DISTRIBUTED WAVELET COMPRESSION ALGORITHMS

FOR WIRELESS SENSOR NETWORKS

by

Alexandre Gomes Ciancio

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

December 2006

Acknowledgments

First of all, I would like to thank my advisor Antonio Ortega, whom I had the pleasure and the privilege to work with during the past five years. He was nothing less than a great person since the first moment I met him. He will serve as a reference in the next steps of my career.

To the "*Perdidos em L.A.*": André, Bob, Denis, Flávia, Glauco, Guilherme, João, Juan, Lisandra, M1, M2, Madhavan, Sandra. You became a family away from Brazil, and made my experience in the U.S. much more enjoyable. To the good friends Heather and Rob and the lovely Sarah Jane and Emma Mae, and their family, for their friendship and love.

I cannot also thank enough to my family in Brazil: my parents Valéria and Sergio, my grandmother Vilma, my sister Alessandra for the unmeasurable support.

Finally, to my lovely wife Cristine, for all her unconditional love and support. This thesis is dedicated to you.

Table of Contents

List Of Tables

List Of Figures

Abstract

We address the problem of compression for wireless sensor networks from a signal processing point of view. Each of the sensors has limited power, and acquires data that should be sent to a central node. The final goal is to have a reconstructed version of the sampled field at the central node, with the sensors spending as little energy as possible.

We propose two distributed wavelet compression algorithms for multihop, distributed sensor networks based on the lifting scheme. The first algorithm introduces extra transmissions in the network so as to give the nodes access to neighboring data, making it possible to compute the transform coefficients. The second algorithm exploits the natural data flow in the network to aggregate data by computing partial wavelet coefficients that are refined as the data flows towards the central node. We study the impact of quantization of partial data on the final distortion obtained, and propose a rule to determine how many bits should be used to quantize the partial information so as to achieve a target level of degradation, in the form of added distortion as compared to calculating coefficients without partially quantized data. We also introduce a framework where the network is represented as a graph, with sensors associated to nodes, transmission costs to edge weights, and different coding schemes associated to modes of operation. Dynamic programming techniques are used to optimize the network, assigning coding schemes to

each of the nodes so that the overall energy cost is minimum. The proposed coding methods and optimization are extended to bidimensional networks, with irregular sensor deployment. In this scenario, our algorithm operates by first selecting a routing strategy through the network. Then, for each route, an optimal combination of data representation algorithms is selected. We then propose a simple heuristic to determine the data representation technique to use once path merges are taken into consideration.

Chapter 1

Introduction

A Sensor Network (SN) is a set of nodes with sensing, communication and processing capabilities. One of the first applications using SN dates back to the early 1950s and involved deploying acoustic sensors at the ocean bottom to detect and track submarines. Nowadays, with the advance of technology, low cost programmable devices can be deployed in the environment and coordinated to perform a variety of tasks, providing unprecedented opportunities for instrumenting and controlling buildings, cities and the environment. There is a huge potential market for embedded network devices in vehicles, mobile phones, and even personal appliances. Some applications involving sensor networks might include [3]:

- *Object detection/tracking.* A network of sensors acquires and processes acoustic and/or visual data to detect and track objects in the environment for monitoring or surveillance purposes.

- *Environment Monitoring.* Sensors deployed in an area affected by chemical agents monitor the agent dispersion and effects on the environment; weather-related measurements can be obtained by a network of sensors and transmitted to a processing station.

- *Traffic control.* Sensors coordinate to operate traffic lights based on vehicle monitoring at intersections.

While wireless networks offer huge mobility and versatility, they face some serious obstacles when compared to wired networks. Physical disadvantages might include that, typically, there is a one order of magnitude difference between the data rates for wireless and wired networks, and that wireless sensors have higher error rates and rely on batteries for power supply, making energy consumption a serious issue directly related to the network survivability [30]. Algorithms that increase the efficiency and survivability of such networks will offer, therefore, a major contribution for the even faster development of any sensor network related application. Design disadvantages include the development of decentralized versions for well known algorithms, having all the wireless network features/limitations in mind.

In the following sections we describe the problem addressed by this work and discuss some relevant work done in the area and some of the challenges that distributed algorithms face in a sensor network scenario. We end the chapter with an outline for the dissertation.

1.1 Problem Description

Assume that a number of wireless, power-constrained sensors are spread over an area and are acquiring spatially correlated data. All data measurements are sent to a central node, or sink, where a reconstructed version of the data in all sensors should be made available based on the other sensor transmissions (Figure 1.1). Communication is done via data

hoping, where data from each sensor is forwarded (using other sensors as relay stations) until it reaches the sink. Such a network is referred to as a multihop network.

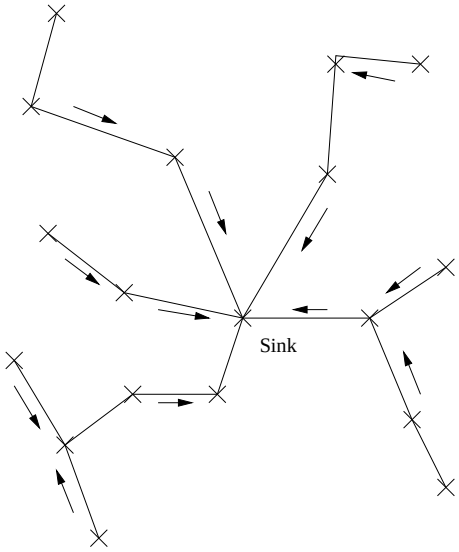


Figure 1.1: Wireless sensors acquire data and forward it to a central station, or sink.

A simple and naive design would be for each sensor to just transmit a quantized version of its own measurement to the central node. However, this approach would not exploit the fact that measurements originated from spatially close sensors are likely to be correlated, and energy would be wasted with the transmission of redundant data to the central node. As an alternative, since data is correlated, it would be reasonable to use some sort of transform as a means to decorrelate the information from sensors, and, therefore, represent the measurements in a more efficient way, using fewer bits. Such a problem has been extensively addressed by standard signal processing techniques, and there are many efficient and well known algorithms to decorrelate and compress data [33, 17]. However, in a wireless network, sensors have access only to their own data, and moving data around the network requires additional energy consumption. Using

transforms to exploit spatial correlation requires inter-sensor communication, so nodes would have access to the data necessary to compute the transform coefficients. This "distributed" approach to transform computation means that power will be consumed both for local processing and for *additional transmission* of information to other sensors. In this work, we use the term *additional*, or *extra transmission* to refer to any transmission that would not be required if all the measurements were simply forwarded to the sink (no data processing). To illustrate this trade-off, consider the effect of choosing transforms of different sizes. In general, larger transforms will tend to provide better decorrelation, but at the expense of added communication cost between sensors. For example, using a block transform of size N would mean that N sensors would need to exchange information, with an average communication distance greater than for, say, an $N/2$ size transform.

Thus, a strategy to design an efficient algorithm may be to spend extra energy with additional local communications, i.e. data exchange communications among a reduced number of neighboring sensors, in the hope that the obtained decorrelated data can be represented more efficiently (requiring fewer bits, and therefore less energy to be transmitted), with an overall energy cost smaller than the case where data is sent without being processed.

In this work, we tackle the problem of compression for sensor networks from a signal processing point of view, and propose two distributed wavelet transform algorithms as a means to decorrelate data, while taking the energy cost into account. We evaluate the performance of these algorithms not only based on signal-to-noise ratio, but also on energy consumption, as compared to non-processed data transmission and DPCM encoding.

1.2 Related Work

The fast development and decreasing cost of wireless technologies have made sensor networks a key technology for the future, and led to substantial interest on distributed transform algorithms targeted at these networks. To better situate our work, we briefly present in this section the main idea behind some relevant work that has addressed the problem of compression for sensor networks.

For the ideal decorrelation case, a Karhunen-Loève transform (KLT) could be computed. In [16], distributed approximations for the KLT are proposed and analyzed. The algorithm consists in computing *partial* KLTs inside subsets of data and/or combining them using a *conditional* KLT, where some sources act as side information to other subsets. The KLT still requires each of the sensors inside a subset to have knowledge about *all* the measurements inside that block, potentially increasing communication costs in a sensor network scenario. While the KLT is flexible in the sense that different block sizes can be chosen, the number of inter-sensor communications could still be too high. It is interesting to note, however, that the block-transform approach proposed in [16] can be extended to other distributed techniques as well. For example, the nodes in a sensor network could be divided into subsets, or blocks, and the Partial Coefficient Approach proposed in Section 2.5 could be applied independently to each of the blocks. The partially computed data generated from each block could be combined using, for example, techniques as the one described in [24].

In [35, 36] decorrelation is achieved by means of a distributed wavelet algorithm. However, the dependency between inter-sensor transmission costs and their distance is

not considered, and, depending on the number of stages of the wavelet decomposition, measurements from a sensor that is far apart might be needed, elevating power consumption. Also, in the performance analysis for that algorithm, a very restrictive assumption is made, namely, that all the information of interest is located in the low-pass subband. While this may be true for some signals, in general, not representing the high-pass information leads to a loss in representation accuracy.

One idea, proposed in [25], was based in the use of coset codes to decorrelate data. In their problem, they consider the case where X and Y are correlated discrete-alphabet independent identically distributed sources, and the goal is to compress X losslessly, with Y being known at the decoder, but not at the encoder. The idea is to partition the space of all outcomes of the source into sets (called cosets) such that the minimum distance between any two codevectors in any coset is "large" enough for a given metric. The encoder saves rate by sending only the index of the coset containing the outcome. The decoder recovers the outcome of X by searching through the coset whose index is received. The search is for that codevector which is "closest" (in the metric) to the outcome Y. Such an encoding strategy did not require inter-node communication. However, although no extra energy is spent with local communications, overall performance is still limited by the predictive nature of the algorithm and might not be as good as a transform-based method, due to the inherent limitations of predictive-based coding schemes. One possible drawback of such a system would be that if the actual correlation between two given sensors is below that the one used in designing the corresponding Wyner-Ziv encoders, then the resulting decoding will have higher distortion, which can then propagate the error to neighboring sensors. We discuss DPCM-like schemes in more detail in Section 2.7.

In [11, 12], the authors propose a method to minimize the transmission costs for data gathering applications in sensor networks by finding the routing strategy that leads to best decorrelation performance. They conclude that due to data correlation, routing affects the rate at each node, and optimization has to jointly consider rate allocation and tree building (routing). They use a joint-entropy model for data compression, and assume only local data is encoded at the nodes, using received data. The correlated data gathering problem and the need for jointly optimizing rate allocation at the nodes and routing structure is also considered in [29], where the authors compare strategies for compression-driven routing and routing-driven compression, and explore compression at several hops and only at cluster heads and conclude that there exist efficient correlation independent routing structures. While the work presented in this thesis proposes a more practical distributed encoding algorithm (as opposed to assume joint-entropy models), simulations also show (see Chapter 5) that different routing strategies affect algorithm performance, and that an optimal routing strategy should jointly consider the algorithms individual decorrelation properties.

We can roughly divide related work in two main categories. First there is research more concerned with algorithms, which proposes distributed ways to decorrelate data in a network, such as [16, 25, 35, 36]. Second, we have a more system-oriented work, addressing issues such as performance bounds, compression efficiency and cost impact in sensor networks, as for example in [11, 12, 29]. There is, however, a gap between these two sets of studies in the sense that the first typically lacks a thorough analysis of network related issues, more specifically the impact of routing and energy consumption. These issues are better addressed by the second set of studies, which in turn lacks a more

detailed description of node operations in a practical environment. One of the goals of this thesis is to propose a cost-aware distributed transform, where we seek to combine ideas from both algorithm and system-related research.

1.3 Challenges

Regardless of the application, at a high level, a sensor network can be viewed as the process of sampling data at locations in a field with the objective of performing further processing on the data. This processing might consist of filtering, feature extraction, or any other data manipulation required by the application. However, in general, only non-distributed versions of the algorithms implemented are known, and a distributed version, which might impose many other constraints, has to be developed. To illustrate this point, consider a simple filtering example. In a non-distributed scenario all data is available for computation and standard filtering operations can be performed. In a distributed scenario, data is not globally available, and each data exchange implies in transmission. Thus, an energy-constrained network might require new, lower energy methods to compute convolution. The non-distributed approach to compute a filtering operation in a network could require a large number of data exchanges, becoming too expensive energy-wise (due to the increase in the number of transmissions), and reducing the life of the network considerably. Although this example highlights transmission and energy issues, other problems might include impact of finite-length representation, effects of routing and network protocols in the standard algorithm, etc.

Two important challenges that a distributed algorithm for the wavelet transform faces are anticausality and finite-precision effects. Consider a sensor network as depicted in

Figure 1.2, with each node representing a sampling point in space. Figure 1.2(a) shows a simple scenario, where no processing is required. The white arrows represent a typical shortest-path route to the sink. Data from the nodes is simply forwarded as indicated by the arrows. Figure 1.2(b) shows the communication along the path from one leaf node to the sink. In Figure 1.2(c), a simple filtering operation is being implemented in the same network. If the convolution sum requires each node to have knowledge of, say, its two immediate neighbors, then additional transmissions (black arrows) will be necessary (compared to Figure 1.2(a)). In the sensor network scenario, since data points are mapped to physical positions in space, and filtering operations occur along routes connecting nodes to the sink, anticausality is meant in the *spatial* sense. In other words, an anticausal filter requires sensors to transmit data "backwards", i.e., away from the sink along a given route (see Figure 1.2(d)), the same way an anticausal filter in time domain would require knowledge of a "future" data point. In a multihop network, where all data already flows in a particular direction, sensors that need to transmit data away from the sink might be wasting resources. A simple solution could be to have the sensors calculating the transform coefficients only after all the necessary unprocessed (raw) data arrive at a future node, in what would be the equivalent to the introduction of a delay in time domain. This approach, however, is clearly inefficient. Since raw data has to be transmitted until it reaches the node that will process it, and the rate allocation for unprocessed data is typically much larger than for transform coefficients, there can be potentially large increases in the energy consumption due to the raw data transmission itself.

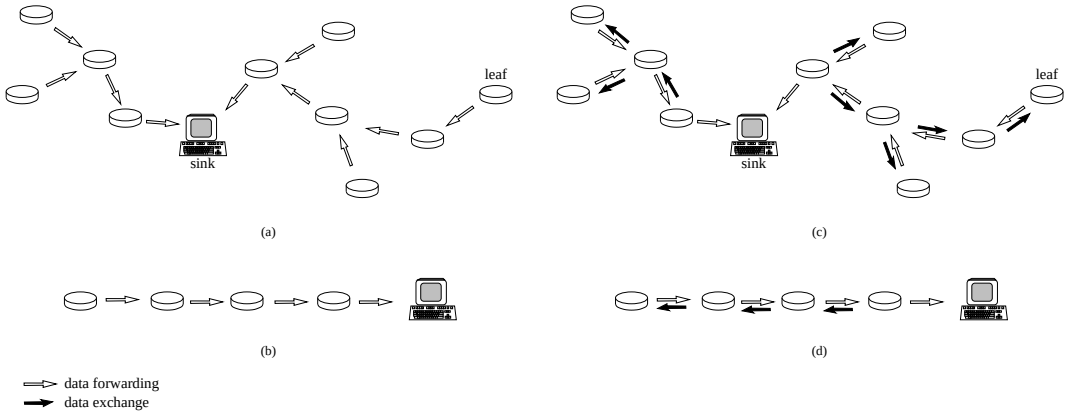


Figure 1.2: (a,b) Data forwarding in a sensor network. (c,d) Data exchange (required for convolution) and forwarding in the same network for a simple filtering operation.

Another important challenge is that all transmissions have to be made with finite precision, which might considerably affect the final distortion. Data transmitted back and forth over the network has to be quantized, since transmissions at full precision might substantially increase energy consumption, affecting the final performance. If data is being aggregated as it is transmitted around the network, being by a transform computation, or by any other method, a careful evaluation of the effects of quantization of partially aggregated data becomes significantly important.

Besides the inherent challenges related to the distributed compression algorithms themselves, other issues related to practical sensor deployment may need to be addressed. For example, wavelet functions are traditionally defined as the dyadic translates and dilates of one particular function, the mother wavelet. These wavelets are sometimes referred to as *first generation wavelets* [38]. However, realistic sensor placements are typically random, implying the irregular sampling of the underlying data field. First generation wavelets, more specifically translation and dilation, cannot be maintained in the irregular

sampling setting. While a first generation wavelet can still be used to represent data sampled irregularly (by simply assuming the data points came from a regular sampling), its performance is most likely to be worse than the case where the same data field is sampled regularly. In Section 2.8 we address in more detail the irregular sampling problem.

1.4 Outline and Contributions of This Thesis

The main contributions of this thesis are:

- *Distributed Wavelet Transform.* We propose two novel distributed algorithms for the discrete wavelet transform, that are designed with the main goal of reducing the energy consumption in the network. The first one introduces extra communication between sensors to acquire data necessary for the coefficient computation; the second exploits the natural data flow in the network, and introduces the concept of partial coefficient computation, eliminating the need for additional transmissions. We also address the issue of irregular sampling, and propose simple modifications to the wavelet algorithm to improve its performance for the random node placement case.

- *Quantization Effects Analysis.* We analize the impact of quantization of partially processed data that is transmitted between sensors on the final distortion, as compared to the case where the partially computed data is computed at full precision, and propose a rule to design sensor quantizers that can be used to find a good trade-off point between extra distortion and energy cost.

- *Network Optimization Using Dynamic Programming.* We propose a framework where a sensor network that can choose among a number of coding schemes to

encode data can be described as a graph, with transmission costs associated to edge weights. We describe how dynamic programming techniques can be used to assign coding schemes to the sensors such that the overall energy consumption is minimized.

- *2D Extension.* We extend the proposed distributed algorithms to operate in bidimensional networks, with no constraints regarding the network topology. We also propose the use of alternate routing strategies to help improve the energy consumption performance. The intuition behind this point is that a routing strategy that connects nodes that are more correlated (but not necessarily physically close to each other) can improve the transform performance, which will then require fewer bits to represent the data, reducing the final energy consumption.

The rest of this thesis is organized as follows: in Chapter 2 we describe the proposed algorithms and propose a modification to the lifting implementation of the wavelet transform to improve its performance when data is irregularly sampled; in Chapter 3 we address the issue of quantization effects on the partial coefficients, and propose a general rule for bit allocation to partial coefficients; in Chapter 4 we describe how the network can be optimized in terms of energy consumption by using a dynamic programming approach to assign different tasks (coding schemes) to sensors based on their physical location relative to each other and to the sink; in Chapter 5 we extend the proposed algorithms to operate in bidimensional networks, providing an extensive performance analysis for a variety of situations, including different routing strategies and network topologies, using both

simulated and real data; in Chapter 6 we present some conclusions, and discuss future work.

Chapter 2

Distributed Wavelet Transform

In this chapter we introduce the main issues arising in a distributed implementation of the wavelet transform,. We also introduce the polyphase representation of wavelets, the lifting algorithm, and two novel compression algorithms that can be applied to wireless sensor networks.

2.1 Introduction

As mentioned in Section 1.3, adapting a standard algorithm to operate in a distributed manner is not a trivial task in general, and must take into account the network requirements and limitations. More specifically, a distributed *wavelet* algorithm must consider a number of issues:

- **Cost.** Ultimately, one of the major concerns of a wireless sensor network is its energy limitations. Whatever the application may be (data compression, query retrieval, etc.), a distributed algorithm must try to be as efficient as possible while spending as little energy as possible.

- **Simplicity.** Simple algorithms lead to a reduced number of operations, lowering processing time and energy consumption at the sensor.

- **Anticausality.** As discussed in Section 1.3, in a multihop network, anticausal filters lead to additional, against the flow transmissions. Large filters, or higher levels of wavelet decomposition might require a large number of extra communication. The algorithm must be able to decide if the extra cost required by communication/processing will still lead to a sufficient decorrelation of the data such that the overall energy consumption in the network is reduced.

- **Finite Precision.** Since transmissions between sensors made with infinite precision incur in prohibitive energy cost, the impact on the final performance of quantization of each transmitted data packet must be considered.

Among the methods available for computing the wavelet transform, the lifting scheme [38, 39] deserves some attention. The lifting factorization provides a convenient representation of the wavelet transform as it assumes in place computation (each sensor represents a single memory location), and explicitly breaks down the transform into elementary operations that can be easily evaluated in terms of communication costs. It offers a number of advantages when compared to the traditional implementation of the wavelet transform:

1. It allows a faster implementation of the wavelet transform (but still $O(n)$). In some cases the operations might be reduced by a factor of two;

2. Full in-place computation of transform coefficients, meaning that no auxiliary memory needs to be allocated;

3. The inverse transform is trivial to find.

In the sensor network scenario, the advantages mentioned above directly contribute to higher efficiency (fewer operations, less energy consumption) and lower cost (less memory

requirements), making the lifting scheme a very promising approach when trying to design a distributed compression algorithm as a means to decorrelate sensor data.

In the first algorithm we propose, we make use of two-way communications between neighboring sensors to mimic the lifting implementation of the wavelet transform. The lifting factorization of the filters provides a step-by-step approach for the computation of the transform coefficients at the network nodes. In the second one, we exploit the natural data flow in the network and the characteristics of lifting in order to aggregate data by computing partial wavelet coefficients that are refined as data flows toward the central node, eliminating backward (against the flow) transmissions.

By operating at the best trade-off point between processing and transmission cost, given the network requirements, our system could be optimized to reduce the overall energy consumption. This can be achieved by dividing the network into groups of sensors, and assigning different tasks to each group. For instance, some nodes might be much closer to the central node than others. That group of nodes would be a good candidate to perform direct transmission instead of a distributed transform, depending on where the trade-off point is. Nodes that are far from the sink could benefit from coding schemes with better compression efficiency, since the number of bits to be transmitted to the central node could be significantly reduced. Since each group of nodes would be essentially independent of the others, the system could be configured such that each of the nodes would be assigned a different coding scheme, optimizing the overall performance. In Chapter 4, we propose the use of dynamic programming techniques to optimize the network configuration.

Even though the lifting scheme offers some advantages, it also raises the question of how an even simpler, off-the-shelf coding scheme would perform. *Differential Pulse Code*

Modulation (DPCM) schemes have been widely used in speech and video coding to exploit data correlation [23]. A more detailed discussion on DPCM and a performance comparison against the methods proposed in this chapter can be found in Section 2.7.

In the next sections we give a brief overview of the wavelet transforms and their polyphase representation, used in the lifting implementation, and describe the lifting algorithm. We then describe the two proposed distributed algorithms, and finalize the chapter addressing cost and irregular sampling issues.

2.2 Wavelet Transforms and Polyphase Representation

Wavelets are basis functions that are used to provide a multiscale representation of signals. Their main feature is to allow a flexible choice between frequency and time resolution. A traditional way of implementing a wavelet transform is by using multirate filter banks.

Let $\tilde{h}$ and $\tilde{g}$ denote the low-pass and high-pass analysis filters, respectively, of a forward wavelet transform, and h and g denote the synthesis filters used for the corresponding inverse transform. Figure 2.1 shows the wavelet decomposition of the input sequence λ_0 into the sequences λ_1 and γ_1. λ_0 is filtered by the analysis filters and then subsampled. The inverse transform upsamples the subband signals and then uses the synthesis filters to reconstruct the original sequence. Additional levels of decomposition can be computed by successively decomposing the sequences λ_j, as shown in Figure 2.2. Equations (2.1) and (2.2) show how to compute λ_{j+1} and γ_{j+1} from λ_j in the time domain ($\lambda_j(n)$ and $\gamma_j(n)$ denote the n-th sample of sequences λ_j and γ_j respectively, and $\tilde{h}_k$ and $\tilde{g}_k$ are the

k-th coefficient of filters $\tilde{h}$ and $\tilde{g}$). For a detailed description of the wavelet transform we refer to [37] and [42].

$$\lambda_{j+1}(n) = \sum_k \tilde{h}_{k-2n} \lambda_j(n) \tag{2.1}$$

$$\gamma_{j+1}(n) = \sum_k \tilde{g}_{k-2n} \lambda_j(n) \tag{2.2}$$

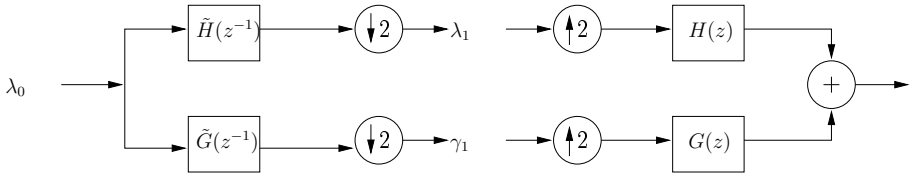


Figure 2.1: Discrete Wavelet Transform. The forward transform consists in filtering the input signal with the analysis filters $\tilde{h}$ and $\tilde{g}$ followed by downsampling. The original signal is recovered by filtering an upsampled version of the subband signals $\lambda_1(n)$ and $\gamma_1(n)$ with the synthesis filters h and g.

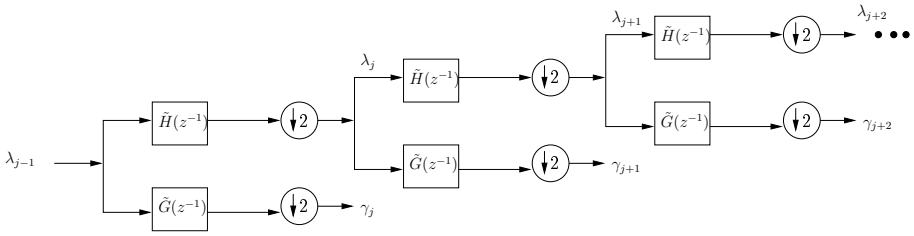


Figure 2.2: Multiple levels of decomposition of the Discrete Wavelet Transform.

Another useful way of representing the wavelet transform is by using a *polyphase representation*. Let $H(z)$ denote the z-transform of a FIR filter h, with coefficients $\{h_a, h_{a+1}, \ldots, h_{b-1}, h_b\}$. $H(z)$ is given by

$$H(z) = \sum_{k=a}^{b} h_k z^{-k}.$$

The polyphase representation of the filter h is then given by

$$H(z) = H_e(z^2) + z^{-1}H_o(z^2)$$

where H_e and H_o are the even and odd parts of H, respectively:

$$H_e(z^2) = \frac{H(z) + H(-z)}{2} \qquad H_o(z^2) = \frac{H(z) - H(-z)}{2z^{-1}}$$

The polyphase matrix for a pair of filters (h, g) is expressed as

$$\mathbf{P}(z) = \begin{bmatrix} H_e(z) & G_e(z) \\ H_o(z) & G_o(z) \end{bmatrix} \tag{2.3}$$

and the corresponding dual polyphase matrix $\tilde{\mathbf{P}}(z)$ is defined as

$$\tilde{\mathbf{P}}(z) = \begin{bmatrix} \tilde{H}_e(z) & \tilde{H}_o(z) \\ \tilde{G}_e(z) & \tilde{G}_o(z) \end{bmatrix} \tag{2.4}$$

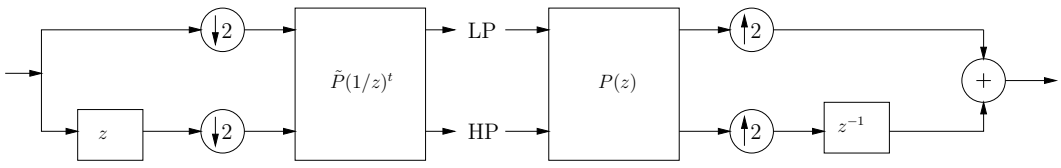


Figure 2.3: Polyphase implementation of the Discrete Wavelet Transform.

Perfect reconstruction is achieved when $\mathbf{P}(z)\tilde{\mathbf{P}}(z^{-1})^t = \mathbf{I}$, with $\mathbf{I}$ denoting the identity matrix. The polyphase implementation of the wavelet transform can be seen in Figure 2.3.

2.3 The Lifting Scheme

The lifting factorization [39] provides a convenient representation of the wavelet transform. It can be described as a technique to construct wavelet bases or to factor existing wavelet filters into elementary building blocks without the use of the Fourier transform. The lifting scheme found its roots in a method to improve a given wavelet transform to obtain some specific properties, and had as one motivation the construction of second generation wavelets, i.e., wavelets which are not necessarily translates and dilates of one basic filter.

Computing the wavelet transform using the lifting approach consists of a sequence of basic steps. In the *split* step, the input sequence λ_0 is split into two smaller subsets λ_1 and γ_1. Although no restriction is imposed on how to split the data, typically, this is done by applying the polyphase transform, so that λ_1 and γ_1 correspond to even and odd samples, respectively. From this point on, we will assume that the set λ_k refers to the even samples, and the set γ_k to the odd samples of set λ_{k-1}. After splitting, the next steps recombine these two sets in subsequent lifting steps that decorrelate the two sequences.

