{U}) \cdot (\mathbf{I} - \mathbf{P})$ and the transform coefficient vector is $\mathbf{y} = (\mathbf{I} + \mathbf{U}) \cdot (\mathbf{I} - \mathbf{P}) \cdot \mathbf{x}$.

A.2 Proof of Proposition 2

We only prove that $(\mathbf{I}-\mathbf{P})^{-1} = \mathbf{I}+\mathbf{P}$. That $(\mathbf{I}+\mathbf{U})^{-1} = \mathbf{I}-\mathbf{U}$ follows from similar arguments. Let $N_o = |\mathcal{O}|$ and $N_e = |\mathcal{E}|$. Without loss of generality, suppose that $\mathcal{O} = \{1, 2, \ldots, N_o\}$ and $\mathcal{E} = \{N_o + 1, N_o + 2, \ldots, N\}$. Under Definition 1, we have that $\mathbf{p}_n(\mathcal{O}) = 0$ for every $n \in \mathcal{O}$ and $\mathbf{p}_m = \mathbf{0}$ for all $m \in \mathcal{E}$. Thus,

$$\begin{aligned}\mathbf{P} &= \begin{bmatrix} \mathbf{P}(\mathcal{O},\mathcal{O}) & \mathbf{P}(\mathcal{O},\mathcal{E}) \\ \mathbf{P}(\mathcal{E},\mathcal{O}) & \mathbf{P}(\mathcal{E},\mathcal{E}) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} & \mathbf{P}(\mathcal{O},\mathcal{E}) \\ \mathbf{0} & \mathbf{0} \end{bmatrix}.\end{aligned}$$

Thus,

$$\begin{aligned}\mathbf{I} \pm \mathbf{P} &= \begin{bmatrix} \mathbf{I}_{N_o} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{N_e} \end{bmatrix} \pm \begin{bmatrix} \mathbf{0} & \mathbf{P}(\mathcal{O},\mathcal{E}) \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I}_{N_o} & \pm\mathbf{P}(\mathcal{O},\mathcal{E}) \\ \mathbf{0} & \mathbf{I}_{N_e} \end{bmatrix},\end{aligned}$$

where $\mathbf{I}_{N_o}$ and $\mathbf{I}_{N_e}$ denote the $N_o \times N_o$ and $N_e \times N_e$ identity matrices, respectively. Therefore, $(\mathbf{I}+\mathbf{P}) \cdot (\mathbf{I}-\mathbf{P}) = (\mathbf{I}-\mathbf{P}) \cdot (\mathbf{I}+\mathbf{P}) = \mathbf{I}$.

A.3 Proof of Proposition 4

We first prove Lemma 1, then use it to prove Proposition 4.

Lemma 1 (Orthogonality of the Dual Basis)**.** *Let K, L and N be non-negative integers such that $K + L = N$. Let $\mathcal{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_K\}$ be a set of linearly independent vectors in $\mathbb{R}^N$ and let $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L\}$ be a set of linearly independent vectors orthogonal to vectors in $\mathcal{A}$, i.e., $< \mathbf{a}_i, \mathbf{b}_j >= 0$ for all i and j. Define*

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^t \\ \mathbf{a}_2^t \\ \vdots \\ \mathbf{a}_K^t \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{b}_1^t \\ \mathbf{b}_2^t \\ \vdots \\ \mathbf{b}_L^t \end{bmatrix} \text{ and } \mathbf{C} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}.$$

Let $\mathbf{C}^{-1} = [\tilde{\mathbf{c}}_1 \;\; \tilde{\mathbf{c}}_2 \;\; \ldots \;\; \tilde{\mathbf{c}}_N]$. Then

$$\mathbf{C}\mathbf{C}^t = \begin{bmatrix} \mathbf{A}\mathbf{A}^t & \mathbf{0} \\ \mathbf{0}^t & \mathbf{B}\mathbf{B}^t \end{bmatrix},$$

where $\mathbf{0}$ is the $K \times L$ zero matrix, and moreover,

$$(\mathbf{C}^{-1})^t\mathbf{C}^{-1} = \begin{bmatrix} (\mathbf{A}\mathbf{A}^t)^{-1} & \mathbf{0} \\ \mathbf{0}^t & (\mathbf{B}\mathbf{B}^t)^{-1} \end{bmatrix},$$

Therefore, we also have a similar orthogonality relationship between the dual basis vectors, i.e., $< \tilde{\mathbf{c}}_i, \tilde{\mathbf{c}}_j >= 0$ for any $i = 1, 2, \ldots, K$ and $j = K+1, K+2, \ldots, N$.

