Globally Optimal Algorithms for Multiple-Transform Signal Compression

by

Lucas Nissenbaum

B.S. ECE, Cornell University, 2013

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2015

© Massachusetts Institute of Technology 2015. All rights reserved.

Signature redacted

Author .............................................................

Department of Electrical Engineering and Computer Science

July 24, 2015

Signature redacted

Certified by........

\[\text{Signature redacted}\]

Professor Jae S. Lim

Professor of Electrical Engineering

Thesis Supervisor

Signature redacted

Accepted by .....\

\[\text{Signature redacted}\]

Professor Leslie A. Kolodziejski

Chairman, Department Committee on Graduate Students
Globally Optimal Algorithms for Multiple-Transform Signal Compression

by

Lucas Nissenbaum

Submitted to the Department of Electrical Engineering and Computer Science on July 24, 2015, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering and Computer Science

Abstract

In video compression, a single transform such as the DCT is typically used. Multiple-transforms such as directional 1-D DCTs have been proposed to exploit the different statistical characteristics of motion compensation residuals. Many issues are associated with this scenario. In this thesis, we will focus on the issue of selecting the appropriate number of coefficients and transforms to be allocated for each block of the signal to optimize the energy compaction.

We propose two new methods to select optimal transforms for different blocks in a signal. The first method is based on thresholding, while the second method is based on dynamic programming and related to the multiple-choice knapsack problem. These algorithms are then compared to two other previous algorithms. We analyze the energy compaction performance of these algorithms in terms of different block-sizes and different input characteristics.

We then extend all of these algorithms to quantized coefficients being transmitted, as well as to take a bit-rate constraint into account.

Thesis Supervisor: Professor Jae S. Lim
Title: Professor of Electrical Engineering
Acknowledgments

First and foremost, I would like to thank my research advisor, Professor Jae Lim, for all his mentoring and guidance during my M.S. research. I am very grateful for all the opportunities he has provided me here at MIT. The intellectual guidance he has given me over these two years has been invaluable and unique, and it has impacted not only my research, but also my life as a whole.

I would also like to thank all of my labmates at ATSP. I feel sincerely grateful to Megan Fuller and Xun Cai for not only listening and discussing my good and bad ideas, but also for taking their time to help me understand the field of image processing and for discussing many random mathematical problems. I also feel extremely grateful to Cindy LeBlanc, for all the support and patience in my time here at MIT.

I would like to thank my family in Brazil, for ensuring my sanity since I moved to the United States. I especially thank my parents and my sister, for all their love and care, for visiting me so often, and for giving me support whenever I needed them. I also thank Deusdita for all her care, love, and support since my first years.

I wish also to thank my friends for all their friendship, trust, and for all experiences we have shared. I especially thank Alfredo, Antonio, Bernardo, Carlo, Hugo, Leonardo, and Rafael for being such good friends in my time in Brazil, and keeping in touch since I have been abroad. I also thank Carlos, Christina, and Nicole, for being next to me in my time abroad and for all of the dinners and trips we have had together.

I wish also to thank all my friends at MIT for listening to me through my good and bad jokes, especially Anuran Makur, Gregory Stein, James Noraky, and Pablo Martinez.

Thank you all.
Contents

1 Introduction ............................................. 13
   1.1 Energy Compaction .................................. 14
   1.2 Multiple-Transform Signal Encoding ............... 17
   1.3 Overview of The Thesis ............................ 19

2 Previous Research ........................................ 21
   2.1 Multiple-Transform Video Encoding .................. 21
   2.2 Algorithm A: Iterative Method ...................... 24
   2.3 Algorithm B: Rate-Distortion Optimization ........ 26
   2.4 Motivation of Thesis ................................ 30

3 Algorithm C: Thresholding ................................ 31
   3.1 Optimality ........................................... 32
   3.2 Complexity .......................................... 35