Lifting steps usually come in pairs of a dual and a primal step. In the dual lifting step, called *prediction step*, the subset λ_1 is used to predict γ_1, based on the correlation present in the original data. Let the predictor operator be denoted as $P(\cdot)$. The set γ_1 is replaced by the difference between itself and its predicted value:

$$\gamma_1 := \gamma_1 - P(\lambda_1)$$

The lifting factorization does not impose any restriction on the nature of the predictor operator. However, in this thesis, we will assume that $P(\cdot)$ is a linear operator, and in this

case, if $\gamma_1(n)$ denotes the n-th sample of the sequence γ_1, we can express the prediction step as a filtering operation:

$$\gamma_1(n) := \gamma_1(n) - \sum_k p_k \lambda_1(n-k),$$

where $p(k)$ represents the k-th coefficient of the filter defined by the P operator.

If the prediction is reasonable, $P(\lambda_1)$ is likely to be similar to γ_1, and their difference will contain much less information than the original γ_1.

The primal step, also called *update step*, is used to restore some properties of the original signal, like the mean value, to have some separation in the frequency domain and to reduce aliasing (so far, the subset λ_1 was obtained just by downsampling λ_0). In this step, the subset λ_1 is substituted by smoothed values, with the use of an update operator, $U(\cdot)$:

$$\lambda_1 := \lambda_1 + U(\gamma_1).$$

In filtering notation (assuming $U(\cdot)$ is a linear operator):

$$\lambda_1(n) := \lambda_1(n) + \sum_k u_k \gamma_1(n-k).$$

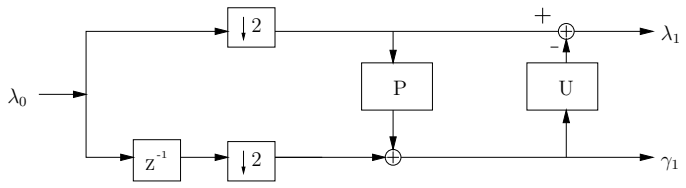


Figure 2.4: Lifting Scheme: split, predict and update.

The block diagram of the lifting steps is shown in Figure 2.4. It can be seen that no matter how P and U are chosen, the scheme will always be invertible. In cases where P

and U are linear operators, the scheme can also be implemented as a perfect reconstruction filter bank.

The sequence of lifting steps can be iterated using at each iteration, j, the set λ_{j-1} as the input sequence, leading to the following wavelet algorithm:

$$\textbf{For j=1 to n:} \begin{cases} \{\lambda_j, \gamma_j\} := Split(\lambda_{j-1}) \\ \gamma_j := \gamma_j - P(\lambda_j), \\ \lambda_j := \lambda_j + U(\gamma_j). \end{cases} \tag{2.5}$$

The inverse algorithm is easily obtained by reversing the operations:

$$\textbf{For j=n down to 1:} \begin{cases} \lambda_j := \lambda_j - U(\gamma_j) \\ \gamma_j := \gamma_j + P(\lambda_j), \\ \lambda_{j-1} := \lambda_j \cup \gamma_j. \end{cases} \tag{2.6}$$

2.3.1 Lifting and the Wavelet Transform

In [39], Daubechies and Sweldens proved that any polyphase matrix $\mathbf{P}(z)$ representing a wavelet transform with FIR filters can be factored into the product of upper and lower 2×2 triangular matrices, and a normalization matrix:

$$\mathbf{P}(z) = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & S_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ T_i(z) & 1 \end{bmatrix} \tag{2.7}$$

It is easy to see this matrix representation as a sequence of lifting steps, where the predictor and update filters are given by $T_i(z)$ and $S_i(z)$. For instance:

$$\begin{bmatrix} 1 & 0 \\ T_1(z) & 1 \end{bmatrix} \begin{bmatrix} \Lambda_1(z) \\ \Gamma_1(z) \end{bmatrix} = \begin{bmatrix} \Lambda_1(z) \\ \Gamma_1(z) + T_1(z)\Lambda_1(z) \end{bmatrix}$$

corresponds to a dual lifting step (prediction). $\Lambda_1(z)$ and $\Gamma_1(z)$ correspond to the z-transform of the sequences λ_1 and γ_1, respectively. Similarly,

$$\begin{bmatrix} 1 & S_1(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Lambda_1(z) \\ \Gamma_1(z) \end{bmatrix} = \begin{bmatrix} \Lambda_1(z) + S_1(z)\Gamma_1(z) \\ \Gamma_1(z) \end{bmatrix}$$

corresponds to a primal lifting step (update). A graphic interpretation of equation (2.7) can be seen in Figure 2.5.

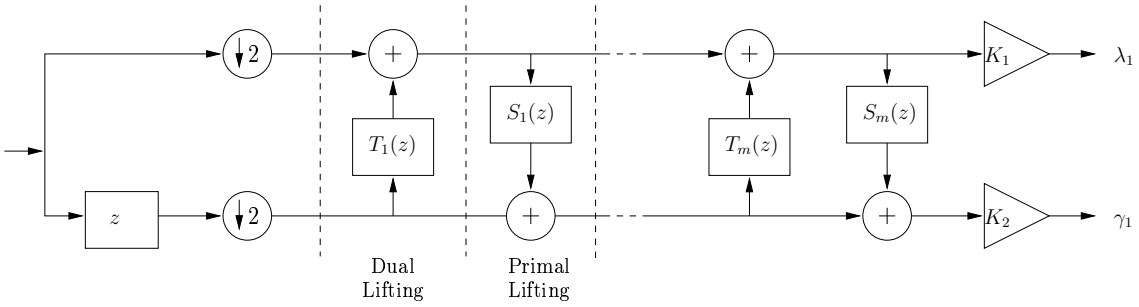


Figure 2.5: Graphical view of the lifting dual and primal steps. After splitting the signal into even and odd parts, a sequence of prediction and update operations is performed, followed by a normalization.

The detailed method used to obtain the polynomials $S_i(z)$ and $T_i(z)$ from the wavelet filters is based on the Euclidean algorithm, and is described in Appendix A. With this algorithm it is possible, by using the polyphase representation and the lifting algorithm, to compute any wavelet transform by applying a splitting step followed by alternating primal and dual lifting steps, and a normalization.

2.4 Two-Way Distributed Wavelet Algorithm

In this section we describe how to use the lifting formulation in our sensor network scenario in order to compute the wavelet transform coefficients. This method is based on a distributed implementation of the lifting factorization. Although anticausal, it provides a very simple way of computing the transform coefficients and calculating the total transmission cost in the network. While it may not be cost efficient for generic sensor placements (due to the transmission of anticausal coefficients), it will help us to introduce the algorithm proposed in Section 2.5.

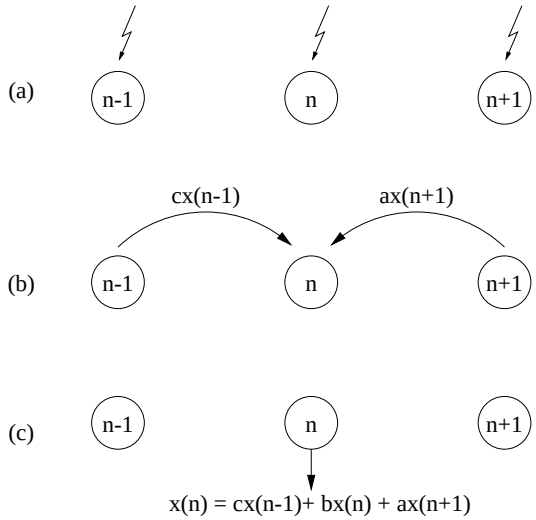


Figure 2.6: Computing data convolution with filter $H(z) = az^{-1} + b + cz$ in a sensor network. $x(n)$ denotes data at sensor n. (a) Sensors acquire data; (b) The two immediate neighbors transmit their weighted data to sensor n; (c) Sensor n computes the sum.

Let us first consider the distributed computation of a simple filtering operation. Consider a simple 3-tap filter, given by $H(z) = az^{-1} + b + cz$. Assume that we wish to filter the data *in space* in a one-dimensional array of sensors, i.e., each data sample to be filtered comes from a different sensor. For each position, the corresponding sensor needs

to receive the data from its immediate neighbors, weight them (with the associated filter coefficients) and add the result. Figure 2.6 illustrates the computation of the convolution of the data with filter $H(z)$ at sensor n. Convolution with larger filters will require more data transmissions, but can be implemented following the same idea. It is important to emphasize that n represents the *physical* position of the sensor in the 1D array, as illustrated in Figure 2.7.

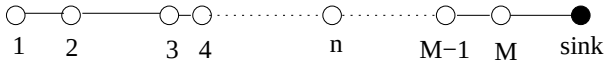


Figure 2.7: 1D array of M sensors and the sink.

Now, consider the implementation of a dual lifting step. Without loss of generality, we assume that $T_i(z)$ and $S_i(z)$ are symmetric, and have only two elements, so that they are multiples of $(1+z)$ (it is always possible to choose a lifting factorization that preserves symmetry, as seen in Appendix A). Let $x(n)$ denote the data at sensor n in the unidimensional array, with z-transform given by $X(z)$, and $x_e(n') = x(2n)$ and $x_o(n') = x(2n+1)$ correspond to its even and odd parts, with z-transforms given by

$$X_e(z^2) = \frac{X(z) + X(-z)}{2} \qquad X_o(z^2) = \frac{X(z) - X(-z)}{2z^{-1}}$$

Let $T(z) = a(1+z)$. Then:

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ T(z) & 1 \end{bmatrix} \begin{bmatrix} \Lambda_1(z) \\ \Gamma_1(z) \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ a(1+z) & 1 \end{bmatrix} \begin{bmatrix} X_e(z) \\ z^{-1}X_o(z) \end{bmatrix} \\ &= \begin{bmatrix} X_e(z) \\ z^{-1}X_o(z) + aX_e(z) + azX_e(z) \end{bmatrix} \end{aligned} \tag{2.8}$$

Equation (2.8) shows that the even part of the data remains unchanged, while the odd part is convolved with the filter $az^{-1} + 1 + az$. In other words, we can compute (2.8) as:

For each sensor n

if n *is even:*

$x(n) := x(n);$

if n *is odd:*

$x(n) := ax(n-1) + x(n) + ax(n+1);$

Let $\bar{x}_o(n)$ denote the data at the odd sensors *after* the dual lifting step. If $S(z) = b(1+z)$, we can describe the primal lifting step as:

$$\begin{aligned} \begin{bmatrix} 1 & S(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Lambda_1(z) \\ \Gamma_1(z) \end{bmatrix} &= \begin{bmatrix} 1 & b(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_e(z) \\ z^{-1}X'_o(z) \end{bmatrix} \\ &= \begin{bmatrix} bz^{-1}\bar{X}_o(z) + X_e(z) + b\bar{X}_o(z) \\ z^{-1}\bar{X}_o(z) \end{bmatrix} \end{aligned} \tag{2.9}$$

and the corresponding algorithm at the nodes is:

For each sensor n

if n *is even:*

$x(n) := b\bar{x}(n-1) + x(n) + b\bar{x}(n+1);$

if n *is odd:*

$\bar{x}(n) := \bar{x}(n);$

Therefore, a wavelet transform can be factorized into lifting steps, and implemented in a sensor network as a sequence of dual and primal steps. The implementation of the two steps is illustrated in Figure 2.8. Let $\bar{x}(n)$ denote the processed (filtered) data at node n. In the dual step (Figure 2.8(a)), each odd sensor $(2n+1)$ receives weighted data from its neighbors and computes $\bar{x}(2n+1) = x(2n+1) + ax(2n) + ax(2n+2)$. In the primal step (Figure 2.8(b)), each even sensor $(2n)$ receives the processed data from its odd neighbors, and computes $\bar{x}(2n) = x(2n) + b\bar{x}(2n-1) + b\bar{x}(2n+1)$. Subsequent dual and primal steps are computed in sequence, as described by Equation (2.7).

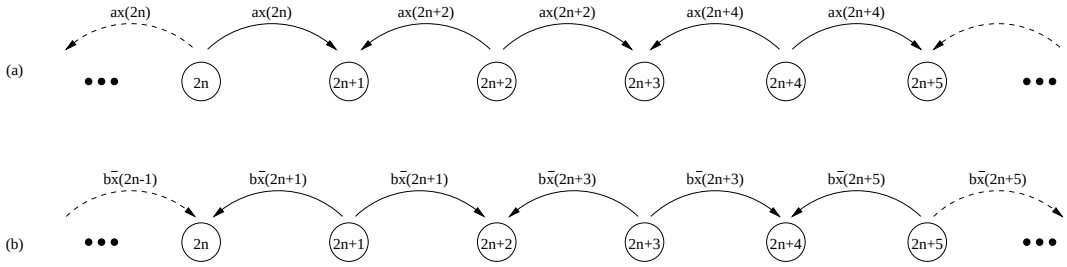


Figure 2.8: Implementation of a sequence of lifting steps. (a) dual step; (b) primal step.

Figure 2.8 illustrates each step in the computation of the first level of the wavelet decomposition. The implementation of a n-level decomposition follows the algorithm described in Equation (2.5). After the first level coefficients are computed, the data at the even sensors is split into even and odd, and the next level of decomposition is computed as before (Figure 2.9).

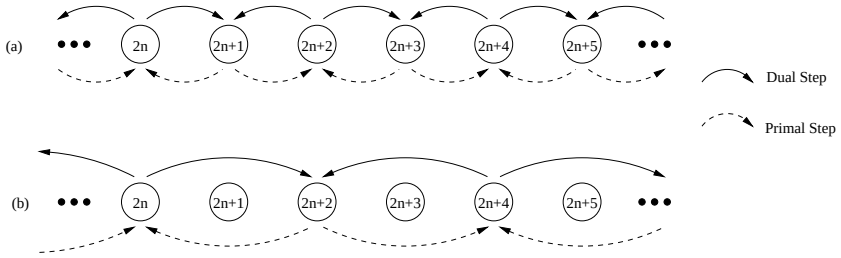


Figure 2.9: Lifting steps for the (a) first and (b) second level of wavelet decomposition.

2.4.1 Example

Consider the filter pair for the Cohen-Daubechies-Feauveau biorthogonal wavelets [8] with 2 vanishing moments for the dual and primal wavelet functions (CDF(2,2)), up to a normalization factor:

$$\begin{aligned} \tilde{H}(z) &= -\tfrac{1}{8}z^{-2} + \tfrac{1}{4}z^{-1} + \tfrac{3}{4} + \tfrac{1}{4}z - \tfrac{1}{8}z^{2} \\ \tilde{G}(z) &= \tfrac{1}{4}z^{-2} - \tfrac{1}{2}z^{-1} + \tfrac{1}{4} \end{aligned}$$

The polyphase matrix for this pair of filters is given by

$$\mathbf{P}(z) = \begin{bmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{1}{4}z \\ \frac{1}{4}z^{-1} + \frac{1}{4} & -\frac{1}{2} \end{bmatrix}$$

The symmetric lifting factorization for the above polyphase matrix (see Appendix A) is (omitting normalization)

$$\mathbf{P}(z) = \begin{bmatrix} 1 & \frac{1}{4} + \frac{1}{4}z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix}. \tag{2.10}$$

Therefore, the CDF(2,2) wavelet can be implemented by the lifting algorithm as described in Table 2.1:

Splitting (even and odd)	$x_e(n) \leftarrow x(2n)$ $x_o(n) \leftarrow x(2n+1)$
Dual lifting	$x_o(n) \leftarrow x_o(n) - \frac{1}{2}(x_e(n) + x_e(n+1))$
Primal lifting	$x_e(n) \leftarrow x_e(n) + \frac{1}{4}(x_o(n-1) + x_o(n))$

Table 2.1: Lifting operations for the CDF(2,2) wavelet.

Node communications in a sensor network computing this wavelet would follow Figure 2.9, where the corresponding filter weights in Figure 2.8 are given by $a = -\frac{1}{2}$, and $b = \frac{1}{4}$.

2.5 Partial Coefficient-based Distributed Wavelet Algorithm

As a means to overcome the causality problem, and reduce the number of transmissions, in our proposed system each sensor performs partial computations using the available data that arrives with the network flow. Let $H(z) = \sum_{k=-a}^{c} h_k z^{-k}$, $(a > 0, c \geq 0)$, be a generic filter defined by the transform to be applied to the network. The z-transform of this filter can be split into two terms

$$H(z) = A(z) + C(z) = \sum_{k=-a}^{-1} h_k z^{-k} + \sum_{k=0}^{c} h_k z^{-k}$$

where $A(z)$ and $C(z)$ are the anticausal and causal parts of the filter, respectively.

In the first step of the partial coefficient approach, each node computes the causal part of the filter convolution, and sends this information to the next node as its "partial coefficient". At subsequent nodes, where new data becomes available, the terms of the anticausal summation are added, refining the coefficient until it is fully computed. As an example, let $x(n)$ denote the raw data and $\bar{x}(n)$ the final coefficient for the n-th sensor. In other words, if the input signal is to be filtered by $h(n)$, then $\bar{x}(n) = x(n) * h(n)$. Consider the filtering operation $\bar{x}(n) = \alpha x(n-1) + x(n) + \beta x(n+1)$. In our approach, since sensor n does not have access to data from sensor $n+1$ (a future node), it computes just the

partial coefficient $\bar{x}_p(n)$ as $\bar{x}_p(n) = \alpha x(n-1) + x(n)$, and transmits it forward. When this partial data arrives at sensor $n+1$, it will be updated to $\bar{x}(n) = \bar{x}_p(n) + \beta x(n+1)$.

In this scenario, the lifting implementation of the wavelet transform offers some advantages. The in-place computation can reduce the memory requirements for the sensors. Since coefficients are only updated at relay nodes as they are forwarded to the sink, the scheme architecture also reduces the complexity in computing the transform coefficients and eliminates the need of extra transmissions. The advantages of lifting are discussed in the next section.

2.5.1 Partial Coefficients and Lifting

Let J denote the number of levels of decomposition in the wavelet transform being implemented. The general idea of the partial coefficient approach can be described by a simple algorithm:

for each sensor n**,**
  for $j = 1$ **to** J**,**
    for each previous partial,
      if data available:
        update partial;

But given the wavelet filters and the corresponding lifting decomposition, we need to determine at which nodes each specific partial coefficient will be updated, or given a specific node, which operations should be performed. We start considering a simple example with only one pair of lifting steps where each sensor only requires data from immediate neighbors at each step:

$$\mathbf{P}(z) = \begin{bmatrix} 1 & \beta(1+z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \alpha(1+z^{-1}) & 1 \end{bmatrix}.$$

In the first step, odd data is to be updated to $\bar{x}(2n+1) = \alpha x(2n) + x(2n+1) + \alpha x(2n+2)$. This step operates on raw data ($x(2n)$, $x(2n+1)$ and $x(2n+2)$), and the operations take place at each corresponding node, without the need of any prior processing. The second step updates even data to $\bar{x}(2n) = \beta\bar{x}(2n-1) + x(2n) + \beta\bar{x}(2n+1)$. This step requires data from the first step ($\bar{x}(2n-1)$ and $\bar{x}(2n+1)$), and updates can only occur at the nodes where these operations are finished. Since the filter sizes in the lifting matrices are known, we can compute at which nodes each of the steps will finish processing the data, and, therefore, at which node the subsequent step will be able to update its own coefficients. In the mentioned example, coefficient $2n+1$ will be updated and become fully available at node $2n+2$. Since coefficient $2n$ requires $\bar{x}(2n+1)$, it will also be updated at $2n+2$. Operations at each node are summarized in Table 2.2, where $x(n)$ denotes raw data from sensor n, $\bar{x}_p(n)$ a partially computed coefficient, and $\bar{x}(n)$ a fully computed coefficient.

Computations performed at sensor (2n+1)	Computations performed at sensor (2n+2):
• $\bar{x}_p(2n+1) := x(2n+1) + \alpha x(2n)$	• $\bar{x}(2n+1) := \bar{x}_p(2n+1) + \alpha x(2n+2)$ • $\bar{x}(2n) := \bar{x}_p(2n) + \beta\bar{x}(2n+1)$ • $\bar{x}_p(2n+2) := x(2n+2) + \beta\bar{x}(2n+1)$

Table 2.2: Node operations for partial coefficient approach for the CDF(2,2) wavelet.

The general idea is that, for any given lifting step, updates will occur whenever each of the terms in the summation (that uses the previous step data) becomes available. For higher levels of decomposition, coefficients will be computed after coefficients from lower

levels are fully available. Consider a generic wavelet polyphase matrix factorized as a product of m dual and primal lifting steps:

$$\mathbf{P}(z) = \prod_{i=1}^{m} \begin{bmatrix} 1 & S_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ T_i(z) & 1 \end{bmatrix}. \tag{2.11}$$

Without loss of generality, let $\mathbf{E}_i(z)$ represent either one of the elementary matrices such that:

$$\mathbf{E}_i(z) = \begin{cases} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} & \text{if } i \text{ even}, \\ \\ \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} & \text{if } i \text{ odd} \end{cases} \tag{2.12}$$

We can now express the polyphase matrix as

$$\mathbf{P}(z) = \prod_{i=1}^{2m} \mathbf{E}_i(z). \tag{2.13}$$

If i is odd, matrix $\mathbf{E}_i(z)$ represents a prediction step, meaning that data from even sensors is used to modify odd sensor data. If i is even, we have an update step, and even sensor data is modified based on odd sensor data. Let

$$E_i(z) = \sum_{k=-a_i}^{c_i} h_{i_k} z^{-k} \quad a_i, c_i \geq 0 \tag{2.14}$$

be the filter in matrix $\mathbf{E}_i(z)$ (i.e., $S_i(z)$ or $T_i(z)$, depending if i is even or odd, respectively). a_i denotes the number of anticausal coefficients in the filter $E_i(z)$. Since the first step $\mathbf{E}_1(z)$ needs a_1 even future measurements, the result of the first lifting step for sensor n (odd) will only be available at sensor $n+(2a_1-1)$ (Figure 2.10(a) illustrates the case where $a_1 = 2$). As a result, an update of the second step $\mathbf{E}_2(z)$ at node n (even) that requires data from sensor $n+1$ (odd) will only be performed at node $(n+1)+(2a_1-1) = n+2a_1$. The even coefficient for the second step at node n will be available at node $n+(2a_1-1)+(2a_2-1)$ (see Figure 2.10(b), where $a_1 = a_2 = 2$).

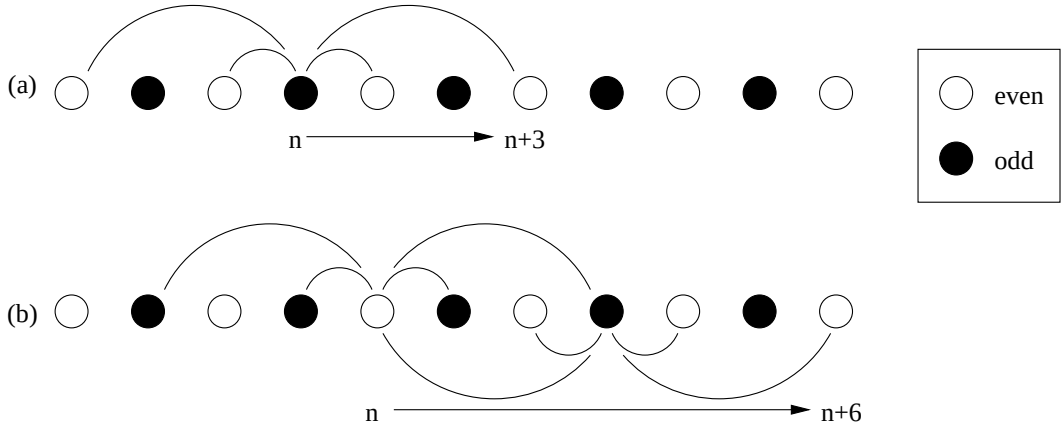


Figure 2.10: Coefficient availability for (a) dual and (b) primal lifting steps when $a_1 = a_2 = c_1 = c_2 = 2$.

Let

$$f_i = \sum_{k=1}^{i} (2a_k - 1), \tag{2.15}$$

$$f_i = f_{i-1} + (2a_i - 1), \quad i > 1$$

represent the number of future nodes after which coefficient for node n at step $\mathbf{E}_i(z)$ becomes available. Then, for a wavelet decomposition factorized as in Equation (2.11), the coefficient for node n at step $\mathbf{E}_i(z)$ will be available at node $n + f_i$, where it can be

used to update the coefficients for step $\mathbf{E}_{i+1}(z)$. The full coefficient for sensor n, $\bar{x}(n)$, will be available at position $n + f_{2m}$ if n is even, and at $n + f_{2m-1}$ if n is odd. Updates of the lifting step given by matrix $\mathbf{E}_i(z)$ for this coefficient occur at nodes $n+(2k-1)+f_{i-1}$, $k = 1, 2, \ldots, a_i$. It is interesting to notice that regardless of a_i, updates will always occur at even nodes, for n even or odd.

Following the notation introduced in Section 2.3, after the coefficients for the first level of decomposition are computed, even nodes will store the set λ_1, and the odd nodes the set γ_1. In the second level of decomposition, the same operations should be applied, but now using as input only the set λ_1. Since data from the first level of decomposition is only fully computed after a few nodes (where the exact place depends on the transform filters), the computation of the second level can only start after the nodes where the first level is completed. We now describe more formally the timing of the various computations taking place in the network. For description clarity purposes, since all the operations take place sequentially in the network, starting with node 1 (see Figure 2.7), we can introduce a time reference. Let t_i denote a time step, or interval, in the network operation. We say that at t_0 *all* the nodes acquire their measurements. At t_1, node 1 sends its data to node 2. At t_2 node 2 will use the previously received data to generate a partial version of the coefficients for both nodes 1 and 2, and forward the result to node 3, and so on. For an array of M nodes, the sink will receive the transform representation for the data measurements at t_M. Since we know that the first level coefficient for sensor n (even) is available at node $n + f_{2m}$ at time $t_{n+f_{2m}}$, the nodes at which coefficients are updated and become available can be found iteratively. For the next level of decompositions, the input data sequence is downsampled, meaning that the new "even" coefficients correspond to

nodes $4n$, $n \in \mathbb{Z}$, and the "odd" to nodes $4n+2$. For the j-th level of decomposition, we can find the values for $f_i^{(j)}$ as:

$$f_1^{(j)} = 2^{j-1}(2a_1 - 1) + f_{2m}^{(j-1)}, \quad j > 1 \tag{2.16}$$

$$f_i^{(j)} = 2^{j-1}(2a_i - 1) + f_{i-1}^{(j)}, \quad i, j > 1$$

And with these values, we can find the position at which each coefficient for a given lifting step should be updated. For higher levels of decomposition ($j > 1$), updates will occur at nodes multiple of $f_{2m}^{(j)} \% 2^j$, where the operator % denotes the remainder of the division.

Algorithm 1 describes the Partial Coefficient Algorithm for a generic wavelet lifting decomposition. We use notations such as $\bar{x}^{(j-1)}(n)$ or $\bar{x}_p^{(j)}(n)$ to indicate variables at different steps (j-th level of decomposition) in the algorithm. Also, we represent the operation $x := x + y$ as $x+ := y$.