Proof. Since every vector in $\mathcal{A}$ is orthogonal to every vector in $\mathcal{B}$, the vectors in $\mathcal{A}$ are linearly independent of the vectors in $\mathcal{B}$. Therefore, $\mathbf{C}$ has N linearly independent columns and $\mathbf{C}^{-1}$ exists. Moreover, $\mathbf{A}\mathbf{B}^t = \mathbf{0}$ and $\mathbf{B}\mathbf{A}^t = \mathbf{0}^t$. Therefore,

$$\mathbf{C}\mathbf{C}^t = \begin{bmatrix} \mathbf{A}\mathbf{A}^t & \mathbf{0} \\ \mathbf{0}^t & \mathbf{B}\mathbf{B}^t \end{bmatrix}.$$

$\mathbf{A}^t$ has linearly independent columns, so for any $\mathbf{x} \neq \mathbf{0}$, $\mathbf{A}^t\mathbf{x} \neq \mathbf{0}$. Thus, $\mathbf{x}^t\mathbf{A}\mathbf{A}^t\mathbf{x} > 0$, which implies that $\mathbf{A}\mathbf{A}^t$ is positive definite. Similarly, $\mathbf{B}\mathbf{B}^t$ is positive definite. By basic matrix properties $(\mathbf{C}^{-1})^t\mathbf{C}^{-1} = (\mathbf{C}^t)^{-1}\mathbf{C}^{-1} = (\mathbf{C}\mathbf{C}^t)^{-1}$, thus

$$(\mathbf{C}^{-1})^t\mathbf{C}^{-1} = \begin{bmatrix} (\mathbf{A}\mathbf{A}^t)^{-1} & \mathbf{0} \\ \mathbf{0}^t & (\mathbf{B}\mathbf{B}^t)^{-1} \end{bmatrix}.$$

This implies that $< \tilde{\mathbf{c}}_i, \tilde{\mathbf{c}}_j >= 0$ for any $i \leq K$ and $j \geq K+1$. □

The proof of Proposition 4 is as follows. Let $N_o = |\mathcal{O}|$ and $N_e = |\mathcal{E}|$. The wavelet coefficient vector is $\mathbf{w} = (\mathbf{I}+\mathbf{U})(\mathbf{I}-\mathbf{P})\mathbf{x} = \mathbf{T}\mathbf{x}$. Since $\mathbf{T}$ is invertible, $\mathbf{T}^{-1}$ exists and so

$$\begin{aligned} \mathbf{x} &= \mathbf{T}^{-1}\mathbf{w} \\ &= \sum_{i=1}^{N} \mathbf{w}(i)\tilde{\mathbf{t}}_i \\ &= \sum_{i=1}^{N} < \mathbf{t}_i, \mathbf{x} > \tilde{\mathbf{t}}_i \\ &= \sum_{i\in\mathcal{E}} < \mathbf{t}_i, \mathbf{x} > \tilde{\mathbf{t}}_i + \sum_{j\in\mathcal{O}} < \mathbf{t}_j, \mathbf{x} > \tilde{\mathbf{t}}_j \\ &= \mathbf{x}_e + \mathbf{x}_o \end{aligned}$$

Without loss of generality, suppose that $\mathcal{O} = \{1, 2, \ldots, N_o\}$ and $\mathcal{E} = \{N_o+1, N_o+2, \ldots, N\}$. Now define

$$\mathbf{A} = \begin{bmatrix} \mathbf{t}_1^t \\ \mathbf{t}_2^t \\ \vdots \\ \mathbf{t}_{N_o}^t \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{t}_{N_o+1}^t \\ \mathbf{t}_{N_o+2}^t \\ \vdots \\ \mathbf{t}_N^t \end{bmatrix} \text{ and } \mathbf{T} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix}.$$