Algorithm 1 Partial Coefficient Algorithm

Initialize:
$f_0^{(j)} = 0; \quad j = 1$ to J
for all n **do**
  for $j = 1$ to J **do**
    $residue = f_{2m}^{(j)} \% 2^j$;
    for $s = 1$ to $2m$ **do**
      generate local partial based on available partial data;
      if $n\%2^j == residue$ (*updates will happen here*) **then**
        for $k = 1$ to a_s **do**
          $\bar{x}_p^{(j)}(n - 2^{j-1}[2k-1] - f_{s-1}^{(j)} - f_{2m}^{(j-1)}) + := h_{s_k} \bar{x}^{(j-1)}(n - f_{s-1}^{(j)} - f_{2m}^{(j-1)})$
        end for
      end if
    end for
  end for
end for

2.5.2 Example

In this section we describe the partial coefficient algorithm for two levels of decomposition using the same pair of wavelet filters in the example of Section 2.4.1. The polyphase matrix decomposition is reproduced here for convenience:

$$\mathbf{P}(z) = \begin{bmatrix} 1 & \frac{1}{4} + \frac{1}{4}z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix}$$

The lifting equations in time domain are

$$\begin{aligned} \bar{x}(2n+1) &= -\tfrac{1}{2}x(2n) + x(2n+1) - \tfrac{1}{2}x(2n+2) \\ \bar{x}(2n) &= \tfrac{1}{4}\bar{x}(2n-1) + x(2n) + \tfrac{1}{4}\bar{x}(2n+1) \end{aligned}$$

In this case we have

$$\begin{aligned} a_1 &= 1 \\ a_2 &= 1 \\ f_1 &= 2 \times 1 - 1 = 1 \\ f_2 &= 1 + (2 \times 1 - 1) = 2 \end{aligned}$$

In the first step for the first level of decomposition, odd sensors $(2n+1)$ compute the causal part of the coefficients. These partials will be updated $f_1 = 1$ node later (at sensor $2n+2$). In the second step, the partials for the even sensors $(2n)$ will be updated after $f_2 = 2$ nodes, at the following even sensor. Therefore, node $2n+2$ updates partials from nodes $2n+1$ and $2n$. The operations are, as in the example of Section 2.5.1:

Computations performed at sensor (2n+1)	Computations performed at sensor (2n+2):
$\bullet \bar{x}_p(2n+1) := x(2n+1) - \frac{1}{2}x(2n)$	$\bullet \bar{x}(2n+1) := \bar{x}_p(2n+1) - \frac{1}{2}x(2n+2)$ $\bullet \bar{x}(2n) := \bar{x}_p(2n) + \frac{1}{4}\bar{x}(2n+1)$ $\bullet \bar{x}_p(2n+2) := x(2n+2) + \frac{1}{4}\bar{x}(2n+1)$

In the second level of decomposition, the same operations should be applied, but now the input data corresponds to the set λ_1, the even first level coefficients $\{4n, 4n+2 | n = 0, 1, 2, \ldots\}$. If $\bar{x}'(n)$ denotes the second level coefficient for sensor n, the lifting equations are given by:

$$\begin{aligned} \bar{x}'(4n+2) &= -\tfrac{1}{2}\bar{x}(4n) + \bar{x}(4n+2) - \tfrac{1}{2}\bar{x}(4n+4) \\ \bar{x}'(4n) &= \tfrac{1}{4}\bar{x}'(4n-2) + \bar{x}(4n) + \tfrac{1}{4}\bar{x}'(4n+2). \end{aligned}$$

We need to find at which nodes each of the terms above become available, so they can be added to the partial data. From (2.16), we have that $f_1^{(2)} = 4$ and $f_2^{(2)} = 6$, and the operations can be described as

$$\bar{x}'(4n+2) \begin{cases} \bar{x}(4n+2) \text{ becomes available at } t_{(4n+2)+2} \\ \qquad \Rightarrow \text{ at node } (4n+4) \text{ compute } \bar{x}'_p(4n+2) = \bar{x}(4n+2) - \frac{1}{2}\bar{x}(4n) \\ \bar{x}(4n+4) \text{ becomes available at } t_{(4n+4)+2} \\ \qquad \Rightarrow \text{ at node } (4n+6) \text{ update } \bar{x}'(4n+2) = \bar{x}'_p(4n+2) - \frac{1}{2}\bar{x}(4n+4) \end{cases} \tag{2.17}$$

$$\bar{x}'(4n) \begin{cases} \bar{x}(4n) \text{ becomes available at } t_{(4n)+2}, \\ \bar{x}'(4n-2) \text{ becomes available at } t_{(4n-2)+4} \\ \qquad \Rightarrow \text{ at node } (4n+2) \text{ compute } \bar{x}'_p(4n) = \bar{x}(4n) + \frac{1}{4}\bar{x}'(4n-2) \\ \bar{x}'(4n+2) \text{ becomes available at } t_{(4n+2)+4} \\ \qquad \Rightarrow \text{ at node } (4n+6) \text{ update } \bar{x}'(4n) = \bar{x}'_p(4n) + \frac{1}{4}\bar{x}'(4n+2) \end{cases} \tag{2.18}$$

Therefore, second level odd nodes $(4n+2)$ will update partials from nodes $4n-2$ and $4n-4$, and generate partials for nodes $4n$. Second level even nodes $(4n+4)$ will generate partials for nodes $4n+2$. Operations described by equations 2.17 and 2.18 can be summarized as:

Computations performed at sensor (4n+2)	Computations performed at sensor (4n+4):
$\bullet \bar{x}'(4n-2) := \bar{x}'_p(4n-2) - \frac{1}{2}\bar{x}(4n)$	$\bullet \bar{x}'_p(4n+2) := \bar{x}(4n+2) - \frac{1}{2}\bar{x}(4n)$
$\bullet \bar{x}'(4n-4) := \bar{x}'_p(4n-4) + \frac{1}{4}\bar{x}'(4n-2)$	
$\bullet \bar{x}'_p(4n) := \bar{x}(4n) + \frac{1}{4}\bar{x}'(4n-2)$	

2.6 Cost Considerations

As seen in Section 2.4, the two-way algorithm benefits from the simplicity and structure of the lifting implementation. However, anticausality might become a problem, since a potentially large number of extra transmissions can be introduced, depending on the wavelet filter lengths and number of levels of decomposition. In order to be useful, the two-way implementation must consider how much extra communication can be introduced

such that it still leads to an overall reduction of the energy cost. The idea is to compare the compression performance (signal-to-noise ratio) and the cost to achieve that performance for a number of possible coding schemes (e.g., wavelet vs. raw data transmission) and number of levels of decomposition.

If the distances between sensors are known, a reasonable estimate for the cost of transmitting a b-bit packet between two sensors separated by a distance d can be given by $C = b \cdot d^{\alpha}$ [44, 21], where α is a constant that depends on the medium (typically 2). Since each lifting factorization step explicitly defines which coefficients need to be exchanged between each node, the total cost can be computed as the sum of all the individual costs for each data exchange. If transmissions are omnidirectional, the cost of each pair of transmissions to two neighboring sensors can be computed as the cost of the transmission to the more distant node.

While the partial coefficient approach has a more complex implementation than the two-way algorithm and increases the latency in the network (since the node closest to the sink has to wait for data from the farthest), it eliminates anticausal transmissions, reducing the energy consumption, and offers a straightforward way of computing the total cost. For a network with M nodes, where each node n is at a distance d_n of the next node and encodes its data using b_n bits, the total cost can be computed as $\sum_{n=1}^{M} b_n d_n^{\alpha}$.

2.7 Comparison Between DPCM and Wavelets

A simpler alternative to the Wavelet Transform to decorrelate data in a sensor network could use a coding scheme such as DPCM. DPCM is a predictive scheme that can reduce

redundancy in a series of highly correlated data [20, 22, 14]. In this coding technique the difference between two neighboring data samples is computed and quantized. Since the difference between two consecutive correlated samples tend to be small, they can be encoded more efficiently, requiring fewer bits to be represented. A simple representation of a DPCM encoder and decoder can be seen in Figure 2.11.

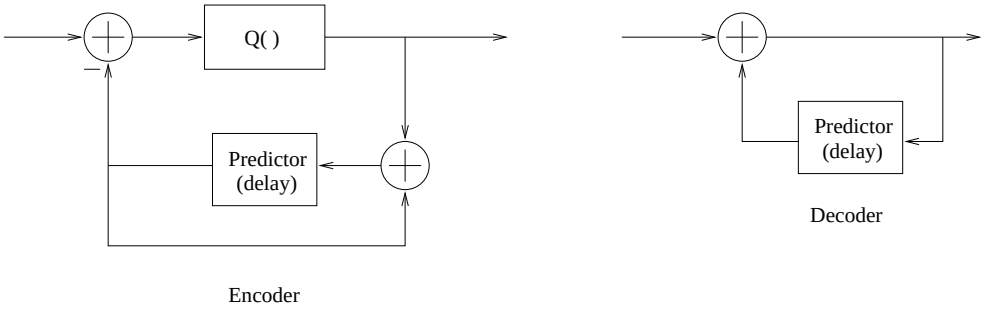


Figure 2.11: DPCM encoder and decoder.

While the simplicity of DPCM can be attractive to the sensor network scenario at first, transform coding methods are known to achieve, in general, superior compression performance as compared to predictive schemes. Also, a practical implementation of DPCM in a sensor network would either require side information or large memory buffers at the nodes. To make this point clearer, consider the distributed implementation of the DPCM scheme of Figure 2.11 along an array of M nodes plus the sink. For any node n, its DPCM coefficient is computed as $\bar{x}_D(n) = x(n) - \hat{x}(n-1)$, where $x(n)$ corresponds to the data value associated to node n, $hatx(n)$ is the quantized version of $x(n)$, and $\bar{x}_D(1) = \hat{x}(1)$. The coefficient is then quantized and forwarded to the sink. There are two main strategies for the nodes to compute their coefficients. In the first one, only DPCM coefficients are transmitted and/or forwarded in the network. Therefore, in order to compute its coefficient, each sensor is required to store the coefficients from

all previous nodes and compute the inverse operations to recover the original data value for the previous node and calculate the current coefficient. For example, node 3 needs to store $\bar{x}_D(1) = \hat{x}(1)$ and $\bar{x}_D(2)$, reconstruct $\hat{x}(2) = \bar{x}_D(2) + \hat{x}(1)$ and then compute $\bar{x}_D(3) = x(3) - \hat{x}(2)$. While this method does not require any additional transmission other than the actual DPCM coefficients, it is clear that memory/processing requirements for large values of M at nodes closer to the sink can become large. Another way of computing the DPCM coefficients at the nodes which does not require extra memory involves the use of side information. In addition to its own DPCM coefficient (which will be forwarded to the sink), each node can *also* transmit its data measurement to the next node, which will use it to compute its coefficient and then discard it. For example, if node 2 transmits $\hat{x}(2)$ to node 3, then node 3 can directly compute $\bar{x}_D(3)$ as $\bar{x}_D(3) = x(3) - \hat{x}(2)$. This method does not have memory requirements (the computations can be done in place), but the energy consumption in the network will be increased due to the transmission of side information, which will reflect in a decrease in performance.

While a combination of the two strategies, where only nodes closer to the sink (which suffer from larger memory requirements) receive side information could help in finding a trade-off between reducing memory requirements and additional energy consumption for a given application, due to the inherent limitations of predictive schemes like DPCM, its overall performance could still be lower than a transform based method. A performance comparison for both DPCM implementations and the method proposed in Section 2.5 can be seen in Figure 2.12. For this simulation an array of 100 nodes sampled a correlated field with data generated by a second order AR model with a high correlation parameter (Figures 2.12) and with a low correlataion parameter (Figure 2.13). It can be seen that

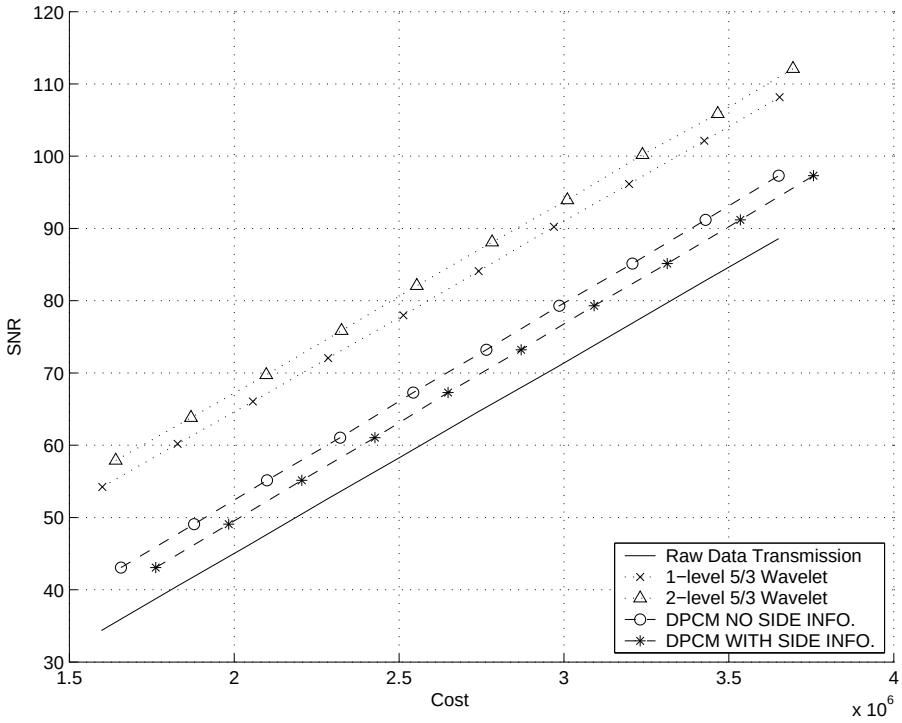


Figure 2.12: Performance comparison between DPCM with and without side information and the Partial Coefficient Approach for the high correlation data. Raw data transmission is included for reference.

there is a slight drop in performance for the DPCM with side information, as compared to the no side information case due to the additional transmissions, and that both DPCM schemes still had a significantly lower performance than the wavelet schemes. Raw data transmission (no processing) was included for reference. The reduction in the data correlation in Figure 2.13 reduced the performance of the coding schemes, and the overhead in the DPCM with side information case caused to even more costly than the no processing case. While there could be cases where the DPCM performance can match or even surpass transform schemes, wavelets are still expected to perform better in typical applications.

Note that DPCM filters are *IIR* (Infinite Impulse Response) filters, while the wavelet filters we use are *FIR* (Finite Impulse Response). In a distributed scenario, FIR filters

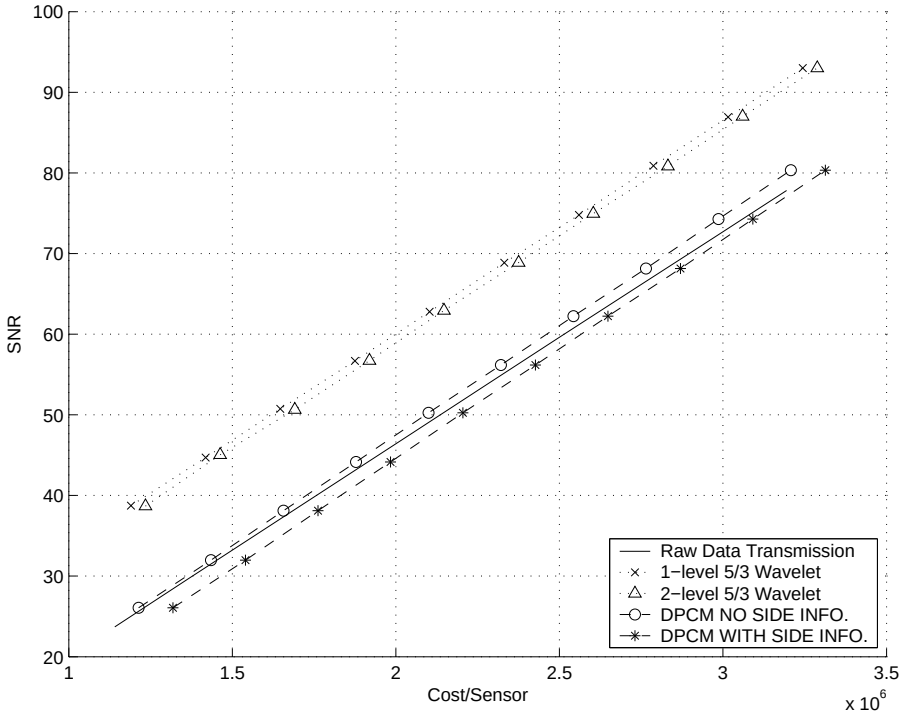


Figure 2.13: Performance comparison between DPCM with and without side information and the Partial Coefficient Approach for low correlation data.

should offer an advantage due to the their limited region of support, meaning that only data from a neighborhood is required in order to compute a transform coefficient, reducing memory and processing requirements when compared to IIR filters.

2.8 Irregular Sampling

Because the lifting scheme does not rely on the Fourier transform, it can be used to construct wavelets where translation and dilation (and thus the Fourier transform) cannot be used. A typical, but important example is the irregular sampling case. Many real life problems can benefit from basis functions and transforms adapted to irregular sampling.

While previous research has addressed the problem of computing wavelet transforms on irregular sampling spaces [2, 27], the vast majority of algorithms assumes the data field is being sampled as a regular, well-behaved structure, almost always a square grid (quincunx sampling grids have also been studied [19, 15]). The Lifting scheme provides a tool to design *second generation wavelets* [38], i.e., ones that are not necessarily translates and dilates of one fixed function, and that can, therefore, be adapted to irregular sampling. In this section we propose a slight modification of the CDF(2,2) wavelet transform, taking advantage of the lifting factorization, to improve the transform efficiency when data is irregularly sampled. While we limit this modification to the prediction step, similarly spatially varying update coefficients can also be computed. We refer to [40] for details.

We can illustrate the lifting versatility with the CDF(2,2) wavelet. While the CDF(2,2) can be derived using standard Fourier techniques, the intuition behind it can be clearly seen using the lifting approach. For this transform, the predictor P is given by the filter $p(z) = -\frac{1}{2}z^{-1} + 1 - \frac{1}{2}z$, which is simply the average of the two even neighbors. The detail coefficient $\gamma_1(n) = x(2n+1) - P(x(2n+1))$ is therefore given by:

$$\gamma_1(n) = x(2n+1) - \frac{x(2n) + x(2n+2)}{2}$$

If the underlying data is sufficiently correlated (as the wavelet transform usually assumes), and the sampled signal is locally linear, this predictor is optimal, since $x(2n+1) = [x(2n) + x(2n+2)]/2$, and therefore $\gamma_1(n) = 0$ (Figure 2.14).

If, however, the same signal was to be irregularly sampled, the predictor would not be optimal anymore, since $x(2n+1) \neq [x(2n) + x(2n+2)]/2$. Nevertheless, the lifting

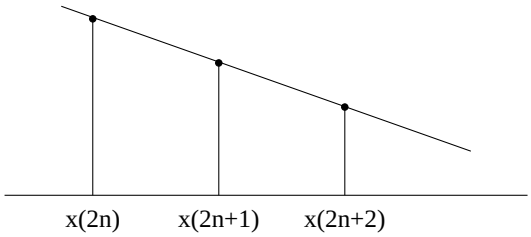


Figure 2.14: CDF(2,2) predictor with regular sampling and locally linear input.

approach allows for an easy solution to this problem. Using the same spatial argument, a good predictor would be $\beta x(2n) + (1-\beta)x(2n+2)$, where β is a parameter that varies spatially and depends on the distance between samples (Figure 2.15). The predictor P would then consist in a linear interpolator, instead of a simple average.

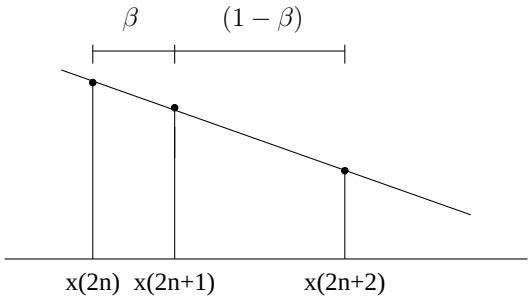


Figure 2.15: Modified CDF(2,2) predictor with irregular sampling and locally linear input.

Typical sensor deployments are usually irregular in nature, and could greatly benefit if a lifting structure that reflects the actual grid were designed. Figure 2.16 illustrates the difference in the magnitude of detail coefficients when a traditional CDF(2,2) transform and its modified version adapted to irregular sampling are used.

The modified version shows a significant reduction in the average energy of the detail coefficients, allowing a more efficient representation, using fewer bits for a same given distortion level, when compared to a traditional transform (Figure 2.17). This modified version of the CDF(2,2) is used in all the simulations in the upcoming chapters.

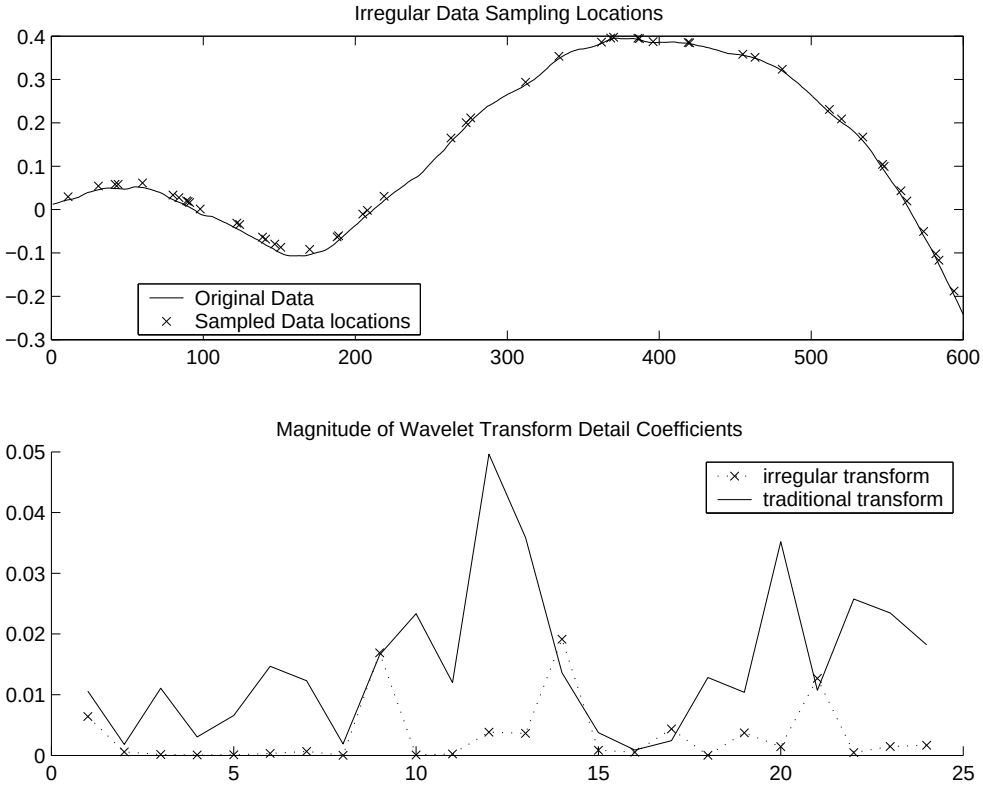


Figure 2.16: (top) A digital signal is sampled at 50 random locations. (bottom) Magnitude of detail coefficients for a transform adapted to irregular sampling and a traditional transform.

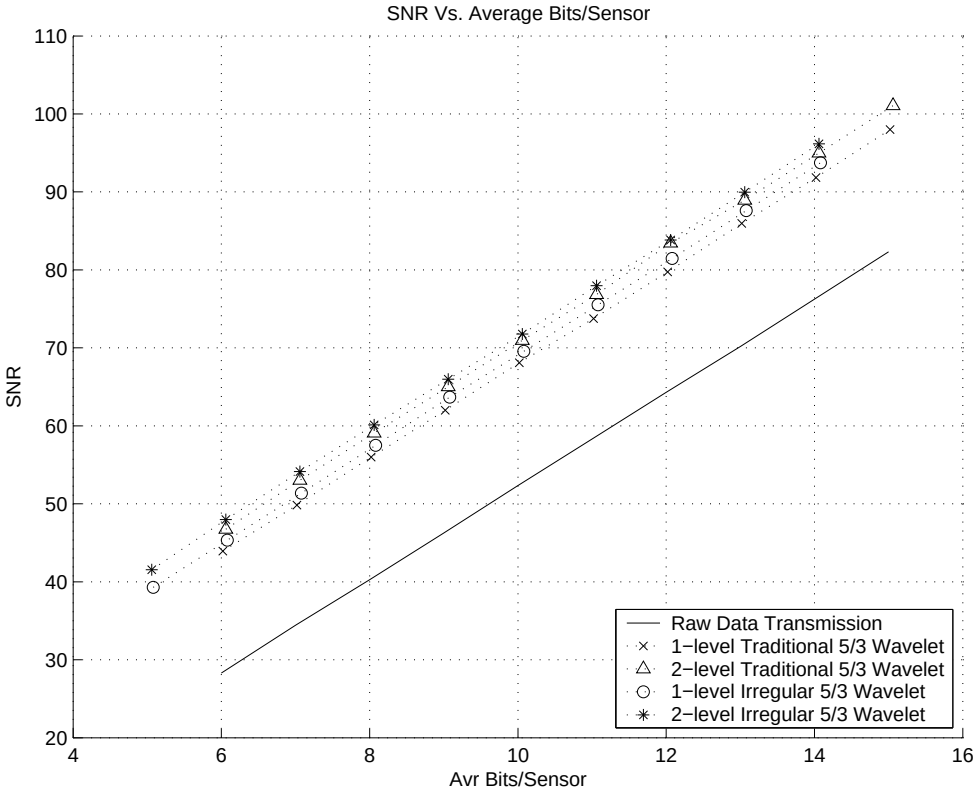


Figure 2.17: Performance of the CFD(2,2) wavelet transform adapted to irregular sampling compared to its traditional implementation, for the signal depicted in Figure 2.16.

Chapter 3

Partial Coefficient Quantization Effects

In standard transform implementations, the computations to obtain the transform coefficients are performed at full precision, and only the final coefficients are quantized. However, in a distributed network scenario, transmissions at full precision would significantly increase energy costs, making it necessary to also quantize partial coefficients. In this chapter we study the impact of partial coefficient quantization on the final distortion, and propose a general rule to allocate bits to partial coefficients, based on the rate allocated to full coefficients.

3.1 Motivation

Since each term added to the partial sums described in Section 2.5 has to be quantized, there will be an increase in distortion in the final transform performance. More specifically, let us assume the we are interested in the sum of two terms X_1 and X_2, and say that the final result is to be quantized by a quantizer Q_L. In other words, the final quantized transform coefficient is given by $Q_L(X_1 + X_2)$. In the sensor network scenario, however, each term should be independently quantized by an intermediate quantizer, Q_ℓ, before the final quantization. The final coefficient is therefore given by $Q_L(Q_\ell(X_1) + Q_\ell(X_2))$.

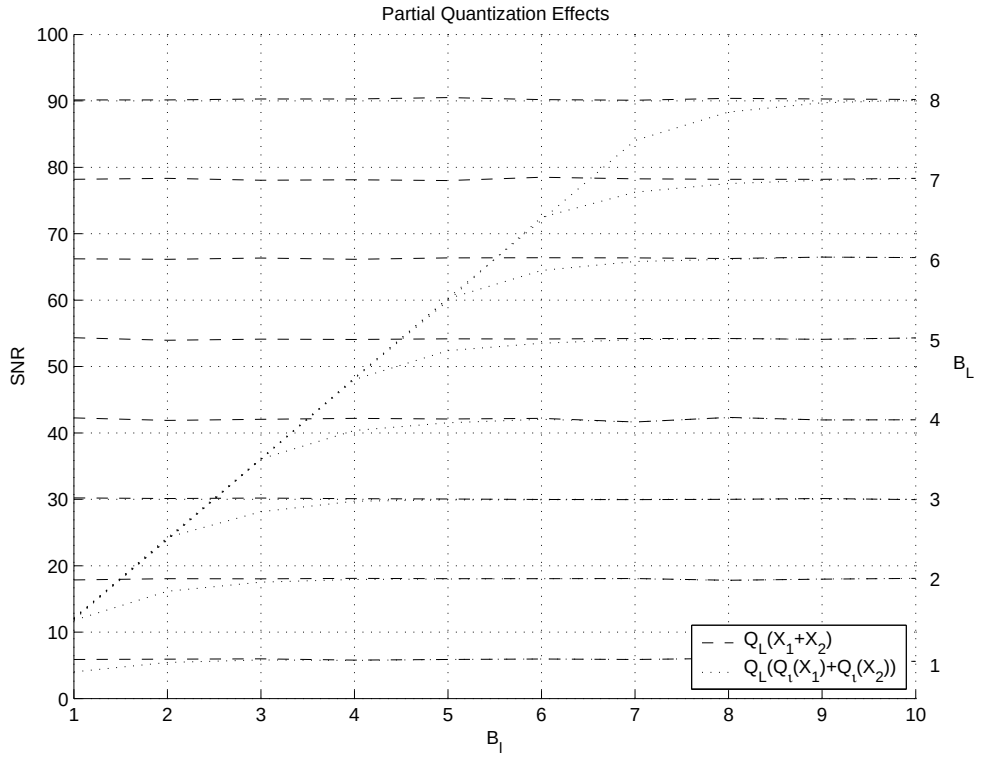


Figure 3.1: SNR performance of 2-lvl 5/3 wavelet transform with and without partial quantization. Intermediate quantization of summation terms (by Q_ℓ) introduces additional distortion

Figure 3.1 illustrates the effect of quantizing X_1 and X_2 separately by a quantizer Q_ℓ before the result of the sum $X_1 + X_2$ is quantized by Q_L. The simulation considered X_1 and X_2 as random variables uniformly distributed over the interval $[0, 1]$. Parallel, horizontal lines represent the performance of quantization only after the sum is computed (X_1 and X_2 are not quantized independently), i.e., $Q_L(X_1 + X_2)$. Each dashed line indicates the performance in terms of signal-to-noise (SNR) ratio of the output when B_L bits are allocated to Q_L. For the dotted lines, the variables X_1 and X_2 were quantized using B_ℓ bits (allocated to Q_ℓ) before the final summation was quantized with B_L bits, or $Q_L(Q_\ell(X_1) + Q_\ell(X_2))$. It can be seen that as B_ℓ increases (tending to full precision),

the dotted curves converge to the dashed curves. Lower values of B_ℓ result in a gap between two corresponding curves (same B_L), representing a drop in performance due to the additional distortion introduced by the partial quantization.

Since the transmission of partial coefficients at full precision would prohibitively increase the energy consumption in a sensor network scenario, we would like to have B_ℓ as low as possible while introducing as little extra distortion as possible. A quantification of the additional distortion introduced by partial quantization in terms of B_L and B_ℓ would allow us to define a rule to design the partial quantizer Q_ℓ such that a trade-off between extra distortion and additional bits is achieved. In the next sections we formalize this problem and provide a rule to design the partial quantizers.

3.2 Problem Description

Assume we have a uniform quantizer Q_L, with bin size equal to L. We want to quantize the result of $\alpha X_1 + \beta X_2$, where α and β are known real constants and X_1 and X_2 are uniform random variables. The resulting quantization $Q_L(\alpha X_1 + \beta X_2)$ gives a mean-squared error of

$$\varepsilon = \int\int (\alpha x_1 + \beta x_2 - Q_L(\alpha x_1 + \beta x_2))^2 p_{X_1,X_2}(x_1, x_2) dx_1 dx_2, \tag{3.19}$$

where $p_{X_1,X_2}(x_1, x_2)$ is the probability density function of the r.v.'s X_1 and X_2. Assume now that a second quantizer Q_ℓ, with bin size equal to ℓ, is used to quantize X_1 and X_2 before $\alpha X_1 + \beta X_2$ is computed. Let the resulting mean-squared error of $Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$ be

$$\varepsilon' = \int\int (\alpha x_1 + \beta x_2 - Q_L(\alpha Q_\ell(x_1) + \beta Q_\ell(x_2)))^2 p_{X_1,X_2}(x_1, x_2) dx_1 dx_2. \tag{3.20}$$

We want to be able to define a general rule on how to design (allocate bits to) the quantizer Q_ℓ such that the additional distortion introduced is below a certain threshold. In the sensor network scenario, Q_ℓ corresponds to quantization performed by the sensors on partial data, while Q_L is the quantization of full coefficients.

3.3 Graphical Interpretation

Let r_{L_k} and t_{L_k} denote, respectively, the k-th reconstruction level and decision level associated to the quantizer Q_L. In other words, for an input u, $Q_L(u) = r_{L_k}$ if $u \in [t_{L_k}, t_{L_{k+1}})$. Therefore, if the input to the quantizer is $\alpha X_1 + \beta X_2$, we have:

$$Q_L(\alpha X_1 + \beta X_2) = r_{L_k} \quad \text{if} \quad t_{L_k} \leq \alpha X_1 + \beta X_2 \leq t_{L_{k+1}} \tag{3.21}$$

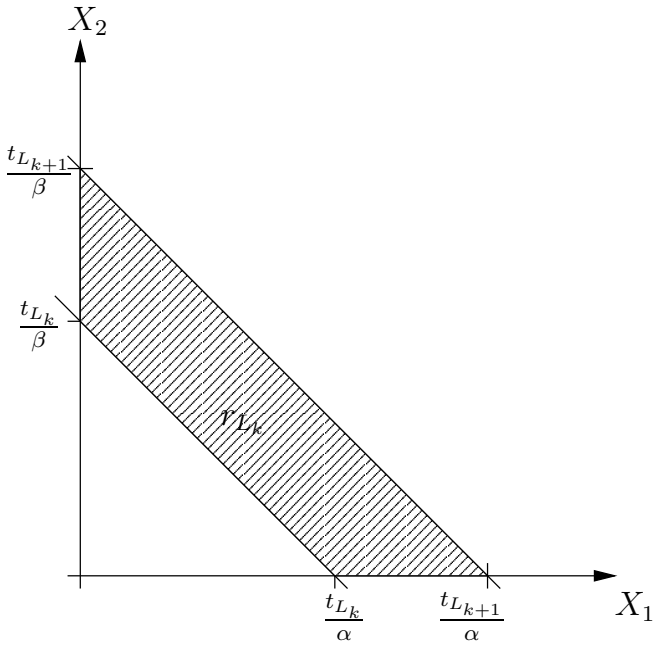


Figure 3.2: Graphical interpretation of Equation (3.21). If the result of $\alpha X_1 + \beta X_2$ falls inside the dashed region, it is quantized by Q_L as r_{L_k}.

Equation (3.21) defines a decision region in 2D space, as seen in Figure 3.2. Uniform quantization of X_1 and X_2 by Q_ℓ means that the 2D space is being divided using a square grid. Each value that falls inside a square bin is quantized to the value given by the central dot in Figure 3.3. The decision regions corresponding to the cases $Q_L(\alpha X_1 + \beta(X_2))$ and $Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$ can be seen in Figure 3.4, denoted by R_k and R'_k respectively.

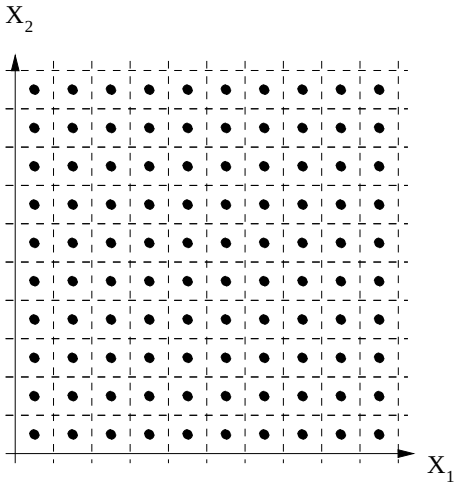


Figure 3.3: The uniform quantizer Q_ℓ divides the 2D space into a square grid. The reconstruction levels for each bin are given by the central dots in the figure.

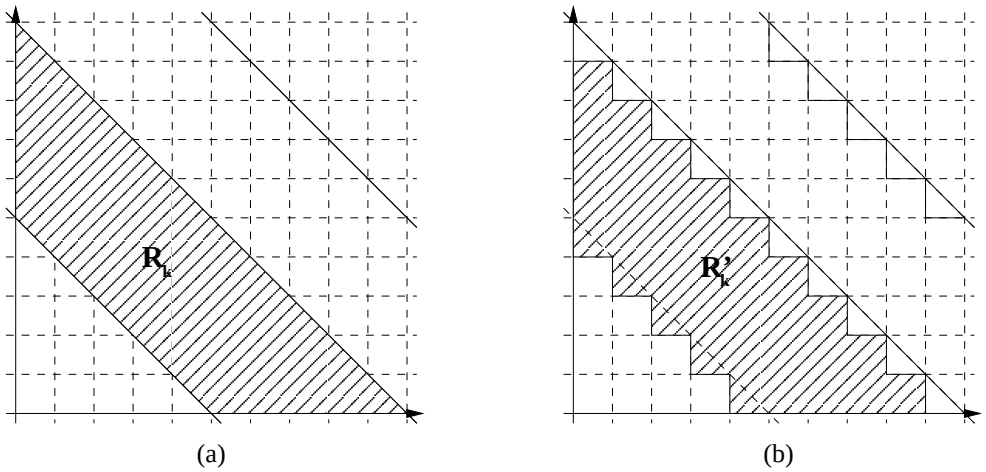


Figure 3.4: (a) Decision regions for $Q_L(\alpha X_1 + \beta X_2)$ (b) Decision regions for $Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$.

The result for the quantization $Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$ can be viewed graphically in Figure 3.5 for the case where $L = \frac{1}{2}$ and $\ell = \frac{1}{10}$. The dashed triangles correspond to the regions for which the quantizer Q_ℓ will introduce extra distortion, or $Q_L(\alpha X_1 + \beta X_2) \neq Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$. It can be seen that the size of the triangles is specified by the bin size of quantizer Q_ℓ, and as ℓ tends to zero (infinite precision), the extra distortion also tends to zero, or:

$$\frac{\varepsilon'}{\varepsilon} \to 1 \quad \text{as} \quad \ell \to 0 \tag{3.22}$$

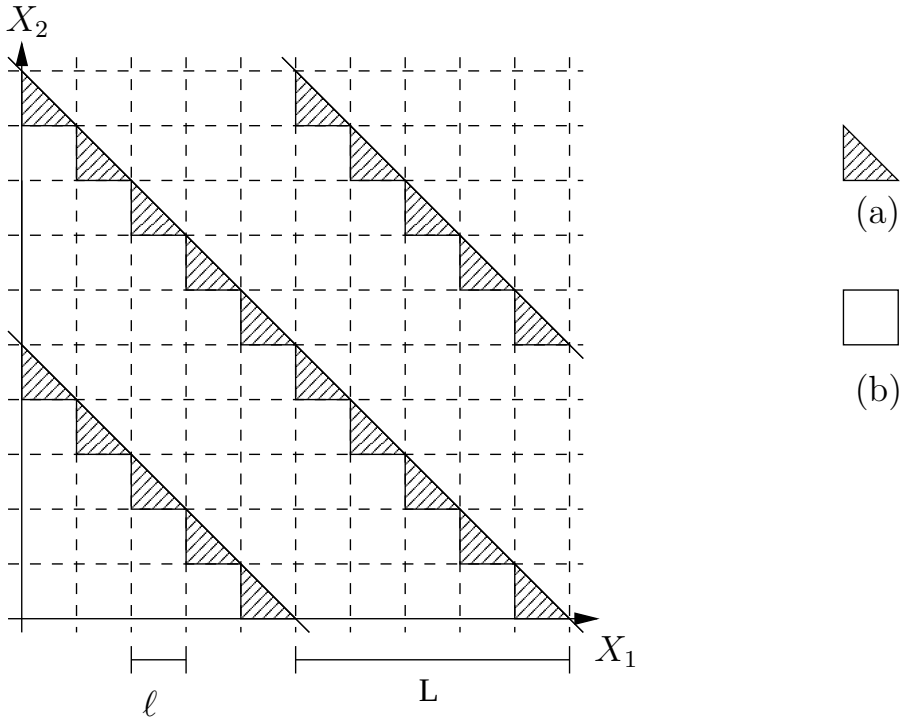


Figure 3.5: Graphical interpretation of quantization effects. (a) Regions for which $Q_L(\alpha X_1 + \beta X_2) \neq Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$ (b) Regions for which $Q_L(\alpha X_1 + \beta X_2) = Q_L(\alpha Q_\ell(X_1) + \beta Q_\ell(X_2))$

3.4 Q_ℓ Quantizer Design Based on Target Distortion

Let N be the number of extra bits that should be assigned to Q_ℓ when compared to Q_L, i.e., if Q_L has 2^B reconstruction levels, then Q_ℓ has 2^{B+N}. By computing the MSE ratio in terms of N, we can define a rule to design the quantizer Q_ℓ for any given value of $\frac{\varepsilon'}{\varepsilon}$, where ε and ε' are given by Equations (3.19) and (3.20) respectively.

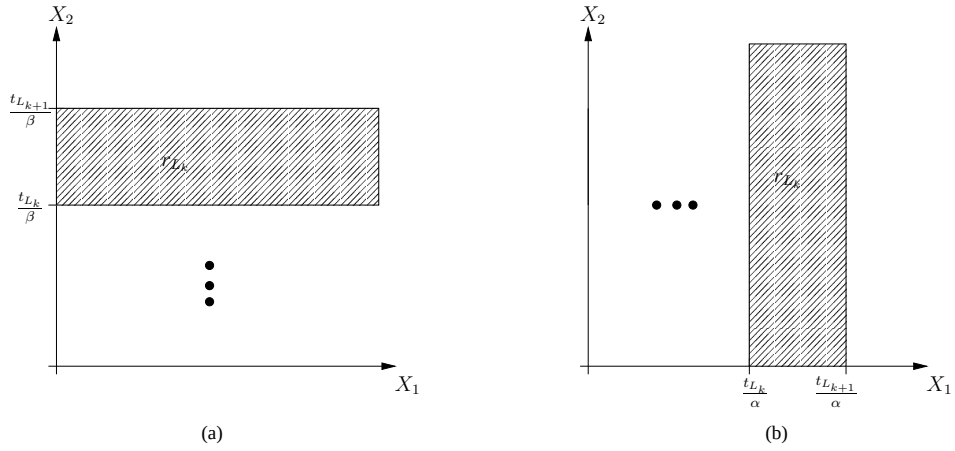


Figure 3.6: Decision regions for quantizer Q_L when (a) $\alpha = 0$ and (b) $\beta = 0$.

Another look at Figure 3.2 shows that the cases where $\alpha = 0$ or $\beta = 0$ correspond to either horizontal or vertical decision regions for Q_L, as depicted in Figure 3.6. For these cases, the quantization of X_1 and X_2 by Q_ℓ does not introduce extra distortion. Therefore, we can say that as α and β tend to zero, the value for the extra distortion is expected to decrease (the area of the dashed triangles in Figure 3.5 approaches zero). Based on these assumptions, we calculated the theoretical values for the above integrals for the case where $\alpha = \beta = 1$ (when the decision boundaries are at 135°), and verified experimentally that this represents a worst case, as shown later.

We evaluated the ratio of (3.19) and (3.20) for the case where $\alpha = \beta = 1$, and considered X_1 and X_2 as independent random variables uniformly distributed over the interval $[0,1]$ ($p_{X_1,X_2}(x_1,x_2) = 1$):

$$\frac{\varepsilon'}{\varepsilon} = \frac{\int_0^1 \int_0^1 (x_1 + x_2 - Q_L(Q_\ell(x_1) + Q_\ell(x_2)))^2 dx_1 dx_2}{\int_0^1 \int_0^1 (x_1 + x_2 - Q_L(x_1 + x_2))^2 dx_1 dx_2} \tag{3.23}$$

$$= \frac{\sum_k \iint_{R'_k} (x_1 + x_2 - r_{L_k})^2 dx_1 dx_2}{\sum_k \iint_{R_k} (x_1 + x_2 - r_{L_k})^2 dx_1 dx_2} \tag{3.24}$$

$$= 1 + \frac{4\ell^2}{L^2} \tag{3.25}$$

$$= 1 + \frac{4}{(\frac{L}{\ell})^2} \tag{3.26}$$

Since the range for the random variables X_1 and X_2 is $[0,1]$, then the range of $\alpha X_1 + \beta X_2$ is $[0,2]$. The bin size for a uniform quantizer is given by:

$$Bin\ Size = \frac{Range}{2^{(Number\ of\ bits)}}$$

We can then write:

$$\frac{L}{\ell} = \frac{\frac{2}{2^B}}{\frac{1}{2^{B+N}}} = 2^{B+N} \cdot \frac{2}{2^B} = 2^{N+1} \tag{3.27}$$

Finally, substituting (3.27) in (3.26) gives:

$$\frac{\varepsilon'}{\varepsilon} = 1 + \frac{4}{2^{2(N+1)}} = 1 + \frac{1}{2^{2N}} \tag{3.28}$$

This result shows that the MSE ratio is independent of the absolute value of the bin sizes of the quantizers, depending only on their ratio.

3.5 Simulations

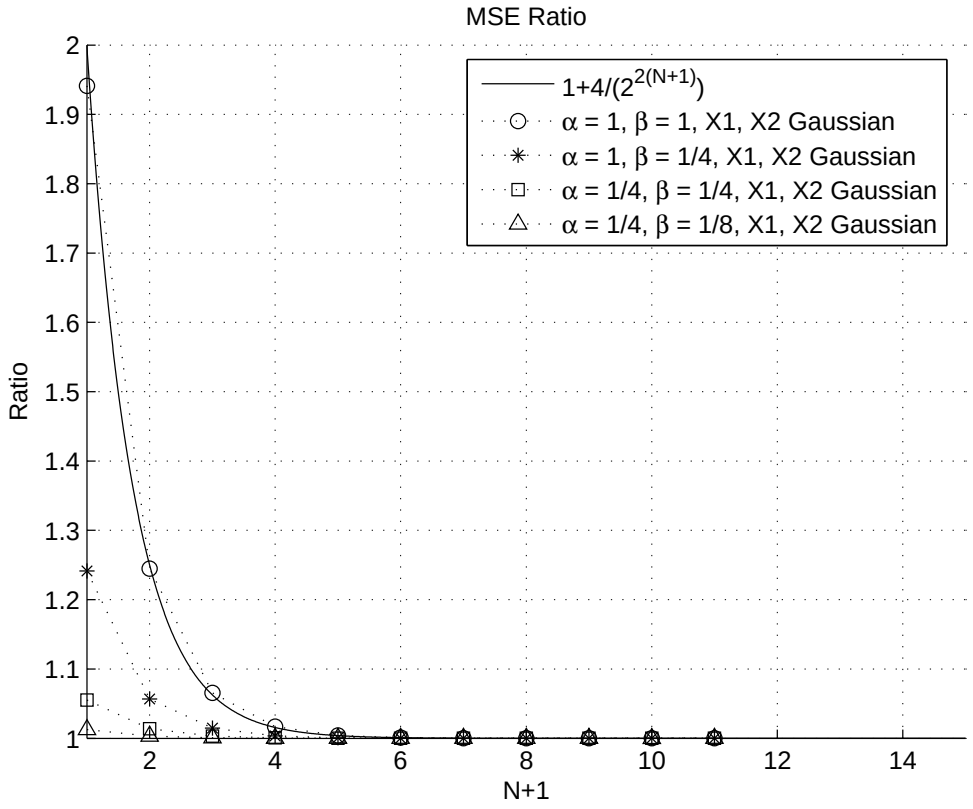


Figure 3.7: Theoretical and simulated results for MSE ratio with and without partial coefficient quantization

Simulations have shown that the case $\alpha = \beta = 1$ indeed represents the worst case, with the ratio $\frac{\varepsilon'}{\varepsilon}$ decreasing as α or β decrease, and that the results are little affected by the probability distributions of X_1 and X_2. Figure 3.7 shows the obtained theoretical curve plotted together with simulated results for a number of cases where $\alpha, \beta \leq 1$ and

X_1, X_2 are Gaussian r.v.'s. In practical transform computations, typical values for α and β are different for each type of output coefficient, but in general they are smaller than one, making the curve $1 + \frac{4}{2^{2(N+1)}}$ a still reasonable approximation for an upper bound of the ratio $\frac{\varepsilon'}{\varepsilon}$. By allocating N extra bits when quantizing X_1 and X_2 than when quantizing $\alpha X_1 + \beta X_2$ we can guarantee that the extra distortion introduced by the intermediate quantization will be bounded by the values given by the theoretical curve. Figure 3.7 also shows that values of N smaller than 3 introduce a large additional distortion, and the reduction in the extra distortion has a diminishing return as N increases for values above 3.

We now show that the partial coefficient quantization has indeed a major impact on the final distortion, and that the results obtained in Section 3.4 can be used to design the partial quantizers such that a good trade-off point between the allocated bits and the extra distortion introduced is achieved.

We compared the SNR of the standard (non-distributed) CDF(2,2) wavelet transform (1 and 2 levels of decomposition), where coefficients are computed at full precision and then quantized, with the proposed algorithms. As shown in Figure 3.7, the case $N = 3$ offers a good trade-off point between extra bits allocated to the partial coefficients and the return in terms of reduction of the additional distortion introduced. Figure 3.8 shows the obtained curves for the cases where partial coefficients are coarsely quantized ($N = 1$) and finely quantized ($N = 3$). It can be seen that the resulting SNR is very sensitive to coarsely quantized partial coefficients. In the sensor network scenario, choosing values of N much larger than 3 will certainly lead to better signal-to-noise ratio, but at the cost of an increase in the overall energy consumption.

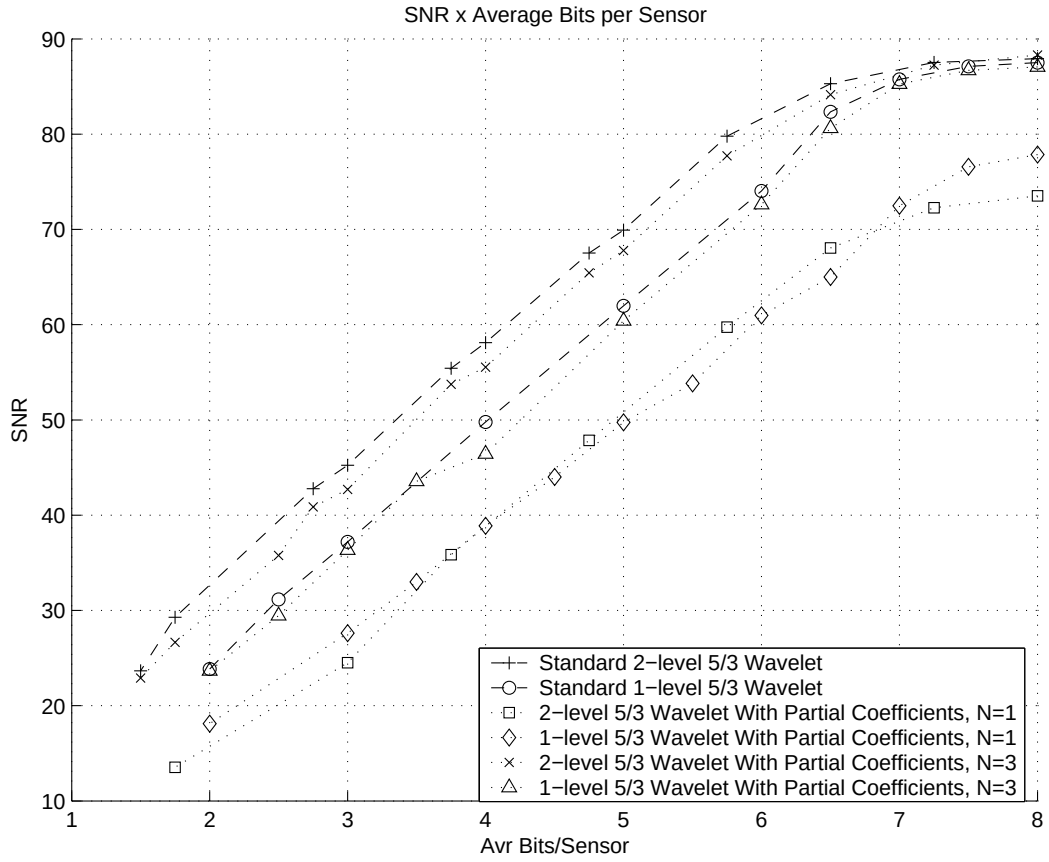


Figure 3.8: Effects of Partial Coefficient Quantization. $N = 1$ corresponds to coarse quantization, and $N = 3$ to fine quantization.

3.6 Conclusion

While the problem description in Section 3.2 considered a simplified case involving only two random variables in the summation, and the results of Section 3.5 have shown experimentally that this approach still provides a reasonable approximation to design the intermediate quantizers for the application considered in this thesis, other scenarios could require a more thorough analysis of partial quantization. We present here ideas that can be explored in future work.

A simple generalization of the problem presented in Section 3.2 involves considering a generic number of terms and quantizers in the summation. In this case, one would be interested in the contribution to the additional distortion introduced by each intermediate quantizer. More formally, we can consider a sequence of n random variables $X_1, X_2, \cdots, X_n$, and n coefficients $a_1, a_2, \cdots, a_n$. The problem is to efficiently design n intermediate quantizers $Q_1, Q_2, \cdots, Q_n$, such that the distortion of $Q_L(a_1Q_1(X_1) + a_2Q_2(X_2) + \cdots + a_nQ_n(X_n))$ is kept within a threshold of the distortion of $Q_L(a_1X_1 + a_2X_2 + \cdots + a_nX_n)$.

In another scenario, each term of the summation is added sequentially, and the problem of cascaded quantization can be addressed. In this case, we want to compare the distortion obtained without intermediate quantization $Q_L(a_1X_1+a_2X_2+\cdots+a_nX_n)$ with the cascade quantization case $Q_L(Q_n(\cdots Q_2(a_1Q_1(X_1) + a_2X_2) + \cdots + a_nX_n))$.

Chapter 4

Network Optimization Using Dynamic Programming Techniques

In practical applications, a number of different algorithms can be used by a sensor network for efficient data representation. In this chapter we provide a framework that allows finding, for a given network topology, which among the available coding methods is more suitable for each of the sensors, such that the whole network operates with a minimum cost in order to achieve a desired distortion level. In our proposed framework, a single-route path is described as a graph, with sensors representing the nodes, and where communication and processing costs are associated to edge weights and the coding schemes associated to states of operation. After describing data transitions and edge costs, we show that a shortest-path algorithm can be used to find the optimum route configuration, i.e., the one that leads to the lowest overall energy consumption.

4.1 Introduction

When data is acquired at multiple correlated sources, aggregation involving in-network data compression can offer a more efficient representation of measurements, significantly

reducing the amount of information that needs to be transmitted over the network, leading to a potentially large reduction in energy consumption. Prior work has addressed a number of distributed source coding (DSC) methods as a means to decorrelate data in sensor networks. While some rely on information exchange and additional computation inside the network leading to distributed versions of transforms, such as the Karhunen-Loève Transform [16], Wavelets [36] or the methods proposed in Chapter 2, others propose schemes that do not require internode communication, such as networked Slepian-Wolf coding [31, 10]. In general, DSC techniques face a trade-off between i) more processing at each node to achieve more compression and ii) less processing which would require more information (bits) to be sent to the sink. This trade-off has also been addressed by previous research. [29] provides an analysis on the regions in a network that should favor compression over routing based on the impact of spatial correlation of the measurements, and suggests that different routing and aggregation strategies should be used for different regions in the network. The performance of aggregation under a more general data model is considered in [18]. [10] proposes methods for network implementation of the Slepian-Wolf algorithm, and arrives at the similar conclusion that sensors closer to the sink benefit from coding schemes with smaller local cost while data from sensors far from the sink should be encoded with methods that achieve better compression performance.

While previous work has proposed methods to decorrelate data in a network and/or individually analyzed their performance, to the best of our knowledge, no prior work has addressed the problem of finding an optimal assignment of compression algorithms to nodes, in the sense of minimizing the energy consumption, when different coding schemes are available. Since the distortion/energy consumption trade-off also depends on factors

such as network topology and medium characteristics, different coding methods may be better suited for different parts of the network. These methods can consist of simple coding schemes such as DPCM, or more complex ones, such as Wavelet transforms with an arbitrary number of levels of decomposition.

To illustrate how the network topology can influence the performance of a given coding scheme, consider the following hypothetical example. Consider a group of three sensors in an array of equally spaced nodes (at a distance d of each other). This group of nodes is distant "N" hops from the sink, and it might choose between two coding schemes to decorrelate data. Method "A" could be a DPCM-like scheme, as discussed in Section 2.7. It requires no extra communication, but is limited in terms of performance. Method "B" could be, for example, the two-way wavelet transform of Section 2.4. It is locally more expensive energy-wise, but achieves better decorrelation. Let the cost of transmitting k bits over a distance d be kd^2. We would like to decide which method offers the smaller cost given the group distance to the sink. Figure 4.1 shows the three-node group in each scenario and the required local cost so that all the transmissions/computations required to have the coefficients ready to be forwarded to the sink have been performed. Say that method A can represent the original data using 4 bits per sensor (12 bits per 3 nodes), achieving a distortion D with a local cost of $18d^2$. Method B can achieve the same distortion D using 3.33 bits per sensor (10 bits per 3 nodes), but requiring a local cost of $26d^2$.

Therefore, for method A, the total cost to make all the data in the group available at the sink is given by the local cost ($18d^2$) plus the cost of forwarding the processed data N hops to the sink ($12Nd^2$), or $C_A = 18d^2 + 12Nd^2$. Similarly, we can calculate the cost

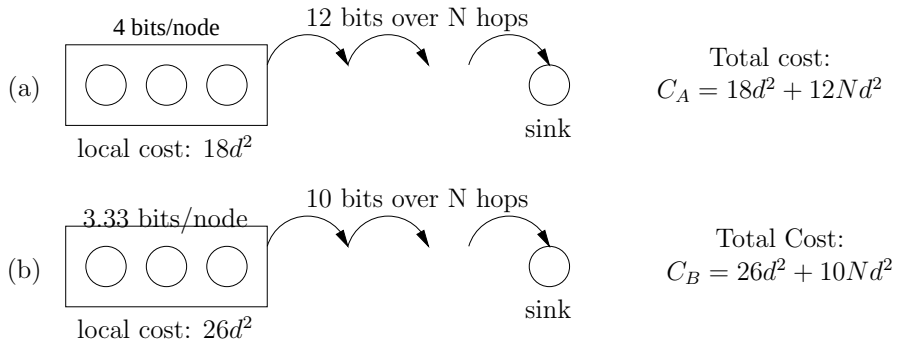


Figure 4.1: (a) Method "A": a simple encoding scheme is used; 12 bits are sent to the sink. (b) Method "B": a locally more expensive method is used, but better compression is achieved; 10 bits need to be forwarded to the sink. The cost of transmitting k bits over a distance d was computed as kd^2.

for method B as $C_B = 26d^2 + 10Nd^2$. We can, therefore, find for which values of N a method leads to lower cost than the other:

$$\begin{aligned} C_B &< C_A \\ 26d^\alpha + 10Nd^\alpha &< 18d^\alpha + 12Nd^\alpha \\ 8d^\alpha &< 2Nd^\alpha \\ N &> 4 \end{aligned}$$

The above equations show that if the three sensors are distant more than 4 hops from the sink, method "B" will lead to a lower energy cost. In general, it is expected that sensors closer to the sink should benefit from coding schemes that offer smaller local cost, while sensors that are far from the sink should encode using schemes that require a smaller average number of bits per sensor for a given distortion, agreeing with [10] and [29].