Proposition 3 implies that $\mathbf{AB}^t = \mathbf{0}$ and $\mathbf{BA}^t = \mathbf{0}^t$ and Lemma 1 implies that

$$(\mathbf{T}^{-1})^t\mathbf{T}^{-1} = \begin{bmatrix} (\mathbf{AA}^t)^{-1} & \mathbf{0} \\ \mathbf{0}^t & (\mathbf{BB}^t)^{-1} \end{bmatrix}.$$

Therefore, $< \tilde{\mathbf{t}}_i, \tilde{\mathbf{t}}_j >= 0$ for all $i \in \mathcal{O}$ and $j \in \mathcal{E}$. Moreover, $< \mathbf{x}_e, \mathbf{x}_o >= 0$.

Bibliography

[1] J. Acimovic, B. Beferull-Lozano, and R. Cristescu. Adaptive distributed algorithms for power-efficient data gathering in sensor networks. *Intl. Conf. on Wireless Networks, Comm. and Mobile Computing*, 2:946–951, June 2005.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[3] R. Baraniuk, A. Cohen, and R. Wagner. Approximation and compression of scattered data by meshless multiscale decompositions. *Applied Computational Harmonic Analysis*, 25(2):133–147, September 2008.

[4] C. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, August 2003.

[5] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *IEEE International Conference on Data Engineering (ICDE)*, pages 3–7. IEEE, 2006.

[6] A. Ciancio. *Distributed Wavelet Compression Algorithms for Wireless Sensor Networks*. PhD thesis, University of Southern California, 2006.

[7] A. Ciancio and A. Ortega. A flexible distributed wavelet compression algorithm for wireless sensor networks using lifting. In *Proc. of ICASSP'04*, 2004.

[8] A. Ciancio and A. Ortega. Distributed wavelet compression algorithm for wireless multihop sensor networks based on lifting. In *Proc. of ICASSP'05*, 2005.

[9] A. Ciancio, S. Pattem, A. Ortega, and B. Krishnamachari. Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet compression algorithm. In *Proc. of IPSN'06*, April 2006.

[10] I. Cidon and M. Sidi. Distributed assignment algorithms for multi-hop packet-radio networks. *IEEE Transactions on Computers*, 38(10), October 1989.

[11] R.L. Claypoole, G.M. Davis, W. Sweldens, and R.G. Baraniuk. Nonlinear wavelet transforms for image coding via lifting. *IEEE Transactions on Image Processing*, 12(12):1449–1459, December 2003.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[13] R. Cristescu, B. Beferull-Lozaon, and M. Vetterli. Networked Slepian-Wolf: Theory, algorithms, and scaling laws. *IEEE Transactions on Information Theory*, 51(12):4057–4073, December 2005.

[14] D. Estrin, D. Ganesan, S. Ratnasamy, and H. Wang. Coping with irregular spatio-temporal sampling in sensor networks. *ACM SIGCOMM Comput. Commun. Rev.*, 34(1):125–130, January 2004.

[15] M. Gastpar, P. Dragotti, and M. Vetterli. The distributed Karhunen-Loève transform. *IEEE Transactions on Information Theory*, 52(12):5177–5196, December 2006.

[16] B. Girod and S. Han. Optimum update for motion-compensated lifting. *IEEE Signal Processing Letters*, 12(2):150–153, February 2005.

[17] V.K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5):9–21, September 2001.

[18] R. Gummadi, X. Li, R. Govindan, C. Shahabi, and W. Hong. Energy-efficient data organization and query processing in sensor networks. *SIGBED Review*, 2(1), January 2005.

[19] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. Technical Report arXiv:0912.3848, Dec 2009.

[20] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 4th edition, 2004.

[21] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient routing protocols for wireless microsensor networks. In *Proc. of Hawaii Intl. Conf. on Sys. Sciences*, January 2000.

[22] A.K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[23] M. Jansen, G. Nason, and B. Silverman. Scattered data smoothing by empirical Bayesian shrinkage of second generation wavelet coefficients. In *Wavelets: Applications in Signal and Image Processing IX, Proc. of SPIE*, 2001.

[24] JBIG. Progressive Bi-level Image Compression. In *ISO/IEC ITU Recommendation T.82*, 1993.

[25] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer-Verlag Press, 2nd edition, 2004.

[26] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila. Depth map distortion analysis for view rendering and depth coding. In *Proc. of ICIP'09*, 2009.

[27] W.-S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila. Depth map coding with distortion estimation of rendered view. In *Proc. of VIPC'10*, 2010.

[28] J. Kovacevic and M. Vetterli. *Wavelets and Subband Coding*. Prentice Hall, 1995.

[29] P. Lai, A. Ortega, C. Dorea, P. Yin, and C. Gomila. Improving view rendering quality and coding efficiency by suppressing compression artifacts in depth-image coding. In *Proc. of VCIP'09*, 2009.

[30] S. Li and W. Li. Shape-adaptive discrete wavelet transforms for arbitrarily shaped visual object coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(5):725–743, August 2000.

[31] H. Luo, Y.C. Tong, and G. Pottie. A two-stage DPCM scheme for wireless sensor networks. In *Proc. of ICASSP'05*, March 2005.

[32] M. Maitre and M.N. Do. Joint encoding of the depth image based representation using shape-adaptive wavelets. In *Proc. of ICIP'08*, 2008.

[33] S. Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 2nd edition, 1999.

[34] K. Mechitov, W. Kim, G. Agha, and T. Nagayama. High-frequency distributed sensing for structure monitoring. In *In Proc. First Intl. Workshop on Networked Sensing Systems (INSS)*, 2004.

[35] Y. Morvan, P.H.N. de With, and D. Farin. Platelet-based coding of depth maps for the transmission of multiview images. volume 6055. SPIE, 2006.

[36] S.K. Narang and A. Ortega. Lifting based wavelet transforms on graphs. In *To Appear in Proc. of Asia Pacific Signal and Information Processing Association (APSIPA)*, October 2009.

[37] S.K. Narang, G. Shen, and A. Ortega. Unidirectional graph-based wavelet transforms for efficient data gathering in sensor networks. In *Proc. of ICASSP'10*, March 2010.

[38] H. M. on Great Duck Island. Online data-set located at http://www.greatduckisland.net.

[39] A.V. Oppenheim and R.W. Schafer. *Discrete-time Signal Processing*. Prentice Hall, 3rd edition, 2009.

[40] S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. *ACM Transactions on Sensor Networks*, 4(4):60–66, August 2008.

[41] S. Pattem, G. Shen, Y. Chen, B. Krishnamachari, and A. Ortega. Senzip: An architecture for distributed en-route compression in wireless sensor networks. In *Workshop on Sensor Networks for Earth and Space Science Applications (ESSA)*, April 2009.

[42] W.B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[43] E. Le Pennec and S. Mallat. Sparse geometric image representations with bandelets. *IEEE Transactions on Image Processing*, 14(4):423– 438, April 2005.

[44] G. Peyre and S. Mallat. Surface compression with geometric bandelets. In *SIGGRAPH '05*, 2005.

[45] S.S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed compression in a dense microsensor network. *IEEE Signal Processing Magazine*, pages 51–60, March 2002.

[46] J.G. Proakis, E.M. Sozer, J.A. Rice, and M. Stojanovic. Shallow water acoustic networks. *IEEE Communications Magazine*, 39(11):114–119, 2001.

[47] K.R. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Academic, 1990.

[48] P. Rickenbach and R. Wattenhofer. Gathering correlated data in sensor networks. In *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing*, October 2004.

[49] A. Said and W.A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[50] A. Sanchez, G. Shen, and A. Ortega. Edge-preserving depth-map coding using graph-based wavelets. In *Proc. of Asilomar'09*, 2009.

[51] G. Shen, W.-S. Kim, A. Ortega, J. Lee, and H.C. Wey. Edge-aware intra prediction for depth-map coding. In *To Appear in Proc. of ICIP'10*, September 2010.

[52] G. Shen, S. Narang, and A. Ortega. Adaptive distributed transforms for irregularly sampled wireless sensor networks. In *Proc. of ICASSP'09*, April 2009.

[53] G. Shen and A. Ortega. Compact image representation using wavelet lifting along arbitrary trees. In *Proc. of ICIP'08*, October 2008.