While both [10] and [29] indicate that the use of different encoding algorithms in different regions of the network would improve energy consumption efficiency in the network, they either require the assumption of specific data models to design the encoders

or the compression algorithms performance is only analyzed based on entropy models, with limited practical applicability. Two significant contributions of our work are that i) it makes no modeling assumptions about the underlying data statistics (instead all is required is training data) and ii) the algorithm directly compares the performance of actual coding schemes (instead of considering entropy) and provides more realistic network cost estimates. Our method is flexible enough to accommodate any network configuration (topology). We believe that the basic principles of our approach could be applied to other data representation selection among other sets of coding schemes (i.e., not limited to wavelets) that operate by exploiting spatial redundancy (in methods that do not involve data exchanges between nodes in the compression process, the optimization becomes straightforward).

4.2 Proposed Framework

We consider data aggregation (compression) along a single-route path from an edge to the sink, as seen in Figure 4.2 (a multi-route approach will be addressed in Chapter 5). This path is assumed to be known, which implies that a routing algorithm has been applied to the network first. Each sensor is assigned a number n, starting from the edge. The network topology (and thus the internode distances) is known, and each node in the path can operate using a coding scheme chosen from a predefined set of available coding schemes. In this work, available schemes are discrete wavelet transforms using the same filterbank but with different number of levels of decomposition (see Section 2.5): when the number of levels decomposition is increased, the potential compression efficiency

also increases (if data is highly correlated across sensors), but at the cost of more local information exchange (because data from more nodes is needed to compute some of the wavelet coefficients).

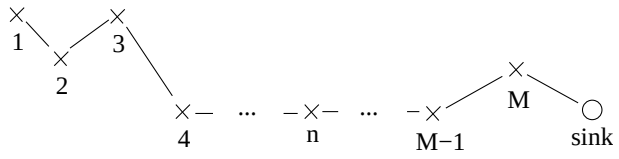


Figure 4.2: Single-route path with M nodes to the sink.

Since the wavelet transform is critically sampled, the number of wavelet coefficients generated is equal to the number of nodes. Using the partial coefficient approach, the wavelet coefficient corresponding to node n is computed in steps: at node n a partial version of the coefficient is first generated, which becomes a full coefficient as it "incorporates" additional data from future nodes (i.e., nodes closer to the sink). As seen in Section 2.5, the number of hops required until a partial coefficient becomes full depends on the specific transform filters being used.

4.2.1 Problem Description

In order to find the best coding scheme for each of the sensors, we propose representing the network as a graph. Figure 4.3 illustrates the graph associated to a single-route path including $M+1$ sensors (the last being the sink), where each sensor can use one of three available coding schemes. Each edge in the graph reflects a possible transition from one coding scheme to another, and has an associated weight that represents the transmission cost to continue in a determined scheme or the extra processing/transmission cost to

change coding schemes. Each possible path in the graph is associated to one choice of coding scheme for each node in the routing path from an edge to the sink.

Let $c_n(i,j)$ be the cost of the transition from method i at node n to method j at node $n+1$, $n = 1,\ldots,M$, where $i,j \in S = \{A,B,C,\ldots\}$, with $A,B,C,\ldots$ representing the coding schemes considered. Let $l_n(i)$ denote the local processing cost for sensor n to encode its data using coding scheme i. Our goal is to find the sequence $\{i_1,i_2,\ldots,i_n,\ldots,i_M\}$ of coding schemes associated to each of the nodes such that $\sum_n(c_n(i_n,i_{n+1}) + l_n(i_n))$ is minimum.

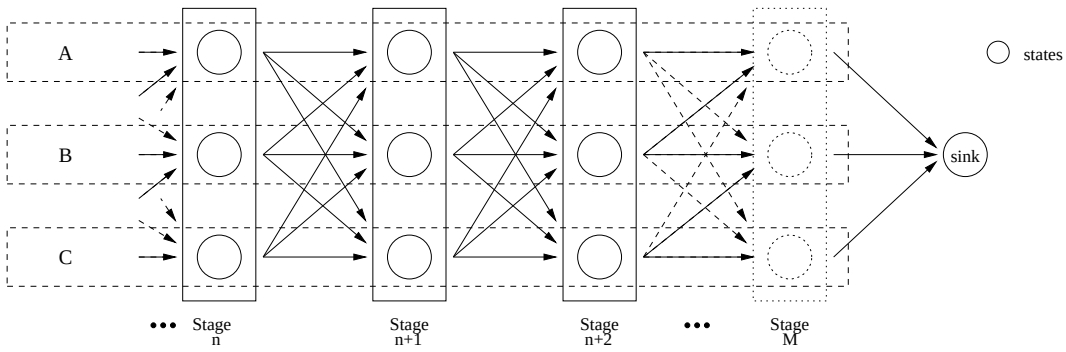


Figure 4.3: Sensor network seen as a graph. Each sensor is a node that can be in one state of operation, associated with a coding scheme (A, B, C) and a position in the network. Edges correspond to transitions between states, and have transmission and processing costs (weight) associated to them.

In this thesis, we assume that the coding schemes in Figure 4.3 consist in wavelet transforms with progressively larger number of levels of decomposition. We can now describe the state and transitions for the state machine of Figure 4.13 based on the partial coefficient approach.

4.2.2 State Description

A state can be described by two parameters: the node's position in the single-route path to the sink (n), and the coding scheme being used (j). We can define the following variables of interest:

- d_n: distance to next node;
- $R_{n,j}$: rate allocation for coefficient at position n encoded with jth coding scheme;
- D_n^2: cost to forward one bit to the sink. If we assume the cost to send b bits over d meters is bd^2, then $D_n^2 = \sum_{k=n}^{M} d_k^2$ (sum of cost for each hop to the sink).

While we make use of a simplified cost metric, practical applications could refer to hardware specifications for a more realistic energy consumption model [26]. Also, the cost function used in the optimization framework is not limited to energy consumption optimization. Other generic metrics can be defined to reflect application-specific constraints, such as minimization of data distance, maximization of correlation, etc.

4.2.2.1 State I/O

Each state in the graph is responsible for a series of computations that depend on the node and coding scheme represented by the state. More specifically, if a node is at position n and is encoding data using a wavelet transform with j levels of decomposition, that node is responsible for generating the j-level partial coefficient corresponding to node n and refining any previous partials that were also encoded with j-levels of decomposition and depend on the measurement at position n to be fully computed (Figure 4.4).

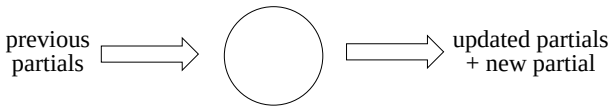


Figure 4.4: State I/O.

Let W_n^j denote the wavelet transform coefficients computed for nodes up to n for the j-th level of decomposition, i.e., $W_n^j = \{\gamma_1, \gamma_2, \cdots, \gamma_n, \lambda_n\}$, where γ_k and λ_k are obtained as described in Section 2.2. The input/output for state (n, j) can be summarized in Figure 4.5.

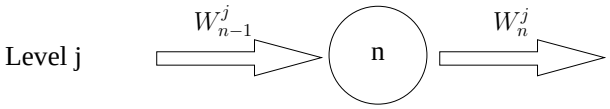


Figure 4.5: State (n, j) updates previous partials at same level and generates a new local coefficient.

In other words, at each state (n, j), $W_n^j = f(W_{n-1}^j)$, where the function $f(\cdot)$ is given by the specific transform being used.

4.2.2.2 State Costs

At state (n, j) the current path cost is incremented by the computational cost $l_n(j)$ of generating the partial coefficient at node n and updating any previous partials that depend on the measurement at node n. $l_n(j)$ is calculated in terms of the number of multiplications required, and depends on the specific state considered. Once a state is considered for a given path, its corresponding coding scheme (and thus the associated rate allocation) is known. Final coefficients will be forwarded to the sink *regardless of what algorithm is selected for the next nodes.* Thus, in evaluating the transmission cost at state (n, j) we include the total cost of transmitting the new final coefficient to the sink as $R_{n,j}D_n^2$.

4.2.3 Transition Description

As seen in Section 2.2, a wavelet transform with multiple levels of decomposition can be computed recursively, i.e., the low-pass coefficients from one level serve as the input to the next level (see Figure 4.6). This also means that any set λ_j can be recovered from λ_{j+1} and γ_{j+1}. In terms of the wavelet coefficients, we can therefore say that $W_n^j = f(W_n^{j-1})$ and also that $W_n^{j-1} = g(W_n^j)$, where $f(\cdot)$ and $g(\cdot)$ are two functions that depend on the specific transform being used. We refer to this property as the *embedded wavelet property.*

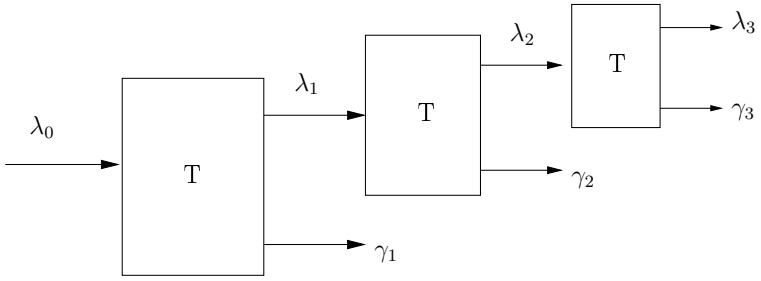


Figure 4.6: Embedded wavelet property.

The information that is conveyed in each of the transition scenarios corresponding to different branches of the state machine can be seen in Figure 4.7. The embedded property allows us to say that any given state (n, j) will be able to have access to the data it needs, i.e., W_{n-1}^j, in order to generate $W_n^j = f(W_{n-1}^j)$, regardless of whether the data is coming from state $(n-1, j-1)$, $(n-1, j)$, or $(n-1, j+1)$.

4.2.3.1 Transition Costs

The partial coefficient approach creates coefficients that are updated as data is transmitted towards the sink. However, the optimization framework allows changes in the algorithm along a single path. In this case, partial coefficients that are still in the process of being updated at future nodes are no longer refined, and are sent as such to the sink. As it

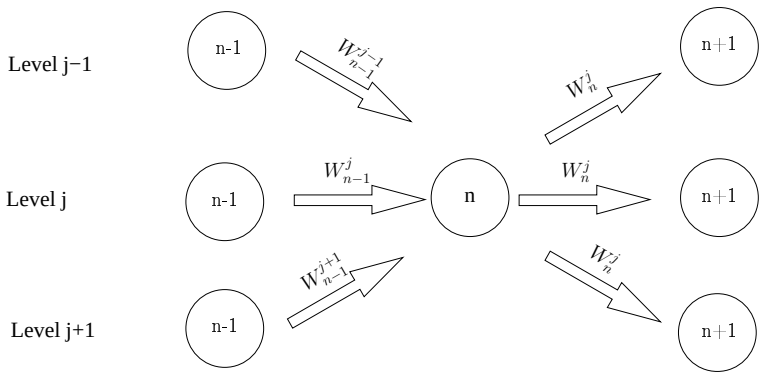


Figure 4.7: Data transitions for partial coefficient approach.

will be discussed in the next paragraphs, transition costs account for these two types of partials: the ones that are not going to be refined, and the ones that will be updated at future nodes. Since the network topology is known, the forwarding cost for data that will remain unchanged can be computed up to the sink at once. Transmission costs related to data that might change at future nodes is accounted for only locally (one hop).

Let $P(n, j)$ denote the number of partial coefficients that state (n, j) has to relay to the sink. Depending on the future state, a number of these partials may become final coefficients, and a new partial coefficient may be created (corresponding to the current node). Therefore if $F(n, j)$ denotes the number of partial coefficients that become final at state (n, j), then $P(n, j) = P(n-1, j) - F(n, j) + 1$. As seen in Chapter 3, every partial coefficient is quantized using N extra bits above the rate allocated for the corresponding final coefficient. As seen in Section 4.2.2.2, the cost for the allocated bits is added to the path cost at the node where the coefficient is created. Also, there are two possible transmission cases involving partial coefficients. In Case 1, some partial coefficients will be forwarded a few hops, then converted to final coefficients. This case occurs when a sufficiently large number of future states maintain the same coding scheme, allowing the

partials to be refined. In this case, the transmission cost for those N extra bits is added to the branch costs corresponding to the transmission of these partials. Since at node n we don't know how the partial coefficients will be transmitted (this will depend on future decisions), the path cost is incremented by only the cost of transmitting partial information to node $n+1$. If state (n,j) relays $P(n,j)$ partials at the additional cost of N bits per partial over the distance d_n, the transmission cost added to the path cost is given by $N.P(n,j)d_n^2$.

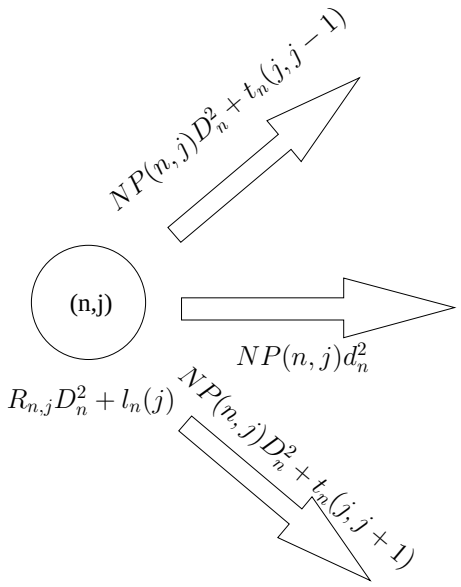


Figure 4.8: State and edge transmission costs.

In Case 2, partial coefficients will be, after a few hops, forwarded to the sink as such, without further refinement. Case 2 arises when there is a change in wavelet transform level. Therefore, at the transitions where there is a change of scheme, $P(n,j)$ partials at the additional cost of N bits per partial will have to be forwarded unchanged to the sink, and the transmission cost added to the path cost is given by $N.P(n,j)D_n^2$. Figure 4.8 shows the state and edge transmission costs for the possible state transitions. In addition

to the transmission costs, when there is a change in coding schemes a, typically small, processing cost $t_n(i_n, i_{n+1})$ is added to the branch to reflect the overhead operations the next node will have to perform to extract the lower level coefficients from the higher level ones, or to start the processing of higher level coefficients based on lower level ones, as allowed by the embedded property. $t_n(i_n, i_{n+1})$ is computed as the energy spent by the sensor processor to compute the multiplications required by the transition operations (additions can be typically neglected, since they are orders of magnitude computationally cheaper than multiplications [45]). Remaining on the same scheme adds no processing overhead, $t_n(j, j) = 0$ for all k. Since $t_n(j, j) = 0$ it is not depicted in the figure.

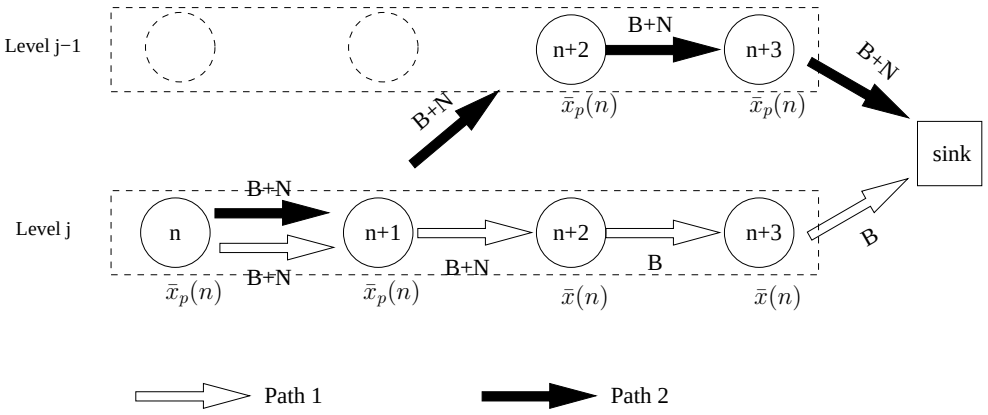


Figure 4.9: Transition costs example.

As an example, consider the transmission of the *single* coefficient for node n at level j, over two possible paths, in a network of $n+3$ nodes, as seen in Figure 4.9 (the transmissions for all the other coefficients are not considered here). Assume that all nodes are at a distance d of each other, and that B bits are allocated for the transform coefficient of node n, $\bar{x}(n)$. For illustration purposes, assume that the coefficient for node n will become a final coefficient at node $n + 2$, provided the same level of decomposition is

maintained for nodes $n+1$ and $n+2$. Until it reaches node $n+2$, the coefficient will be only partially computed, and represented by $\bar{x}_p(n)$, which is quantized with $B+N$ bits. The edge weights in Figure 4.9 depict the number of bits being transmitted over each edge (processing costs are not shown for clarity purposes).

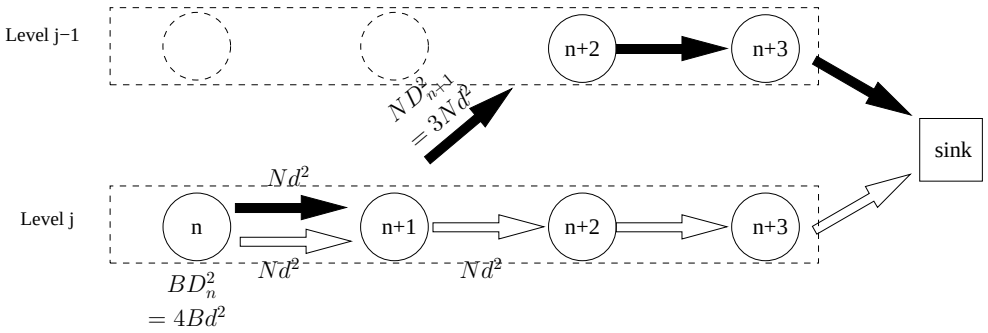


Figure 4.10: Transition costs computation for example of Figure 4.9.

In Path 1, the coding scheme is maintained and the coefficient for node n is refined until it becomes final, at node $n+2$. After that point, it is quantized only with B bits and $\bar{x}(n)$ is forwarded to the sink. In Path 2, the coefficient computation is interrupted since the coding scheme is changed, and $\bar{x}_p(n)$ will have to be forwarded to the sink as a permanent partial, quantized with $B+N$ bits. The costs added to the current paths are computed as depicted in Figure 4.10. The total transmission cost for Path 1 is $C_1 = 4Bd^2 + 2Nd^2$, and for Path 2, $C_2 = 4(B+N)d^2$.

4.2.4 Path Optimization

The partial coefficient approach guarantees that any computation at any given node requires only data from previous nodes. Also, due to the embedded property discussed in Section 4.2.3, any node always has access to the past coefficients it needs to compute its own partial coefficient, regardless of whether the data it is receiving was generated

from the same level j or from any other level. As a result, transition costs depend only on the present state: the physical position of the node in the network and the coding scheme being used, and, since the output of a node (its coefficient) is the same regardless of the previous path, costs up to a particular node do not influence the cost for a future transition. Without losing generality, consider the state (n, j), and let $S_a = \{i_{a_1}, i_{a_2}, \cdots, i_{a_{n-2}}, j-1\}$,$S_b = \{i_{b_1}, i_{b_2}, \cdots, i_{b_{n-2}}, j\}$ and $S_c = \{i_{b_1}, i_{b_2}, \cdots, i_{b_{n-2}}, j+1\}$ represent the three optimum (minimum cost) sequences of coding schemes up to states $(n-1, j-1)$, $(n-1, j)$ and $(n-1, j+1)$ respectively (transitions from a generic previous state could be defined similarly). Let $C_k(n)$ represent the *arriving* path cost up to node n (i.e., excluding the outbound costs for node n) for a scheme sequence S_k. We have that:

$$C_k(n) = \sum_{m=1}^{n} R_{m,i_{a_m}} D_m^2 + l_m(i_{a_m}) + NP(m-1, i_{a_{m-1}}) + t_{m-1}(i_{a_{m-1}}, i_{a_m}). \qquad (4.29)$$

The arriving path costs for state (n, j) for sequences S_a, S_b, and S_c can be seen in Figure 4.11.

Since the overhead cost to retrieve the necessary data from the coefficients from level $j+1$ is added to the incoming transition, the local processing cost and the number of outbound partials, $P(n, j)$, at state (n, j) will not change, regardless of the incoming path. Figure 4.12 shows the inbound/outbound costs for state (n, j) for the solution represented by S_a, S_b and S_c. Therefore, for any arriving path, outbound costs can be decoupled from inbound costs. Choosing a best path arriving to a specific state does not eliminate optimal paths, and so a path that minimizes the path metric (lowest cost) can be found using a shortest-path algorithm. At each stage n of the decision, the algorithm

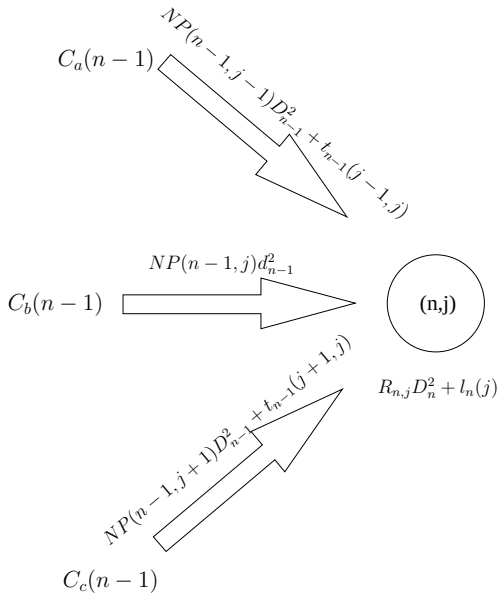


Figure 4.11: Arriving costs for state (n,j).

finds the best transition coming into each state at that stage by computing the metrics of all the possible paths coming into the state, and then selecting the path with the minimum metric as the survivor path coming into that state. At the last stage, the survivor path with the minimum path metric is selected as the optimum path. Since each state describes the coding scheme used at each of the nodes, the optimum path also provides the optimal selection of coding schemes for each sensor in the network, such that the energy consumption is minimized. A more detailed description about dynamic programming and the shortest-path algorithm can be found in [9].

4.3 Performance Evaluation

In the following simulations, we considered two different simple network configurations. We compared our optimization technique to solutions where the same coding scheme is used for all nodes in the network.

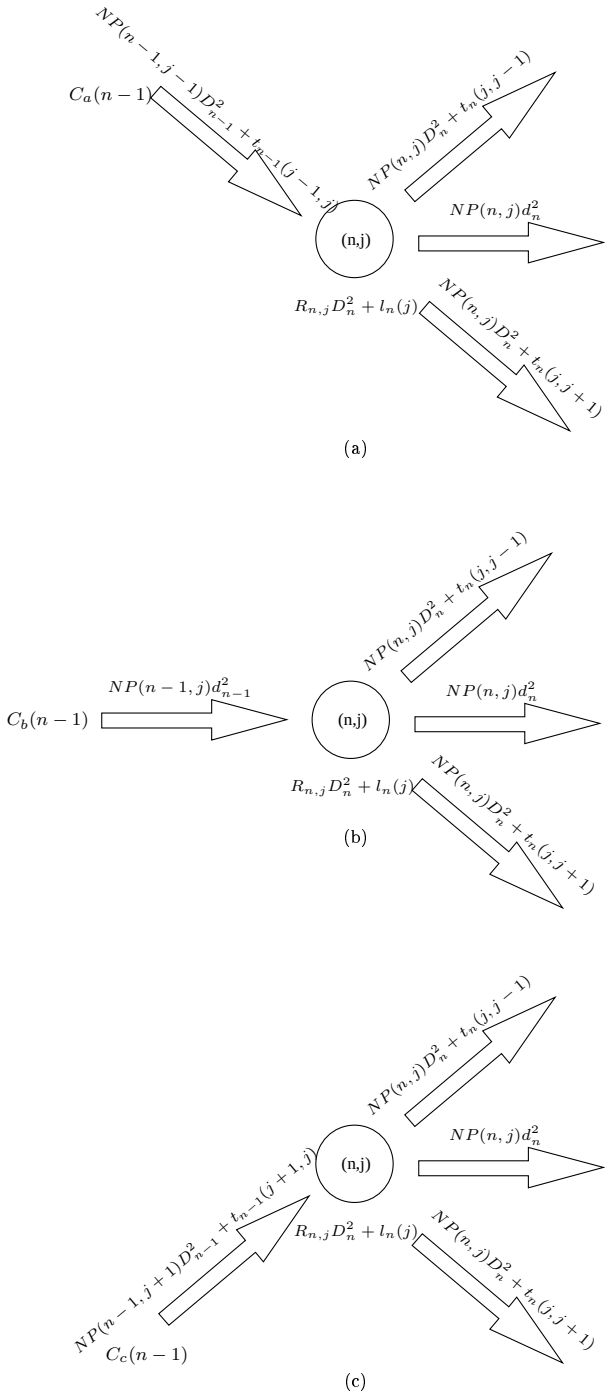


Figure 4.12: Outbound costs for state (n,j) for incoming scheme sequences (a) S_a, (b) S_b and (c) S_c.

This was done with the three available coding schemes. The coding schemes considered are raw (quantized) data transmission, wavelets with one level of decomposition, and wavelets with two levels of decompositions.

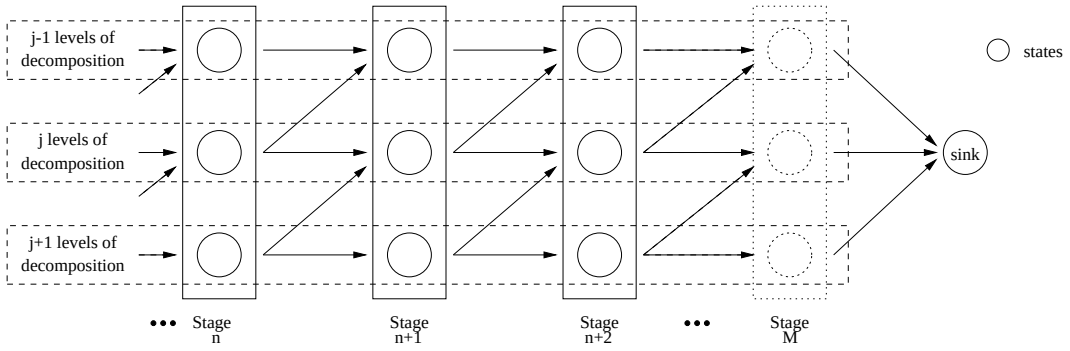


Figure 4.13: Data transitions for partial coefficient approach.

Also, without losing generality, and for simplification purposes, we constrained the number of possible transitions in the graph, as illustrated in Figure 4.13. The allowed transitions mean that as a node gets closer to the sink it can only choose between staying on the same scheme as the previous node or encoding its data using a simpler scheme, specifically one that uses one fewer level of decomposition than the current node. As motivated by the example in Section 4.1, the intuition behind this limitation is that as nodes get closer to the sink, simpler coding schemes tend to be more efficient energy-wise. This idea can also be linked to results obtained in [10, 29], where the authors addressed the problem of joint rate allocation and transmission structure optimization for sensor networks, and concluded that network regions far from the sink benefit better from methods with better compression performance, which may not be the most effective for regions close to the sink due to the computation cost overhead. This intuition is valid for cases where data correlation is approximately constant throughout the network, since

the decorrelation efficiency of the algorithms remains relatively the same compared to one another. However, when data correlation suffers substantial changes for different regions of the network no constraints on the possible scheme transitions should be made. In an extreme example, if data is completely uncorrelated at a region far from the sink, nothing will be gained by using a complex compression scheme, and simple unprocessed data transmission would be the most efficient way of sending data, since it has no overhead cost.

The input process data was created using a second order AR model, with poles placed such that a reasonably smooth output would be generated from white noise (poles were at $0.99e^{\pm j\frac{\pi}{64}}$). Figure 4.14 shows the energy consumption of different single-scheme methods (only one coding scheme for the whole network) at different distortion levels, in a network with 3 clusters of 5 sensors each (internode distance of 2m, intercluster distance of 37m).

For this network, the optimum configuration in terms of energy consumption, obtained by the proposed dynamic programming framework is shown in Figure 4.15.

Figure 4.16 shows the energy consumption of different single-scheme methods at different distortion levels, in a network with 1 cluster of 30 sensors (internode distance of 1m).

For this network, the optimum configuration obtained by the proposed dynamic programming framework is shown in Figure 4.17.

Although the results suggest changes for just a few sensors when compared to the best single-scheme method, in general, such a behavior cannot be predicted beforehand. Also, as seen in Figures 4.14 and 4.16, different single-scheme methods perform differently for different network topologies. Network performance can be affected by a number of

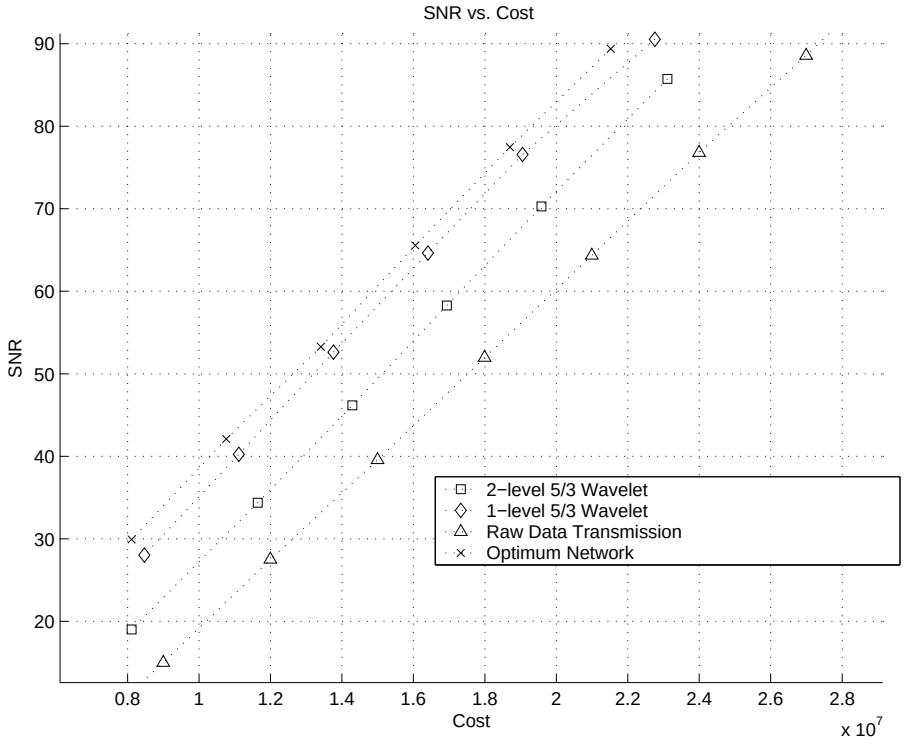


Figure 4.14: Energy consumption comparison; system with 3 clusters of 5 sensors each.

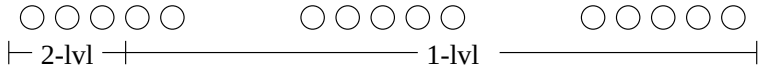


Figure 4.15: Optimum network configuration obtained for simulation in Figure 4.14.

factors, such as coding schemes being used, network topology, number of sensors, medium properties, data correlation, etc. Thus, a single-scheme approach might not necessarily result in near-optimal performance. Optimization still proves to be necessary to point out the configuration that will lead to the lowest energy cost. For the simulated case shown in Figures 4.14 and 4.16, for same distortion levels, the optimum network consumed around 6% less energy than the best single-scheme method (1-lvl wavelet for Figure 4.14 and 2-lvl wavelet for Figure 4.16) and around 32% less energy than simple raw (quantized) data transmission.

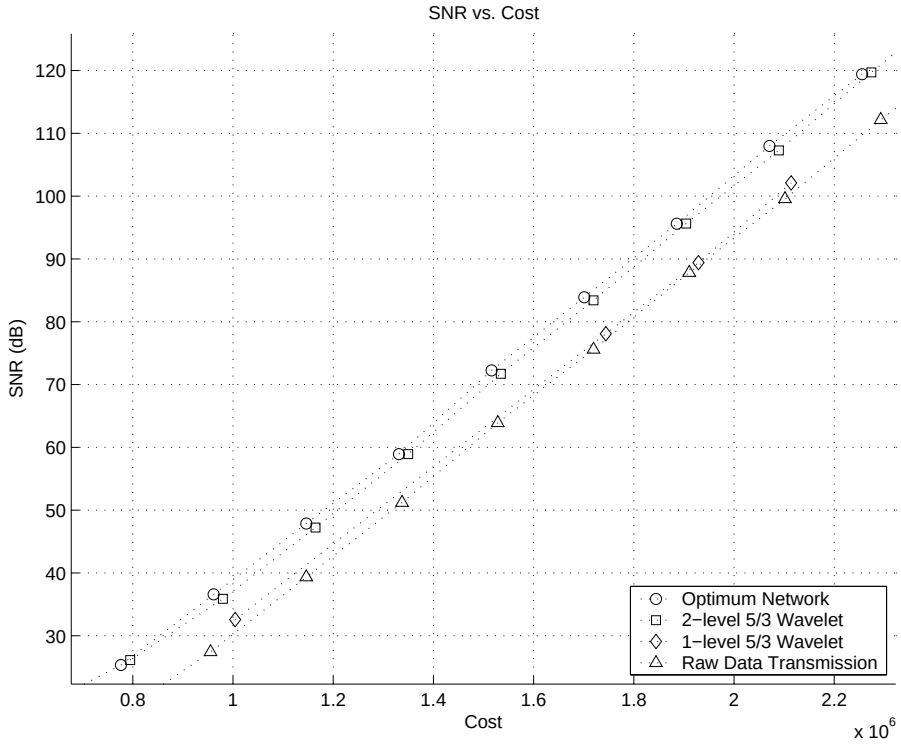


Figure 4.16: Energy consumption comparison; system with 1 cluster of 30 sensors.

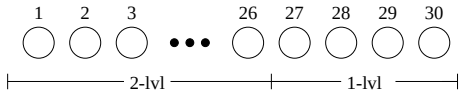


Figure 4.17: Optimum network configuration obtained for simulation in Figure 4.16.

4.4 Conclusion

In this chapter, we have proposed a dynamic programming framework that can be used to assign coding schemes to each of the nodes in a WSN such that the energy consumption in the network is minimized. This algorithm is flexible enough to accommodate any network configuration, and could also be used as a benchmark to evaluate the relative performance of fast heuristics. Simulation results have shown that different coding algorithms can perform very differently (in terms of distortion and energy consumption) depending on a number of factors such as network topology, medium properties, and data correlation,

and that the proposed methodology provides a framework that can be used to minimize energy consumption in a WSN by efficiently assigning different coding schemes to different regions of the network.

Chapter 5

Extension to 2D Networks

5.1 Introduction

In the previous chapters we introduced a distributed compression algorithm based on the lifting factorization of the wavelet transform that exploited the natural data flow in the network to aggregate data by computing partial wavelet coefficients that are refined as data flows towards the central node.

However, the partial coefficient algorithm introduced in Section 2.5 operates along a single route connecting a node to the sink, and the optimization framework from Chapter 4 minimizes the cost along this single route. In a realistic node deployment, such as the one shown in Figure 5.1, there could be many routes connecting sensors to the sink, and individual paths may merge before reaching the central node. One problem that arises in such a scenario is that if data compression occurs independently along individual paths, nodes that belong to multiple paths will generate multiple coefficients (due to merging), one for each path that uses the node to reach the sink. Different strategies can be used to address the path-merging problem. For example, multiple coefficients can be locally encoded, or alternative routing trees that reduce the number of merges can

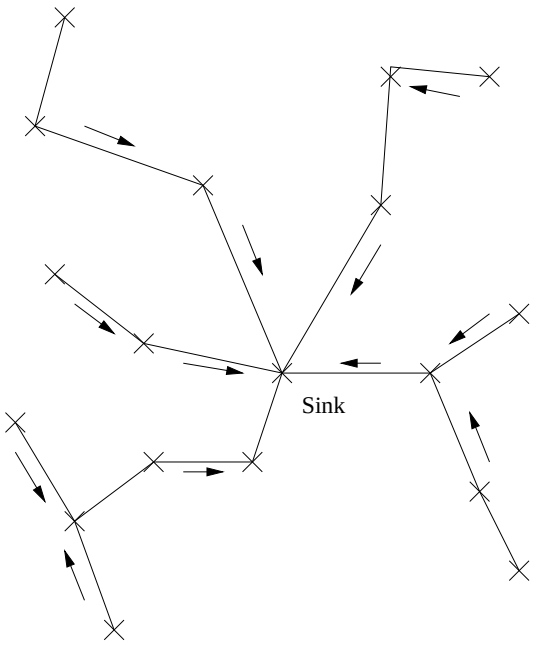


Figure 5.1: A realistic sensor deployment involves multiple routes that may merge before reaching the sink.

be used. Regardless of the strategy employed, for the partial coefficient algorithm, path-merging incurs in a cost penalty, and its impact should be considered when extending the algorithm operation to multi-route networks. Also, an optimization strategy to assign coding schemes in the multi-route scenario, as compared to individually optimizing single-routes, needs also to be defined.

In this chapter we extend the partial coefficient approach, and the network optimization using dynamic programming, to the case of multi-route networks. We propose to operate by first selecting a routing strategy throughout the network. Then, for each route, an optimal combination of data representation algorithms, i.e., algorithm assignment at each node, is selected. A simple heuristic is used to determine the data representation technique to use once path merges are taken into consideration. We demonstrate that

by optimizing the coding algorithm selection the overall energy consumption can be significantly reduced when compared to cases where data is not processed and where only a single coding scheme is used at all the nodes in the network. We evaluate the algorithm using both a second-order autoregressive (AR) model and empirical data from a real wireless sensor network deployment.

5.2 Related Work

One of the first works to propose combining routing with standard vector compression techniques was [34]. The correlated data gathering problem and the need for jointly optimizing the coding rate at nodes and routing structure is also considered in [11]. The authors provide analysis of two strategies: the Slepian-Wolf, or DSC model, for which the optimal coding is complex (needs global knowledge of correlations) and optimal routing is simple (always along a shortest path tree) and a joint entropy coding model with explicit communication for which coding is simple and optimizing routing structure is difficult. For the Slepian-Wolf model, a closed form solution is derived, while for the explicit communication case it is shown that the optimization problem is NP-complete and approximation algorithms are presented. In [29], the approach is to simplify the optimization for the explicit communication case by using an empirically obtained approximation for the joint entropy of sources. The optimal routing structure is then analyzed under this approximation. The analysis demonstrates that the optimal routing structure also depends on where the actual data compression is performed; at each individual node or at "micro-servers" acting as intermediate data collection points. In [32], "self-coding" and

"foreign-coding" are differentiated. In self-coding, a node uses data from other nodes to compress its own data, while in foreign-coding a node can also compress data from other nodes. With foreign-coding, the authors show that energy-optimal data gathering involves building a directed minimum spanning tree (DMST). For self-coding, it is shown in [11] that the optimal solution is NP-complete. Also, it is expected to exist a tradeoff between a shortest-path tree (SPT) and a traveling salesman path (TSP). Both these works assume that the data is compressed only once, after which it is decompressed at the sink. Techniques such as those described in this thesis and [1] allow compression at several hops, potentially leading to reduction in transported data. In [29], the authors explore compression at several hops and only at cluster heads and conclude that there exist efficient correlation independent routing structures. Some recent research [46] argues that the improvement from correlation aware routing is limited. Using a correlation model proposed in [12], it is shown that in terms of energy efficiency, a shortest path tree has at least 0.5 times performance compared to an optimal correlation aware routing structure. However, this result is contingent on a limited data compression model - compression gain, independent of number of neighbors and distances between nodes. In [29], [32] and [12], a detailed practical compression algorithm is not proposed, and computational costs associated with it are not considered. Depending on the signal field and the degree of spatial correlations, this cost can significantly impact the routing structure. To illustrate this impact, we present some results on a variety of network topologies and routing structures.

In closely related work [1], the authors propose using different data gathering algorithms for different classes of signal fields. They state that wavelet based processing is well-suited for deterministic signals such as piecewise constant signals, and prediction

based DPCM processing is optimal for random Gaussian correlated fields. The algorithms presented account for the combined costs of both communication (for data transport) and computation (for decorrelating, i.e., compressing raw data). Distributed and power-optimal operation of these algorithms result in division of the network into segments (or clusters) within which incurring the cost for compression is efficient. In contrast, we consider a scenario where a number of different compression schemes are available at each node. In this case, the problem is one of making a decision at each node on which compression scheme to use based on the expected computation/communication cost tradeoff. Currently, we have addressed the assignment in a two dimensional field assuming the routing structure is known, using a heuristic extension of the dynamic programming based optimal solution for a single-route path presented earlier in Chapter 4.

5.3 2D Data Representation and Optimization

In the multiple-route scenario we are considering, individual single-route paths may merge before reaching the sink. We now extend our single-route wavelet representation algorithms to the multiple-route, or 2D, case. Following the idea introduced in Chapter 4, we start by optimizing each individual path, and then we look for alternative strategies to deal with multiple coefficients generated after merge points.

5.3.1 Multiple Path Merging

Referring to Figure 5.2, assume that the same single-route wavelet transform is used along both paths. The node information captured along Paths 1 and 2 may be correlated but is not expected to be identical. Thus, in general, node n receives different wavelet

coefficients from each of the two paths. Assume first that we continue processing both paths independently, i.e., the coefficients received from Path 1 and Path 2 are updated to incorporate nodes n, $n+1$, and so on. Since these new coefficients are generated with common information, and our wavelet filters have finite memory, after several iterations, say at node $n+k$ (where k will depend on the filter length), the newly produced wavelet coefficients will be identical regardless of whether the data was originated from Path 1 or Path 2. Taking this into account, from that point forward the two sets of data can be merged, so only one set of wavelet coefficients needs to be sent from node $n+k$ onwards.

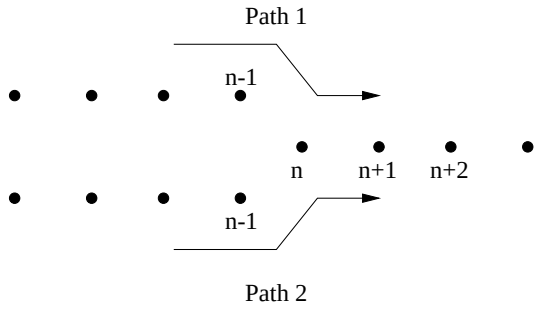


Figure 5.2: Multiple coefficients are generated at merge points.

In addition, Paths 1 and 2 up to node n are generally correlated and even if the correlation was not high, the pairs of coefficients (one per path) generated in nodes n through $n+k-1$ will become increasingly similar. This suggests a simple approach to further reduce the cost incurred in merging paths. At any node where multiple coefficients are generated (due to two or more path mergings), we encode this array of coefficients using DPCM, i.e., we compute and quantize the difference between consecutive values in the array. If the values are similar, this difference is small, and can be more efficiently represented by DPCM. This additional data is forwarded to the sink without being modified by future nodes. The rate allocation for the DPCM coefficients is done globally, i.e.,

the allocation takes into account the transform coefficients at the nodes and the multiple coefficients jointly.

As an alternative to this merge strategy, we also analyze the performance for the case where only selected multiple coefficients, namely the low-pass coefficients, are encoded using DPCM. Since the multiple high-pass coefficients are expected to have smaller energy, they can be efficiently encoded by themselves and, therefore, are not DPCM encoded. We refer to this strategy as *selective local coding*, as opposed to the global encoding strategy.

5.3.2 Proposed Algorithm

The proposed algorithm assumes a routing topology has been selected (in future work we will consider a joint selection of route and coding algorithm). Given the topology, we apply the 1D optimization of Chapter 4 (based on the partial coefficient approach) *independently* to each of the paths connecting an edge to the sink, where we consider complete paths from edge to sink, i.e., we optimize the data representation for some nodes multiple times (e.g., selection for nodes n, $n+1$, etc. in Figure 5.2 will be considered twice corresponding to Paths 1 and 2). Thus, some specific nodes, e.g., node n in Figure 5.2, could be assigned different data representations when optimized under different path configurations. When this happens we assign to the node the simplest representation (i.e., least number of levels in the wavelet decomposition). With this heuristic we force the merged paths to use the least computationally intensive among candidate algorithms.

Obviously, in cases where nodes are assigned the same algorithm by all single-route optimizations, we use the chosen algorithm and compress the merged path information as described above. Algorithm 2 describes the proposed method.

Algorithm 2 Multi-Route Algorithm

```
1) Optimization:
for each path from edge to the sink do
  -assign optimum scheme to node;
  if assignment conflicts with previous paths then
    use simpler scheme;
  end if
end for
2) Encoding:
for each path do
  encode data using optimum node assignment;
  if multiple coefficients then
    encode using DPCM;
  end if
end for
```

5.4 Routing

When the proposed approach for choosing encoding schemes is used for a two dimensional network, an extra processing cost is incurred whenever two different paths merge en route to the sink. In particular cases, as will be seen in Section 5.5, the extra processing and added information needed to represent merged paths could lead to inefficiency in the data gathering, so that a topology with fewer merges might be preferable, even if it has overall longer transmission paths.

In order to study the impact of these merges, we consider a variety of topologies, namely, degree-constrained trees. These trees are generated using a modified version of Dijkstra's algorithm over weighted graphs. Consider a graph $G(V, E)$ with edge weights $EW(i, j)$ for nodes $i, j \in V$ and edge $(i, j) \in E$, and a given degree constraint $maxDegree$. Each node n maintains a value for its best weighted path distance to the sink $WPD(n)$ (initial value $= \infty$) and the number of nodes that are its children $C(n)$ (initial value $= 0$)

in the tree rooted at the sink. Starting with a set T that initially contains the sink S, at each step we add to T the node $p \notin T$ for which

- there is a node $p' \in T$ such that edge $(p, p') \in E$,
- if $p' \neq S$ then $C(p') < maxDegree - 1$, and
- the weighted path cost to the sink $[= EW(p, p') + WPD(p')]$ is minimized.

The updates are made as $WPD(p) = WPD(p') + EW(p, p')$ and $C(p') = C(p') + 1$. The algorithm stops when $|S| = |V|$ or when no more nodes can be added (since all their neighbors have already hit the degree constraint). We avoid the latter case by considering well-connected graphs. When $maxDegree$ is greater than the maximum node degree in G, the algorithm reduces to finding the shortest weighted path and when $maxDegree = 2$, it results in long paths with no merges except at the sink. In the experiments that follow, we generate trees using the maximum number of merges M $(= maxDegree - 1)$ as a tunable parameter.

5.5 Experiments

In our experiments, we used both a second-order AR model as well as empirical data from a real wireless sensor network deployment. The simulated data consisted in generating a 600×600 2D processes using a second order AR model with high data correlation. The nodes were placed in the 600×600 grid, and their measurements corresponded to the data value from the associated position in the grid. The real data is from a subset of 19 sensor nodes from a habitat monitoring deployment [28, 41] on the Great Duck Island.

The dataset used is for 200 temperature readings taken at each sensor location on August 1, 2003 at roughly 5 minute intervals.

The sensor locations for simulated data included random sensor placement and square grid, both with 100 nodes. The routing algorithm uses degree-constrained trees generated as described in Section 5.4. The branch costs used for data representation optimization were proportional to the square of the distance between nodes, with the constant of proportionality being the number of bits allocated to the node transmitting data towards the sink.

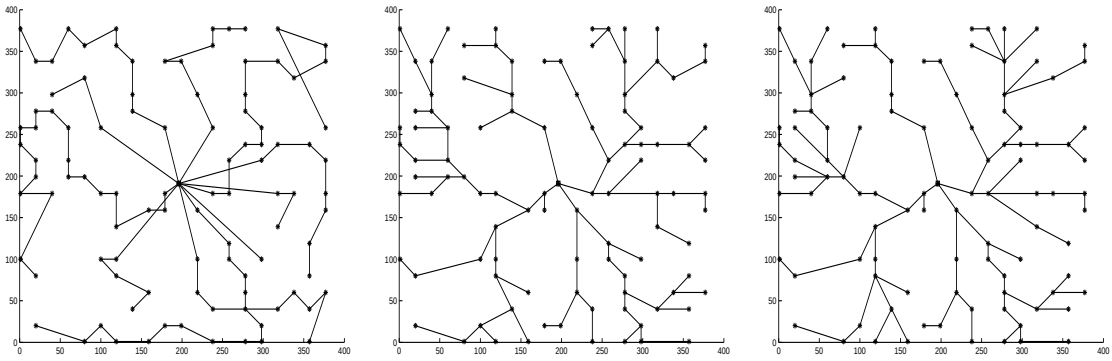
In Section 5.5.1 we compare the performances of the two merging strategies (global or selective encoding of multiple coefficients) proposed in Section 5.3.1 and address the impact of merge overhead on the final energy consumption in the network. In Section 5.5.2 we analyze the performance of the proposed method for a given routing topology. This allows us to demonstrate that gains are achievable by selecting different data representations for different nodes. Then, in Section 5.5.3, we select the data representation to be that obtained with our optimization algorithm and we compare the performance of different routing topologies, using degree-constrained trees with varying degree. For a sufficiently large degree value, the tunable tree reduces to the shortest-path tree. This allows us to demonstrate that, for certain cases, shortest path routing is in fact not optimal, as it leads to an undesirably large number of path merges.

In all network graphs that follow, nodes marked with "$\times$" are encoded with a 1-level wavelet algorithm and nodes marked with "$\circ$" are encoded with a 2-level wavelet. DPCM is represented by the symbol "$\diamond$". Raw data transmission (i.e., no compression) performance

is included in the graph for reference. The wavelet algorithms used were modified to adapt to irregular sampling, as described in Section 2.8.

5.5.1 Merging Strategy Analysis

As seen in Section 5.3.1, the proposed merging strategy increases the amount of information that needs to be transmitted over the network by creating multiple versions of coefficients at (and after) merge points. In this section we analyze the performance of the proposed merging strategies (global or selective encoding of multiple coefficients) and address the impact of the merging overhead cost in the final algorithm performance.



(a) (b) (c)

Figure 5.3: Routing strategies. (a) M=1. (b) M=2. (c) M=10.

We consider a network with 100 nodes randomly spread over the data field, and three possible routing strategies with varying degrees for the tunable tree. The three strategies are depicted in Figure 5.3 and correspond to the no-merge case (M=1), a maximum of 2 merges per node (M=2) and the shortest-path tree (M=10). Figure 5.3(a) has no merge overhead, since there are no multiple coefficients generated. While the number of multiple coefficients in Figure 5.3(c) is larger than in Figure 5.3(b), this number is not expected

to be high, since there are no significant changes in the routing trees obtained from the two strategies.

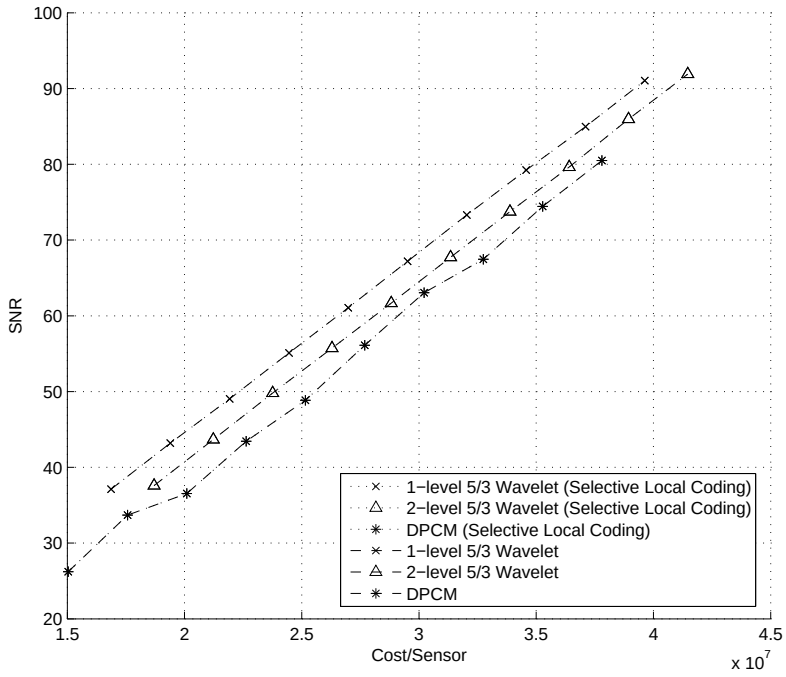


Figure 5.4: Global vs. selective encoding of multiple coefficients, M=1.

The performance comparison of the global and selective encoding of multiple coefficients can be seen in Figures 5.4 and 5.5 for the no-merge and the shortest-path routing respectively. Since in the no-merge case there are no multiple coefficients to be encoded, the performances of the two merging strategies match for each coding scheme (Figure 5.4). For the shortest-path routing, however, it can be seen (Figure 5.5) that the selective encoding methods suffer a decrease in performance when compared to the global DPCM encoding approach. This decrease in performance is related to the amount of uncoded multiple coefficients, and therefore varies with the size of the network and the number or merges, and can be attributed to the fact that the similarities among multiple coefficients

at each node still favor a local DPCM encoding. Also, due to the irregular sampling grid, the low-pass coefficients can have higher energy, conflicting with the initial assumption that they could be efficiently encoded without the need of a local DPCM. In future simulations in this section and in Sections 5.5.2 and 5.5.3 we will consider only the global multiple-coefficient encoding approach.

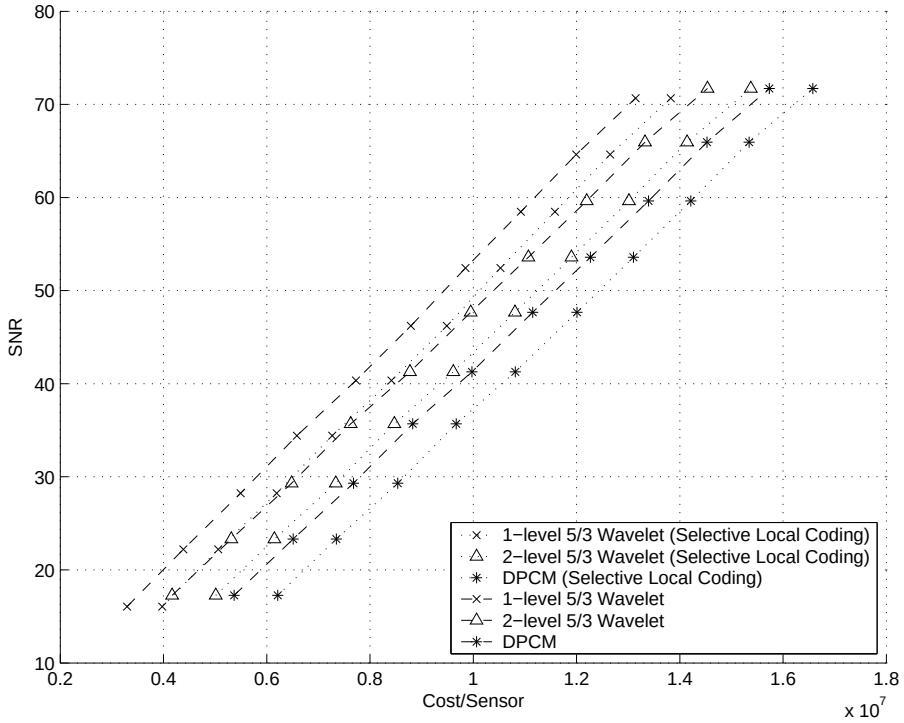


Figure 5.5: Global vs. selective encoding of multiple coefficients, M=10.

Another important aspect of the proposed merging strategy in the final cost is the amount of overhead added. The performance of the algorithm along with the no-encoding strategy (raw data transmission) is shown in Figures 5.6– 5.8.