[54] G. Shen and A. Ortega. Joint routing and 2D transform optimization for irregular sensor network grids using wavelet lifting. In *IPSN '08*, April 2008.

[55] G. Shen and A. Ortega. Optimized distributed 2D transforms for irregularly sampled sensor network grids using wavelet lifting. In *Proc. of ICASSP'08*, April 2008.

[56] G. Shen and A. Ortega. Tree-based wavelets for image coding: Orthogonalization and tree selection. In *Proc. of PCS'09*, May 2009.

[57] G. Shen and A. Ortega. Transform-based distributed data gathering. *IEEE Transactions on Signal Processing*, 58(7):3802–3815, July 2010.

[58] G. Shen, S. Pattem, and A. Ortega. Energy-efficient graph-based wavelets for distributed coding in wireless sensor networks. In *Proc. of ICASSP'09*, April 2009.

[59] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G.B. Akar, G. Triantafyllidis, and A. Koz. Coding algorithms for 3DTV: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(11):1606–1621, Nov. 2007.

[60] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5), October 2000.

[61] A. Sridharan and B. Krishnamachari. Max-min fair collision-free scheduling for wireless sensor networks. In *Proc. of IEEE IPCCC Workshop on Multi-hop Wireless Networks*, April 2004.

[62] H. Stark and J.W. Woods. *Probability and Random Processes with Applications to Signal Processing*. Prentice Hall, 3rd edition, 2002.

[63] G. Strang. *Linear Algebra and its Applications*. Thomson Learning, 3rd edition, 1988.

[64] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1997.

[65] W. Sweldens. The lifting scheme: A construction of second generation wavelets. Tech. report 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, 1995.

[66] M. Tanimoto, T. Fujii, and K. Suzuki. View synthesis algorithm in view synthesis reference software 2.0(VSRS2.0). ISO/IEC JTC1/SC29/WG11, Feb. 2009.

[67] D. Taubman and D. Marcellin. *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 1st edition, 2001.

[68] TinyOS-2. Collection tree protocol. http://www.tinyos.net/tinyos-2.x/doc/.

[69] D. Tulone and S. Madden. PAQ: Time series forecasting for approximate query answering in sensor networks. In *Proceedings of the European Conference in Wireless Sensor Networks (EWSN)*, pages 21–37. IEEE, February 2006.

[70] G. Valiente. *Algorithms on Trees and Graphs*. Springer, 1st edition, 2002.

[71] V. Velisavljevic, B. Beferull-Lozano, M. Vetterli, and P.L. Dragotti. Directionlets: Anisotropic multidirectional representation with separable filtering. *IEEE Transactions on Image Processing*, 15(7):1916– 1933, July 2006.

[72] R. Wagner, R. Baraniuk, S. Du, D.B. Johnson, and A. Cohen. An architecture for distributed wavelet analysis and processing in sensor networks. In *IPSN '06*, April 2006.

[73] R. Wagner, H. Choi, R. Baraniuk, and V. Delouille. Distributed wavelet transform for irregular sensor network grids. In *IEEE Stat. Sig. Proc. Workshop (SSP)*, July 2005.

[74] A. Wang and A. Chandraksan. Energy-efficient DSPs for wireless sensor networks. *IEEE Signal Processing Magazine*, 19(4):68–78, July 2002.

[75] R. Willett and R. Nowak. Platelets: A multiscale approach for recovering edges and surfaces in photon-limited medical imaging. *IEEE Transactions on Medical Imaging*, 22(3):332–350, March 2003.

[76] K. Yamamoto, M. Kitahara, H. Kimata, T. Yendo, T. Fujii, M. Tanimoto, S. Shimizu, K. Kamikura, and Y. Yashima. Multiview video coding using view interpolation and color correction. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(11):1436–1449, Nov. 2007.

[77] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM '02*, 2002.

[78] B. Zeng and J. Fu. Directional discrete cosine transforms for image coding. In *Proc. of ICME'06*, 2006.

[79] Y. Zhu, K. Sundaresan, and R. Sivakumar. Practical limits on achievable energy improvements and useable delay tolerance in correlation aware data gathering in wireless sensor networks. In *IEEE SECON'05*, September 2005.

[80] L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Transactions on Graphics*, 23(3), Aug. 2004.