Figures 5.6– 5.8 also show that the SNR Vs. Cost curves for different algorithms and topologies have different slopes. Different slopes can be attributed to changes in the routing strategy, which present a trade-off on the algorithms performance in terms of energy

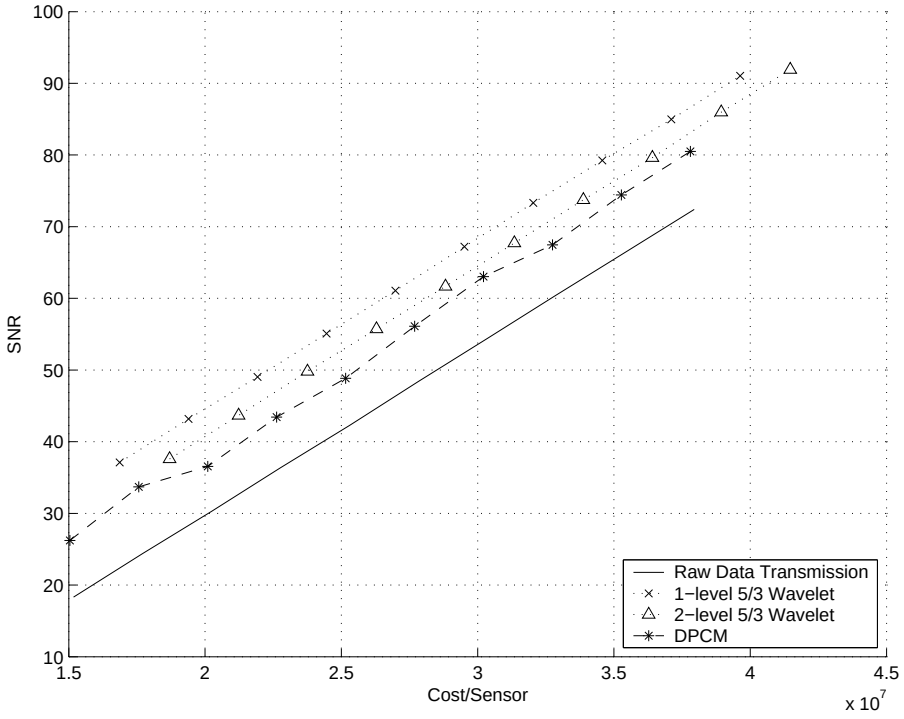


Figure 5.6: Algorithm performances using global encoding of multiple coefficients, M=1.

consumption. Strategies that allow a higher number of merges (such as the SPT) increase the overall energy consumption by generating a larger number of multiple coefficients at the merge points. Conversely, routing trees that attempt to reduce the merge overhead by limiting the number of merge points in the network, also contribute to an increase in energy consumption, by increasing the sum of the squared distance (and therefore the transmission costs) between nodes. Depending on the network topology, the overhead cost due to the increase in the distance might surpass the gain with the reduction in the number of merges, or vice-versa. Also, more complex algorithms (2-level wavelet transform) tend to have slightly lower slopes than simpler schemes (DPCM), indicating a higher number of multiple coefficients generated, as expected. We will compare the performance of different routing strategies in Section 5.5.3. While the absolute performance of the

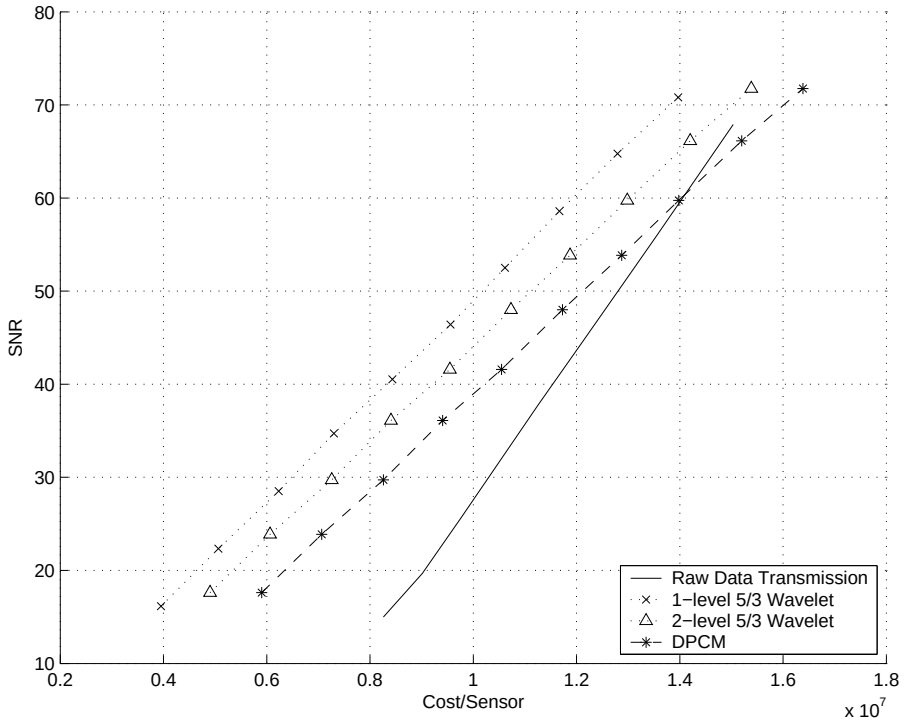


Figure 5.7: Algorithm performances using global encoding of multiple coefficients, M=2.

algorithms at a target SNR may change depending on issues like the network topology, data correlation, etc., algorithms with higher SNR Vs. Cost slopes can be thought of as being more efficient in the sense that they can achieve a better improvement in terms of SNR for a smaller additional cost, as compared to algorithms with lower slopes.

As a suggested alternative to overcome the extra penalty incurred by the multiple coefficient generation and encoding, we can consider an approach where no multiple coefficients would be generated at the merge points. This could be achieved, for example, by interrupting the encoding in a path whenever it merges to another path, as illustraded in Figure 5.9, and simply forwarding the coefficients from that point to the sink without further refining. Such a method would not suffer the problem of having to encode and transmit multiple coefficients, but it would, however, certainly increase the number

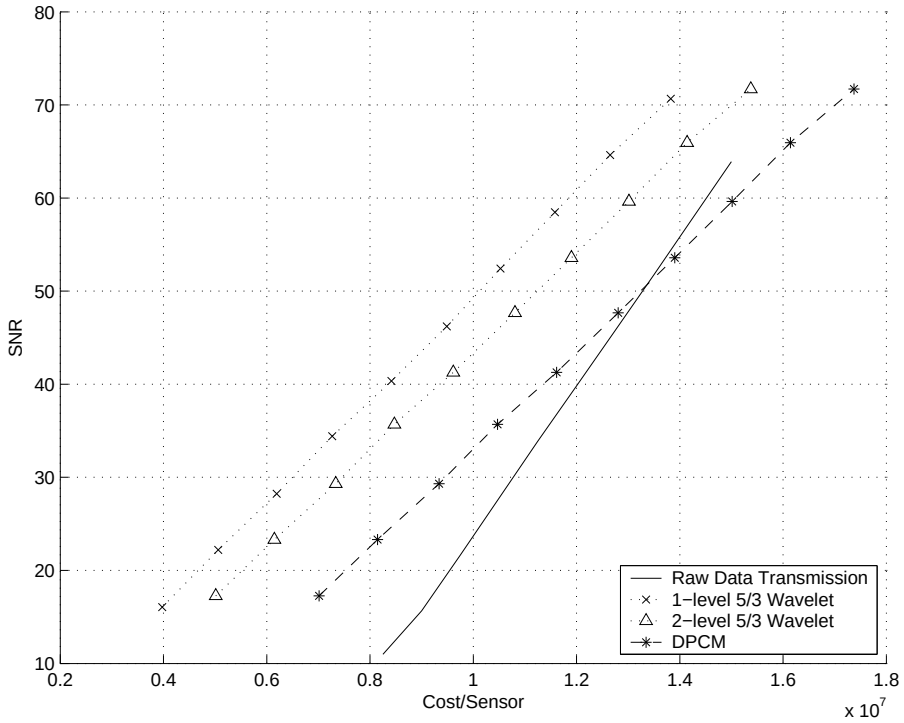


Figure 5.8: Algorithm performances using global encoding of multiple coefficients, M=10.

of partial coefficients that would not be further refined and transmitted as such to the sink. While further investigation to quantitatively compare both methods is necessary, the transmission of partials from nodes far from the sink would also incur in a significant additional cost, and this approach would possibly also suffer from routing strategies with a large number of merge points.

5.5.2 Algorithm Performance for a Given Network Topology

Figures 5.10– 5.14 illustrate the performance of the different algorithms in terms of data representation distortion (measured by Signal-to-Noise ratio, SNR) as a function of total energy consumption in the network. In each figure we depict the routing together with the data representation algorithm used for each sensor node. The energy consumption is

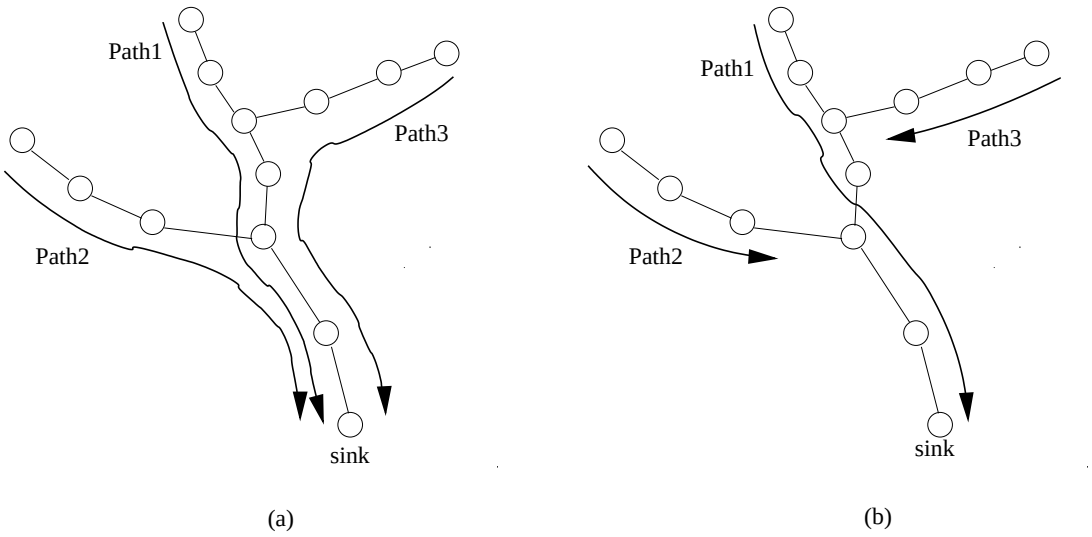


Figure 5.9: (a) Multiple-coefficient approach. (b) Interrupted-path approach.

averaged over multiple realizations of the data. The data representation itself is generated taking as an input training data generated with the AR model, i.e., we do not use the model parameters directly and our algorithm can be applied, as shown below, to optimize representation for any training data set. Observe that, in all cases, the optimized data representation, as expected, outperforms configurations where a single representation is used. Note, however, that in some of the cases the gains provided by the optimal representation are modest. This can also be seen by noting that most nodes use one representation, so the difference with respect to assigning that representation to all nodes is small.

In Figures 5.10– 5.13 nodes were uniformly distributed. Instead, in Figure 5.14 nodes were clustered to split the data field in two regions separated by a gap, and the network sink was placed in the region where the node density was lower. Observe that the behavior in Figure 5.14 is intuitively reasonable: in the cluster of nearby nodes on the top right region of the sensor field a 2 level wavelet is chosen so as to minimize the overall rate required. This is because this small cluster is far from the sink and thus any excess rate

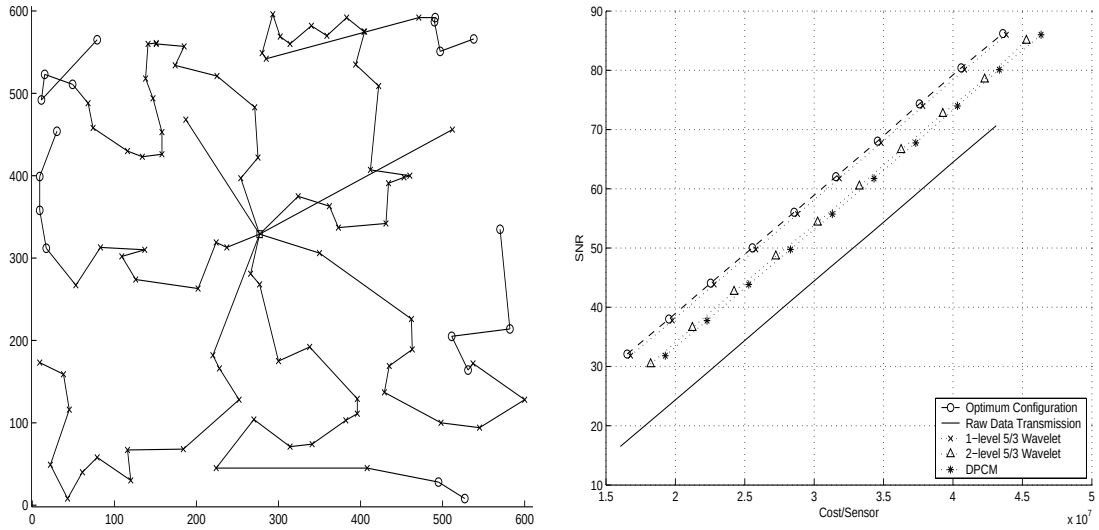


Figure 5.10: Algorithm performance for random network with no-merge routing

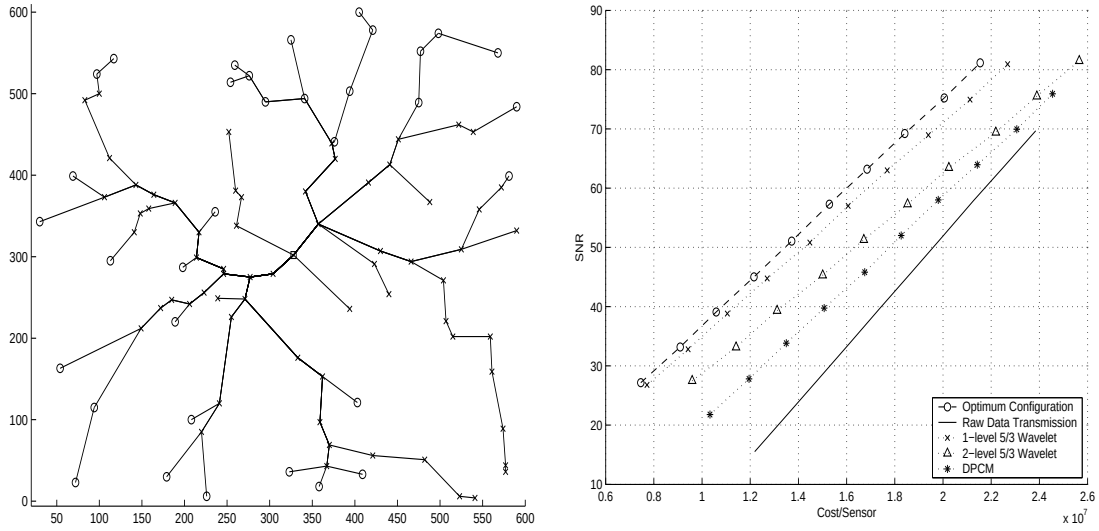


Figure 5.11: Algorithm performance for random network with shortest-path routing

in the representation (due to using a less efficient encoding) would result in significant energy required for transmission of data to the sink.

Figures 5.15 and 5.16 illustrate the performance of our methods with real data. It can be seen that when no merge is allowed (Figure 5.15), the optimum algorithm performs considerably better than the other schemes. This confirms our initial assumption that

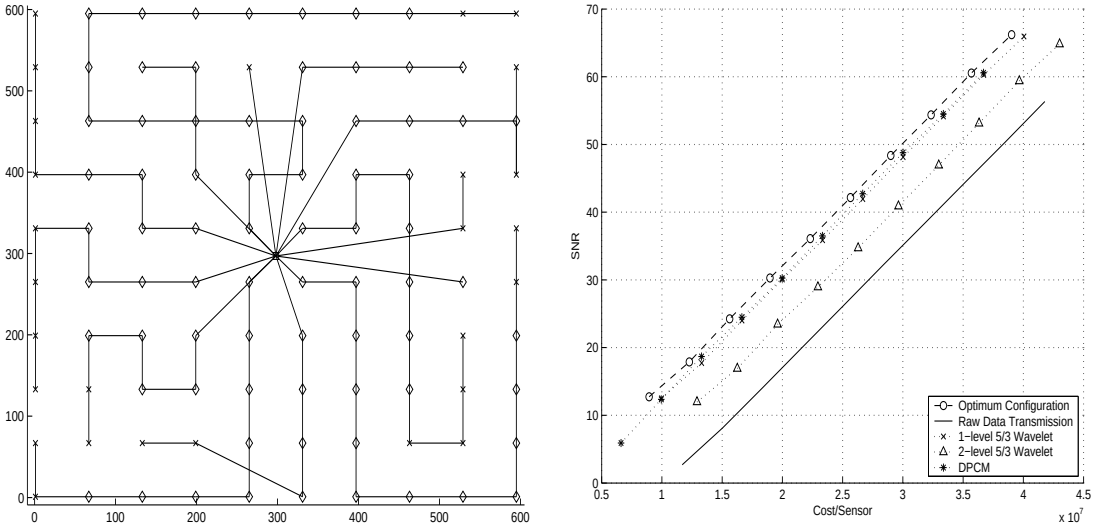


Figure 5.12: Algorithm performance for square grid network with no-merge routing.

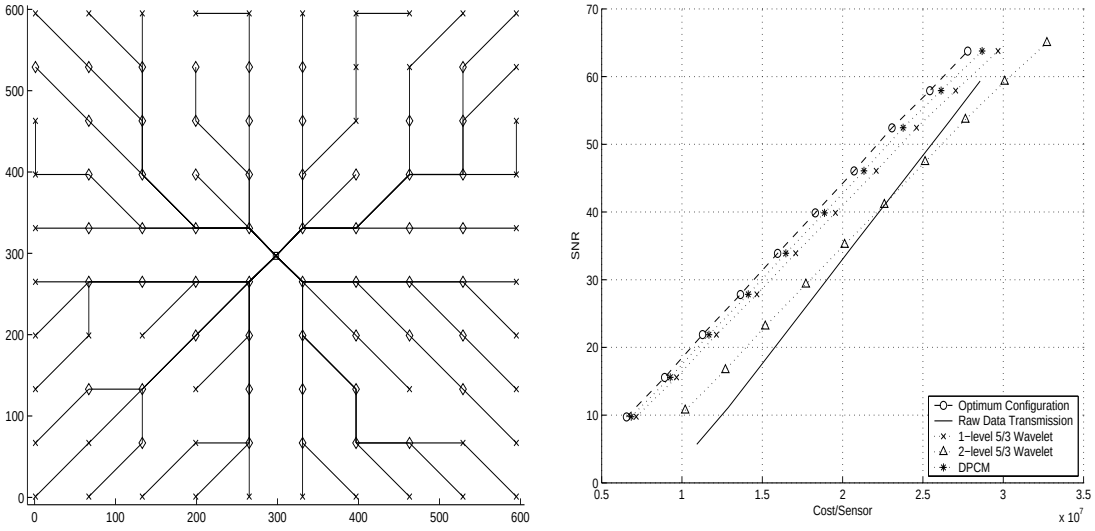


Figure 5.13: Algorithm performance for square grid network with shortest-path routing.

merging paths tend to be costly. However, as will be seen, shortest path routing still provides overall better performance.

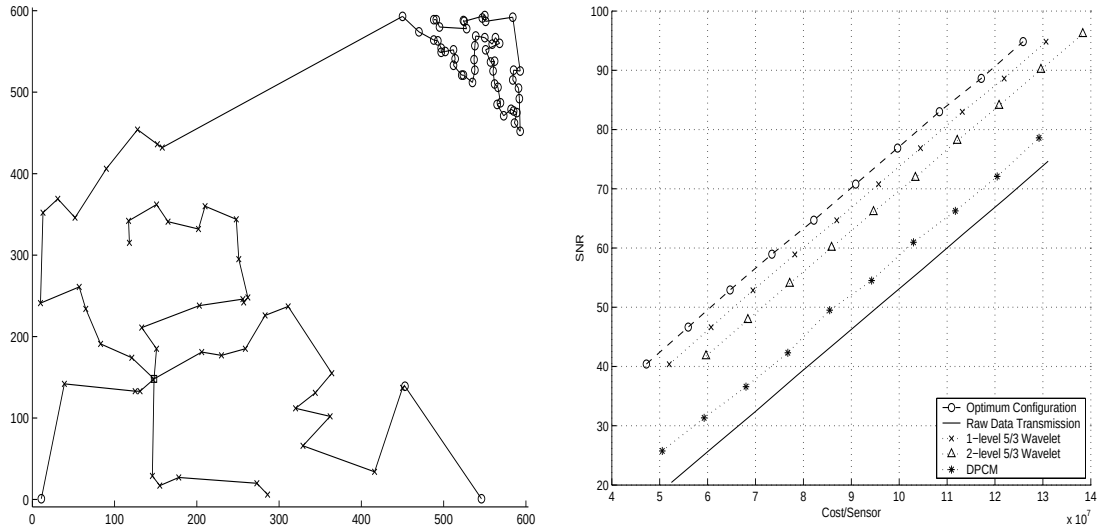


Figure 5.14: Algorithm performance for clustered network with shortest-path routing.

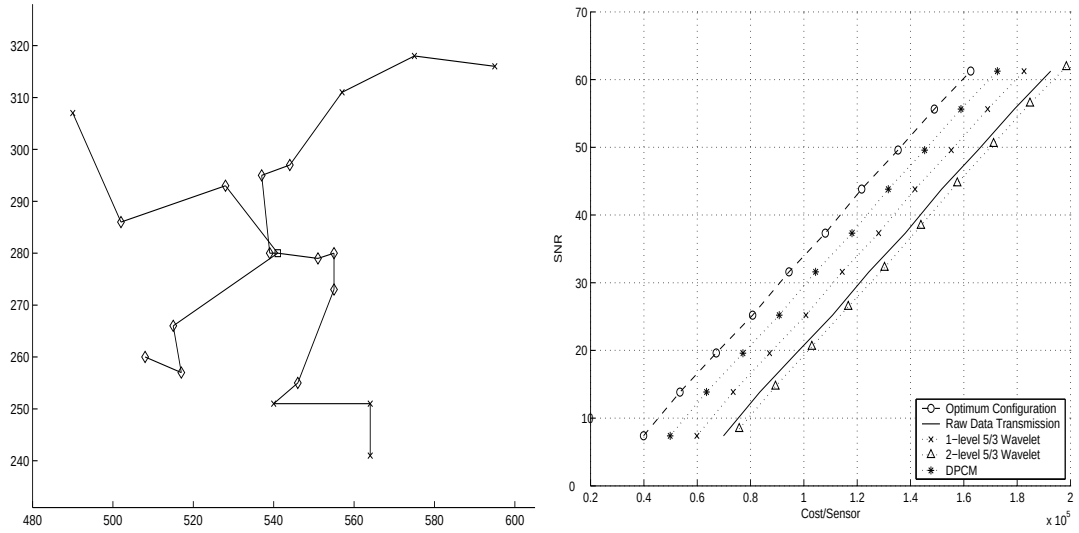


Figure 5.15: Algorithm performance for real network data with no-merge routing.

5.5.3 Routing Comparison With Optimum Network Representation

Now that we have established that data representation optimization leads to lower overall cost for a given network topology, we compare different network topologies in terms of their overall cost with optimized representation (see Figures 5.17 – 5.19). In particular we compare the performance of no-merge and shortest-path routing. As will be seen in

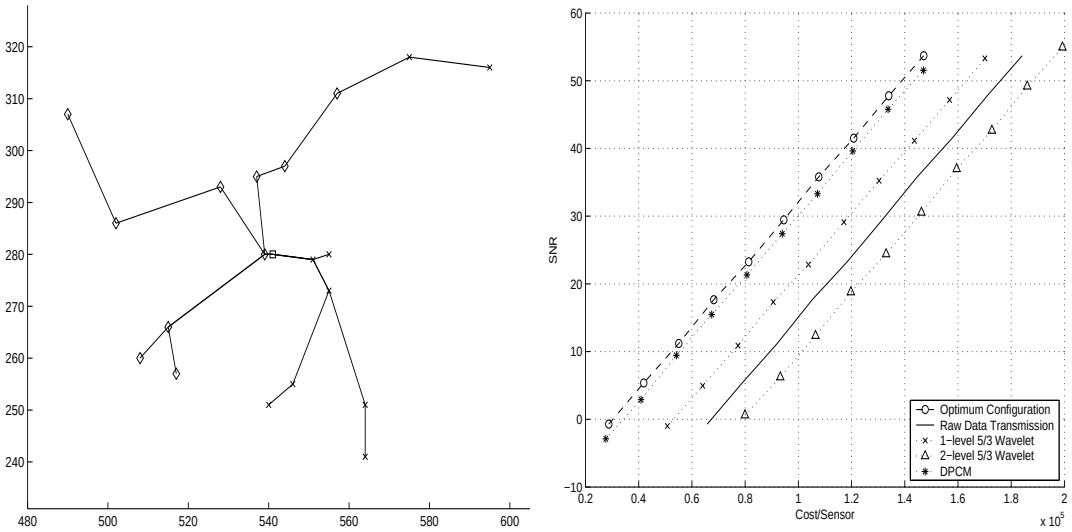


Figure 5.16: Algorithm performance for real network data with shortest-path routing.

this section, the basic intuition is that while the number of merges do imply in some overhead cost (due to the encoding of multiple coefficients), topologies that reduce the number of merges increase the average internode distance, also increasing the transmission cost between nodes and, therefore whether having fewer merges is better depends on how much the extra distance is, and what the cost of the merges was for each specific network topology.

For both simulated random network data with uniform sampling (Figure 5.17) and real data (Figure 5.18), we can see that shortest path routing leads to better overall performance. For the denser network (simulated data) overall performance is very sensitive to the average length of the single-route paths in the network, so that the gains from using shortest-path routing are substantial. In this case, for the shortest-path routing, 20% of the nodes in the network were merge nodes, while the no-merge routing increased the average distance between nodes in the network by 63%.

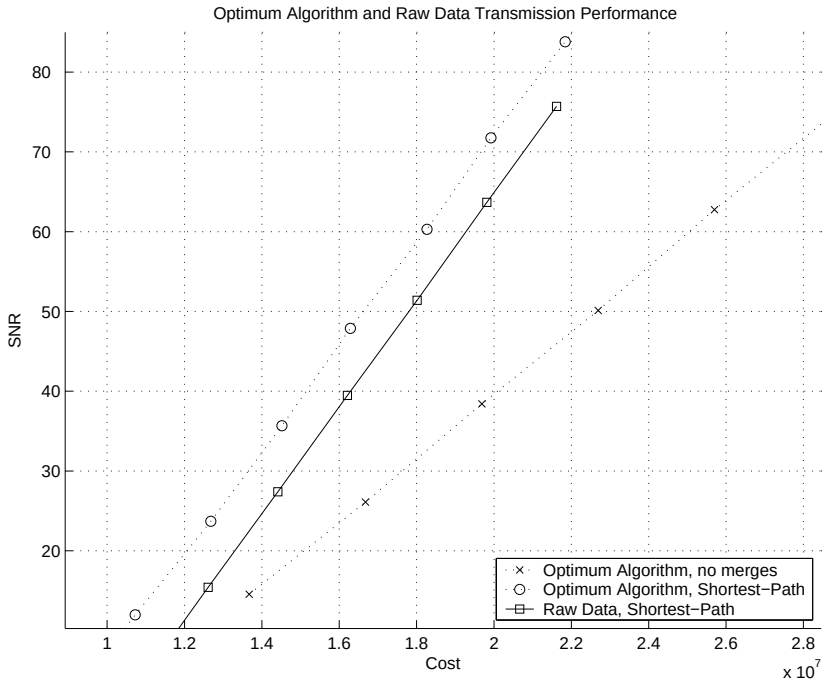


Figure 5.17: Optimum algorithm performance compared to raw data transmission for no-merge and shortest-path routing for random network

In contrast, for the smaller real-data network (Figure 5.18), performance is not as significantly affected by the routing, due to the reduced number of nodes. We believe that in the case of Figure 5.17, given the density of the network the cost of merging is not as significant (as a result of the proposed DPCM coding of merged paths) and therefore the penalty due to allowing longer paths (as in the no-merge configuration) is too high to be compensated by the reduction of merge costs. It is also interesting to notice that the no-merge and shortest-path routing have similar performance at low SNR (this can be seen in the real-data case, which we operate at lower rates). This can be explained as follows: in shortest-path routing information needs to be sent for every merge, for a number of nodes dependent on the filter length. At low rates this overhead becomes more significant as the data transmission rates in no-merge routing become lower.

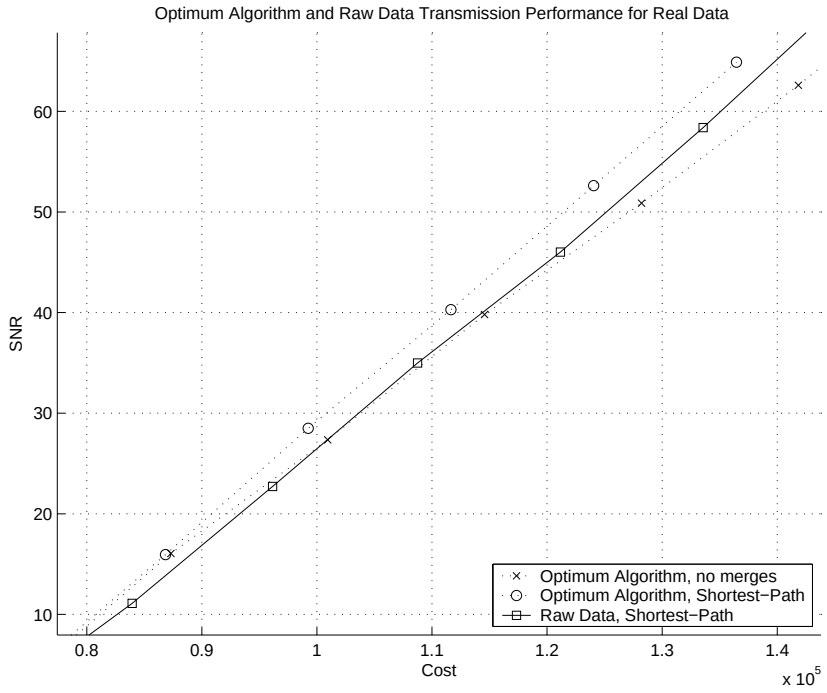


Figure 5.18: Optimum algorithm performance compared to raw data transmission for no-merge and shortest-path routing for real data network

Figure 5.19 represents results for the clustered network scenario. Note that in this case, unlike the two preceding ones, no-merge routing leads to the best overall performance. In this scenario, as in the random network case, 20% of the nodes are merge nodes when shortest-path routing is used. However, the no-merge routing increases the average distance between nodes in only 30% as compared to the random case, while also offering a significant gain in terms of compression performance by more efficiently grouping correlated nodes.

Note that in Figures 5.17, 5.18, and 5.19 the performance curve slope is lower for no-merge routing (i.e., the quality degrades more slowly as the transmission cost decreases). We again attribute this to additional cost of data merging in the shortest path routing case.

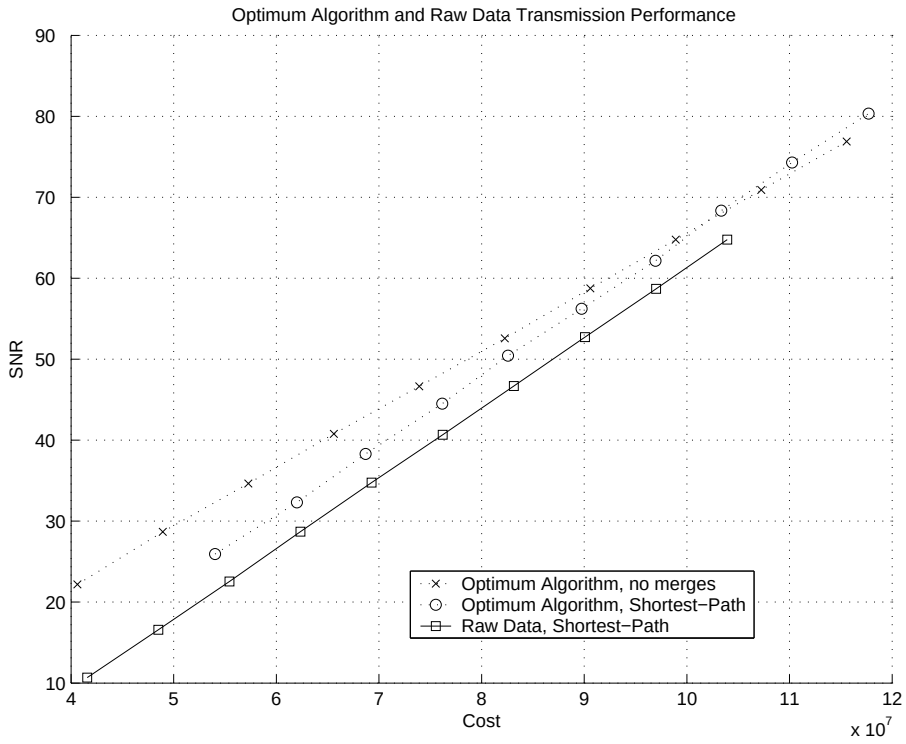


Figure 5.19: Optimum algorithm performance compared to raw data transmission for no-merge and shortest-path routing for clustered network

5.6 Interpolatory Wavelet Transform

Among the many related work discussed in Section 5.2, one deserves special attention. The research group at Rice University has recently proposed an interpolatory wavelet transform for irregular sampling grids [43], also based on the lifting scheme.

Their approach is based on a multiscale representation, with cascade pairs of lifting stages, similar to what has been described in Section 2.3. Their most relevant contribution, however, was the introduction of a true bi-dimensional wavelet decomposition based on lifting. The bi-dimensional idea was an extension of the lifting scheme for unidimensional spaces, and relied on splitting the nodes in 2D space into even and odd sets. For example, a so called "even" node would grow its neighborhood until it had enough neighbors to

guarantee a stabilized predictor operator (assuring perfect reconstruction is possible). Those neighbors would then belong to the "odd" set of nodes. The process would continue with the odd nodes finding their even neighbors and so on, at any given scale. After even and odd sets are defined at the different scales, the prediction and update operations of the lifting decompositions would take place, generating the wavelet decomposition coefficients at each node. We refer to [43] for details.

While their work proposes a true 2D wavelet transform (in our work, we have a 1D transform that encodes 2D data but operating along 1D paths), they suffer from a major setback. Each stage of the lifting decomposition implies that all partial data involved in that step has to transmitted between even and odd nodes as described by the corresponding matrix. Therefore, even relatively small values of m would imply a large number of transmissions, increasing the transmission costs. In our work, this problem is solved with the partial coefficient approach, which eliminates the need of backward transmissions. In [43], they overcome this issue by discarding detail information below a given threshold, greatly reducing the required number of transmissions and processing.

The work in [43] also provides an energy consumption performance analysis, showing significant improvements when compared to the direct transmission of original data.

Even though the work proposed in [43] still suffers from a significant energy overhead imposed by the large number of required transmissions, and due to this, at this point, is only viable for very large networks (with more than 1000 nodes), it is interesting to note its similarities with the work proposed in this thesis (distributed wavelets based on the lifting scheme with energy consumption constraints). It seems that many of the ideas proposed in these two works can be applied to each other, allowing a performance improvement

from both parts. Our work could certainly benefit from a true 2D lifting approach, and applying the partial coefficient idea to a true 2D algorithm will open new possibilities to the research, making this a strong topic to be explored in the near future.

5.7 Conclusion

In this work, we have provided a framework that allows finding, for a given network topology, which among a number of available coding methods is more suitable for each of the sensors. We have shown that by optimizing the coding algorithm selection, the overall energy consumption can be significantly reduced when compared to the case when data is just quantized and forwarded to the central node. In our simulations we were able to compare different routing techniques and identify those that are most efficient overall, for given node locations, and for both simulated data and from a real wireless sensor network deployment. Even for the cases where the proposed distributed wavelet transform algorithms were not as efficient as simpler schemes such as DPCM (see Figures 5.12 and 5.13), we verified that the proposed optimum algorithm always finds the best coding scheme that should be used by each node in the network, and that shortest-path routing does not always leads to best performance.

A number of topics can still be addressed. While in this work the algorithm analysis was limited to predefined routing topologies, joint selection of route and coding algorithm seems to be a natural next step for optimization. Also, more realistic cost functions can be incorporated to the decision process (e.g., transmission costs a constant term in addition to a term proportional to distance), and a more efficient representation of natural

phenomena (e.g. different regions of the field are modeled with different parameters) can be used. While in this work we assumed nodes exchanged information by data hoping, in a practical application broadcast capabilities could be used to improve even more the network efficiency. For example, coefficients in the paths that were already fully computed could reach the central node by broadcasting, eventually skipping hops and saving more energy; nodes that need a measurement from a common sensor could acquire it via broadcasting in one single step, also reducing the energy consumption.

Chapter 6

Conclusion and Future Work

In this thesis we have addressed the problem of compression for wireless sensor networks from a signal processing point of view. We have proposed two novel algorithms for the distributed wavelet transform as a means to decorrelate data and more efficiently represent sensor measurements, requiring fewer bits and reducing the energy consumption in those networks.

The first method consisted in a distributed implementation of the discrete wavelet transform based on the lifting scheme. In the second, we proposed the computation of partial wavelet coefficients to exploit the natural data flow direction in the network and eliminate unnecessary transmissions. We evaluated the impact of quantization of transmitted data in the final distortion, defining a rule to design sensor quantizers, and verified that encoding partial data using 3 extra bits offered a reasonable trade-off between additional distortion and extra energy cost.

We also proposed a generic optimization procedure based on dynamic programming to configure the network, dividing it into regions that would process data using different coding schemes, such that the final energy consumption is minimized. Depending on the filter sizes and number of levels of decomposition, energy costs can become very high.

However, one the main points of this work is not only simply providing a distributed wavelet algorithm, but one that is cost-aware. Depending on the application, a large number of sensors might be involved, and more complex transforms might offer a more efficient way of representing data. We provided a method to decide if a given wavelet transform using a certain number of levels of decompositions will lead to energy savings for a given sensor network compared to other possible coding schemes.

We finished by proposing an extension of the distributed wavelet transforms to be implemented in 2D networks, with a generic topology. A result of this extension was that a wavelet compression algorithm could be used to represent 2D data obtained from an irregular sampling pattern.

Experiments with both simulated and real data have shown that the proposed methods can significantly reduce the energy consumption in wireless multihop sensor networks. However, compression for wireless sensor networks is still a relatively new research area, and many problems are still open. Despite the fact that this work has achieved interesting results and proposed a number of novel strategies, there are still relevant topics or ideas that can be addressed in future research, and are worth mentioning.

- **Merging Strategy.** The heuristic for processing data at merging points in a multiple-route network proposed in section 5.3.1 is very sensitive to the number of merge points in the network. The amount of side information in the form of DPCM coefficients can become very large depending on the number of ramifications in the routing tree for a given network. While searching for alternative routings that can improve the algorithm performance is valid approach, one can argue that

the problem lies in the merging strategy itself. New strategies, such as the one suggested in Section 5.3.1, could avoid the increase in side information by providing alternative ways of combining data at the merging points (heuristic or not).

- **Exploiting Broadcast Capabilities.** Transmissions in a wireless sensor network are done via broadcasting. This means that when a node is transmitting its data, all the nodes inside its range can also have access to the data being transmitted. This could allow a higher volume of data exchange without any increase in energy consumption, and could have an impact on the algorithm performance. Also, in cases where the optimal routing does not follow a shortest-path tree for data processing, broadcasting could be used to forward data that does not need to be further processed faster to the sink, thus reducing energy consumption.

- **Quantization.** Even though the theoretical bound obtained in chapter 4 is shown to be satisfactory in practice, a more detailed study of the quantization effects might be necessary. Topics to be addressed can include the sequential quantization of more terms in the sum, and the effect of different quantizers other than uniform;

- **Vector data at sensors.** Measurements at the nodes can consist of vector data, meaning that sensors are acquiring data over time. In this case, besides spatial correlation, time correlation at each node can also be exploited.

Bibliography

[1] J. Acimovic, B. Beferull-Lozano, and R. Cristescu. Adaptive distributed algorithms for power-efficient data gathering in sensor networks. In *2005 International Conference on Wireless Networks, Communications and Mobile Computing*, June 2005.

[2] W. Chen, S. Itoh, and J. Shiki. Irregular sampling theorems for wavelet subspaces. *IEEE Transactions on Information Theory*, 44(3):1131–1142, May 1998.

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

[4] A. Ciancio and A. Ortega. A distributed wavelet compression algorithm for wireless sensor networks using lifting. In *Proceedings of the 2004 International Conference on Acoustics, Speech and Signal Processing - ICASSP04*, Montreal, Canada, May 2004.

[5] A. Ciancio and A. Ortega. A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting. In *Proceedings of the 2005 International Conference on Acoustics, Speech and Signal Processing - ICASSP05*, Philadelphia, USA, March 2005.

[6] A. Ciancio and A. Ortega. A dynamic programming approach to distortion-energy optimization for distributed wavelet compression with applications to data gathering in wireless sensor networks. *International Conference on Acoustics, Speech and Signal Processing - ICASSP06*, 2006.

[7] A. Ciancio, S. Pattem, A. Ortega, and B. Krishnamachari. Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet comression algorithm. *International Conference on Information Processing in Sensor Networks - IPSN06*, 2006.

[8] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure and Applied Mathematics*, 42:485–550, 1992.

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

[10] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. Networked Slepian-Wolf: Theory and algorithms. *1st European Workshop on Sensor Networks EWSN 2004*, 2004. Berlin, Germany.

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

[12] R. Cristescu, B. Beferull-Lozano, M. Vetterli, and R. Wattenhofer. Network correlated data gathering with explicit communication: Np-completeness and algorithms. *To appear in IEEE/ACM Transactions on Networking*, 2005.

[13] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[14] N. Farvardin and J. Modestino. Rate-distortion performance of DPCM schemes for autoregressive sources. *IEEE Transactions on Information Theory*, 31:402–418, May 1985.

[15] M. Feilner, D. Van De Ville, and M. Unser. An orthogonal family of quincunx wavelets with continuously adjustable order. *IEEE Transactions on Image Processing*, 14(4):499–510, 2005.

[16] M. Gastpar, P. Dragotti, and M. Vetterli. The distributed Karhunen-Loève transform. In *Proceedings of the 2002 International Workshop on Multimedia Signal Processing*, St. Thomas, US Virgin Islands, December 2002.

[17] A. Gersho and R.M. Gray. *Vector Quantization and Signal Processing.* Kluwer Academic publishers, 1992.

[18] A. Goel and D. Estrin. Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. In *SODA*, pages 499–505, 2003.

[19] A. Gouze, M. Antonini, M. Barlaud, and B. Macq. Optimized lifting scheme for two-dimensional quincunx sampling images. In *Proceedings of the 2001 International Conference on Image Processing, ICIP'01*, Thessaloniki, Greece, October 2001.

[20] A. Hayashi. Differential pulse code modulation of stationary gaussian inputs. *IEEE Transactions on Communication*, COMM-26:1137–1147, Aug. 1978.

[21] W. R. Heinzelman, A. Sinha, A. Wang, and A. P. Chandraksan. Energy-scalable algorithms and protocols for wireless microsensor networks. *ICASSP - International Conference on Acoustics, Speech, and Signal Processing*, June 2000.

[22] E. Janardhanan. Differential PCM systems. *IEEE Trans. Commun.*, COMM-27:82–93, Jan. 1979.

[23] N. S. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video.* Prentice-HAll, 1984.

[24] W. Jiang and A. Ortega. Lifting factorization-based discrete wavelet transform architecture design. *IEEE Transactions on Circuits ans Systems for Video Technology*, 11(5):651–657, May 2001.

[25] J. Kusuma, L. Doherty, and K. Ramchandran. Distributed compression for sensor networks. *ICIP-International Conference on Image Processing*, October 2001.

[26] S. Lindsey and C. S. Raghavendra. PEGASIS: Power-efficient gathering in sensor information systems. In *In proceedings of the IEEE Aerospace Conference*, Los Angeles, USA, 2002.

[27] Y. Liu. Irregular sampling for spline wavelet subspaces. *IEEE Transactions on Information Theory*, 42(2):623–627, March 1996.

[28] Habitat Monitoring on Great Duck Island. Online data-set located at<http://www.greatduckisland.net>.

[29] S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, April 2004.

[30] Charles E. Perkins. *Ad-hoc Networking.* Addison-Wesley, 2001.

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

[32] P.V. Rickenbach and R.Wattenhofer. Gathering correlated data in sensor networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing, Philadelphia, PA, USA*, pages 60–66. ACM, October 2004.

[33] Khalid Sayood. *Introduction to Data Compression.* Morgan Kaufmann Publishers, 1996.

[34] A. Scaglione and S. D. Servetto. On the interdependence of routing and data compression in multi-hop networks. *International Conference on Mobile Computing and Networking - Mobicom'02*, 2002.

[35] Sergio D. Servetto. Distributed signal processing algorithms for the sensor broadcast problem. *Conference on Information Sciences and Systems, The Johns Hopkins University*, March 2003.

[36] Sergio D. Servetto. Sensing lena - massively distributed compression of sensor images. *ICIP - International Conference on Image Compression*, September 2003.

[37] G. Strang and T. Nguyen. *Wavelets and Filter Banks.* Wellesley, Cambridge, 1996.

[38] W. Sweldens. The lifting scheme: A construction of second generation wavelets. Technical report 1995:6, Industrial Mathematics Initiative, Department of Mathematics, University of South Carolina, (ftp://ftp.math.sc.edu/pub/imi_95/imi95_6.ps), 1995.

[39] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[40] W. Sweldens and P. Schröder. Building your own wavelets at home. *Wavelets in Computer Graphics, ACM SIGGRAPH Course notes*, pages 15–87, 1996.

[41] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2004.

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

[43] R. S. Wagner, R. G. Baraniuk, S. Du, D. B. Johnson, and A. Cohen. An architecture for distributed wavelet analysis and processing in sensor networks. *International Conference on Information Processing in Sensor Networks - IPSN06*, 2006.

[44] A. Wang and A. Chandraksan. Energy-efficient dsps for wireless sensor networks. *IEEE Sigal Processing Magazine*, pages 68–78, July 2002.

[45] A. Wang and A. Chandraksan. Energy-efficient DSPs for wireless sensor networks. *IEEE Signal Processing Magazine*, pages 68–78, July 2002.

[46] 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 Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), Santa Clara, California, USA,*, September 2005.

Appendix

Lifting Factorization of Generic Filters

In chapter 1.4 we make use of the lifting factorization to propose distributed wavelet algorithms. The folowing sections are based in what is presented in [13], and describe how a generic pair of wavelet filters can be factorized into lifting steps.

A.1 Filters and Laurent Polynomials

Consider a filter h with an impulse response given by $\{h_k \in \mathbf{R} \mid k \in \mathbf{Z}\}$. Assume that h has a finite impulse response (FIR), i.e. h has a finite number of non-zero coefficients. The z-transform of h is a Laurent polynomial $h(z)$ and is given by

$$h(z) = \sum_{k=k_a}^{k_b} h_k z^{-k}.$$

The degree of a Laurent polynomial is defined as

$$|h| = b - a$$

Therefore, if z^n is seen as a Laurent polynomial it has degree zero, while if seen as a regular polynomial it has degree n. The degree of the zero polynomial is set by definition to $-\infty$.

While the sum, difference and product of two Laurent polynomials are still Laurent polynomials, exact division is not possible in general. However, division with remainder is. Consider two Laurent polynomials $a(z)$ and $b(z) \neq 0$, with $|a(z)| \geq |b(z)|$. Then, there always exist Laurent polynomials $q(z)$ (quotient) and $r(z)$ (remainder) with $|r(z)| < |b(z)|$ such that

$$a(z) = b(z)q(z) + r(z).$$

$q(z)$ and $r(z)$ can be denoted as

$$q(z) = a(z)/b(z) \quad \text{and} \quad r(z) = a(z)\%b(z)$$

If $b(z)$ is a monomial ($|b(z)| = 0$), then the division is exact ($r(z) = 0$). A Laurent polynomialis invertible if and only if it is a monomial. Long division of Laurent polynomials is not necessarily unique, as seen in the following example.

Example 1. We wish to divide two Laurent polynomials $a(z)$ and $b(z)$, where $a(z) = z^{-1}+6+z$, and $b(z) = 4+4z$. $q(z)$ must be of degree one, so that $r(z) = a(z)-b(z)q(z)$ has degree zero. This means that $b(z)q(z)$ has to match $a(z)$ in two terms. We, therefore, have three possible solutions for the division, depending on which terms we choose to match.

If we choose to match the terms in z^{-1} and the constant, we have $q(z) = \frac{1}{4}(5 + z^{-1})$, and $r(z)$ is of degree zero:

$$r(z) = (z^{-1} + 6 + z) - (4 + 4z)(1/4z^{-1} + 5/4) = -4z.$$

Choosing to match the terms in z and z^{-1} we find $q(z) = \frac{1}{4}(z^{-1} + 1)$, and

$$r(z) = (z^{-1} + 6 + z) - (4 + 4z)(1/4z^{-1} + 1/4) = 4.$$

Finaly, matching the constant and the term in z gives $q(z) = \frac{1}{4}(5z^{-1} + 1)$, and $r(z) = -4z^{-1}$. □

A.2 Primal and Dual Lifting

A pair of filters h and g is said to be *complementary* if the associated polyphase matrix has determinant 1. Because of the perfect reconstruction condition, wavelet filter pairs are always complementary. If the pair (h, g) is complementary, so is the pair $(\tilde{h}, \tilde{g})$.

For any complementary pair of filters (h, g), the *lifting theorem* [13] states that any other finite filter g^{new} complementary to h is of the form

$$g^{new} = g(z) + h(z)s(z^2) \tag{A.30}$$

where $s(z^2)$ is a Laurent polynomial. In polyphase domain, a new polyphase matrix is created as

$$P^{new}(z) = \begin{bmatrix} h_e(z) & h_e(z)s(z) + g_e(z) \\ h_o(z) & h_o(z)s(z) + g_o(z) \end{bmatrix} = P(z) \begin{bmatrix} 1 & s(z) \\ 0 & 1 \end{bmatrix} \tag{A.31}$$

Similarly, the lifting step also creates the filter $\tilde{h}^{new}(z)$, complementary to $\tilde{g}(z)$

$$\tilde{h}^{new} = \tilde{h}(z) + \tilde{g}(z)\tilde{s}(z^2) \tag{A.32}$$

with

$$\tilde{P}^{new}(z) = \begin{bmatrix} \tilde{h}_e(z) + \tilde{g}_e(z)\tilde{s}(z) & \tilde{h}_o(z) + \tilde{g}_o(z)\tilde{s}(z) \\ \tilde{g}_e(z) & \tilde{g}_o(z) \end{bmatrix} = \begin{bmatrix} 1 & \tilde{s}(z) \\ 0 & 1 \end{bmatrix} \tilde{P}(z) \tag{A.33}$$

For the dual lifting step, the equations become

$$h^{new} = h(z) + g(z)t(z^2) \tag{A.34}$$

$$P^{new}(z) = \begin{bmatrix} h_e(z) + g_e(z)t(z) & g_e(z) \\ h_o(z) + g_o(z)t(z) & g_o(z) \end{bmatrix} = P(z) \begin{bmatrix} 1 & 0 \\ t(z) & 1 \end{bmatrix} \tag{A.35}$$

and

$$\tilde{g}^{new} = \tilde{g}(z) + \tilde{h}(z)\tilde{t}(z^2) \tag{A.36}$$

$$\tilde{P}^{new}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{g}_e(z) + \tilde{h}_e(z)\tilde{t}(z) & \tilde{g}_o(z) + \tilde{h}_o(z)\tilde{t}(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \tilde{t}(z) & 1 \end{bmatrix} \tilde{P}(z) \qquad \text{(A.37)}$$

The previous equation can be interpreted as if a given wavelet, represented by a polyphase matrix $P(z)$, were *lifted* to a more sophisticated level but each of the lifting steps. In the next section, the same analogy is applied to perform the go in the opposite direction: starting with a complex filter, decompose it into elementary steps.

A.3 Decomposition Into Lifting Steps

Starting, for instance, with equation A.36 we can write

$$\tilde{g}(z) = \tilde{h}(z)\tilde{t}(z^2) + \tilde{g}^{new}. \qquad \text{(A.38)}$$

The for of equation A.38 is identical to a long division with remainder of Laurent polynomials, where $\tilde{g}^{new}(z)$ is the remainder. Rewriting equation A.37, we obtain

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e(z) & \tilde{h}_o(z) \\ \tilde{h}_e(z)\tilde{t}(z)\tilde{g}_e^{new}(z) & \tilde{h}_o(z)\tilde{t}(z) + \tilde{g}_o^{new}(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \tilde{t}(z) & 1 \end{bmatrix} \tilde{P}^{new}(z) \qquad \text{(A.39)}$$

and we see that, starting with the original polyphase matrix, two long divisions are needed in order to extract one lifting step. After this step is extracted, the algorithm can continue by extracting more lifting steps from the new polyphase matrix until a diagonal matrix

is found. In [13], it is proven that starting with a complementary filter pair (h, g), it is always possible to factor $P(z)$ into lifting steps:

$$P(z) = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix} \prod_{i=1}^{m} \begin{bmatrix} 1 & s_i(z) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z) & 1 \end{bmatrix} \tag{A.40}$$

A.3.1 Euclidean Algorithm

In this section we describe the Euclidean algorithm, used to perform the long division of the Laurent polynomials. The Euclidean algorithm was originaly developed to find the greatest common divisor (*gcd*) of two natural numbers, but can be extended to Laurent polynomials.

Euclidean Algorithm. Let $a(z)$ and $b(z) \neq 0$ be two Laurent polynomials with $|a(z)| > |b(z)|$. Make $a_0(z) = a(z)$ and $b_0(z) = b(z)$ and iterate the following steps, starting with $i = 0$

$$\begin{aligned} a_{i+1}(z) &= b_i(z) \\ b_{i+1}(z) &= a_i(z) \% b_i(z) \\ q_{i+1}(z) &= a_i(z) / b_i(z) \end{aligned}$$

Then $a_n(z) = gcd(a(z), b(z))$, where n is the smallest number for which $b_n(z) = 0$. The result of the algorithm can be written as

$$\begin{bmatrix} a(z) \\ b(z) \end{bmatrix} = \prod_{i=1}^{n} \begin{bmatrix} q_i(z) & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_n(z) \\ 0 \end{bmatrix} \square \tag{A.41}$$

A.3.2 Example

Assume we are given the following wavelet filter pair

$$\begin{aligned}\tilde{h}(z) &= -\tfrac{1}{8}z^{-2} + \tfrac{1}{4}z^{-1} + \tfrac{3}{4} + \tfrac{1}{4}z - \tfrac{1}{8}z^{2} \\ \tilde{g}(z) &= \tfrac{1}{4}z^{-2} - \tfrac{1}{2}z^{-1} + \tfrac{1}{4}\end{aligned}$$

With corresponding polyphase matrix is given by

$$\tilde{P}(z) = \begin{bmatrix} -\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z & \frac{1}{4} + \frac{1}{4}z \\ \frac{1}{4}z^{-1} + \frac{1}{4} & -\frac{1}{2} \end{bmatrix}$$

For the factorization of the polyphase matrix, we can start with the extraction of a dual lifting step:

$$\tilde{P}(z) = \begin{bmatrix} \tilde{h}_e^{new}(z) & \frac{1}{4} + \frac{1}{4}z \\ \tilde{g}_e^{new}(z) & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \tilde{t}(z) & 1 \end{bmatrix} \tilde{P}^{new}(z) \tag{A.42}$$

The two divisions to be solved are

$$-\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z = \tilde{t}(z)\left(\frac{1}{4} + \frac{1}{4}z\right) + \tilde{h}_e^{new}(z)$$

and

$$\frac{1}{4}z^{-1} + \frac{1}{4} = \tilde{t}(z)\left(-\frac{1}{2}\right) + \tilde{g}_e^{new}(z).$$

Using the Euclidean algorithm with $a_0(z) = \tilde{h}_e(z)$ and $b_0(z) = \tilde{h}_o(z)$ we have the following options for the division, depending on which two terms of $a_0(z)$ are chosen to match with $b_0(z)$:

$$-\frac{1}{8}z^{-1} + \frac{3}{4} - \frac{1}{8}z = \begin{cases} \left(-\frac{1}{2}z^{-1} + \frac{7}{2}\right)\left(\frac{1}{4} + \frac{1}{4}z\right) - z \\ \left(-\frac{1}{2}z^{-1} - \frac{1}{2}\right)\left(\frac{1}{4} + \frac{1}{4}z\right) + 1 \\ \left(\frac{7}{2}z^{-1} - \frac{1}{2}\right)\left(\frac{1}{4} + \frac{1}{4}z\right) - z^{-1} \end{cases} \tag{A.43}$$

Choosing the symetric term, we arrive at the following decomposition:

$$\tilde{P}(z) = \begin{bmatrix} 1 & \frac{1}{4} + \frac{1}{4}z \\ 0 & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix} \tag{A.44}$$

Applying the Euclidean algorithm to $\tilde{g}_e(z)$ and $\tilde{g}_o(z)$ of equation A.44, we find:

$$\tilde{P}(z) = \begin{bmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{4} + \frac{1}{4}z \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2}z^{-1} - \frac{1}{2} & 1 \end{bmatrix} \tag{A.45}$$

This equation gives a fully factorized version of the given filters.