4 Algorithm D: Dynamic Programming ....................... 37
   4.1 Optimality ........................................... 41
   4.2 Complexity .......................................... 43
   4.3 Multiple-choice Knapsack Problem .................. 44
      4.3.1 The 0-1 Knapsack Problem ..................... 44
      4.3.2 The Multiple-choice Knapsack Problem ......... 45

5 Algorithm Comparison ..................................... 49
   5.1 Experimental Setup .................................. 49
List of Figures

1-1 General scheme for block-based video encoder .......................... 14
1-2 Example in reconstruction of an image using the $8 \times 8$ 2-D DCT and
only a fraction of the original image. ...................................... 16
1-3 General scheme for multiple-transform block-based video encoder 18

2-1 Example of frame and motion-compensated residuals for frame. .... 22
2-2 Possible directions for $4 \times 4$ directional 1-D DCT ................... 23
2-3 Possible directions for $8 \times 8$ directional 1-D DCT ................... 24
2-4 Implementation of Algorithm B for each block ......................... 29

3-1 Implementation of Algorithm C for each block ......................... 31

5-1 First frame of video sequences to be encoded. ......................... 50
5-2 First frame of motion compensated residuals to be encoded. ....... 51
5-3 Fraction of energy kept as a function of the fraction of coefficients kept
for $8 \times 8$ blocks of motion compensated residuals. .................. 52
5-4 Zoom for comparison between algorithms B and D when encoding the
football motion compensation residuals from figure 5-3c. .............. 53
5-5 Comparison between algorithms A and D when encoding the football
motion compensation residuals and using a small number of coefficients. 54
5-6 Fraction of energy kept as a function of the fraction of coefficients kept
for $4 \times 4$ blocks of motion compensated residuals. .................. 55
5-7 Fraction of energy kept as a function of the fraction of coefficients kept
for $16 \times 16$ blocks of motion compensated residuals. ................ 56
5-8  Fraction of energy kept as a function of the fraction of coefficients kept for $8 \times 8$ blocks of first frame of video sequences. ........................................ 57

6-1  Setup with truncation and quantization. ................................................. 59

6-2  Mid-thread uniform quantizer with step-size 2. ...................................... 61
## List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example of transform coefficients in magnitude-squared.</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of block optimal energy function for signal in table 2.1.</td>
<td>28</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of profit function for signal in table 2.1 when $\lambda = 4$.</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of transform coefficients in magnitude-squared.</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Example of block optimal energy function for signal in table 4.1.</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Example of algorithm C for signal in table 4.1.</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Example of transform coefficients for second block in magnitude-squared.</td>
<td>39</td>
</tr>
<tr>
<td>4.5</td>
<td>Example of block optimal energy function for signal in table 4.4.</td>
<td>40</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of algorithm D for two blocks.</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>Algorithm comparison as a function of type of optimality, convergence,</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>complexity without the transform operation, and parallelizable over</td>
<td></td>
</tr>
<tr>
<td></td>
<td>different blocks.</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Example of transform coefficients.</td>
<td>61</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of quantized transform coefficients.</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Algorithm comparison with quantization and under a constraint on</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>the number of bits as a function of type of optimality, convergence,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>complexity without the transform operation, and parallelizable over</td>
<td></td>
</tr>
<tr>
<td></td>
<td>different blocks.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

One major objective of signal coding is to represent this signal with as few bits as possible. Two kinds of coding may be done for a signal: lossless coding or lossy coding. In lossless coding, the objective is to use the smallest number of bits necessary to transmit this residual without losing any information. In other words, the decoder must be able to retrieve the original signal exactly. In lossy coding, we allow some loss of the original information, and minimize this loss subject to a bit-rate constraint. Accepting this loss in information leads to a very significant reduction in bit-rate when compared to the lossless case. In this thesis, we will focus on lossy signal coding.

Lossy encoding of a signal is generally performed in a block-based manner. First, the signal is divided into blocks. These blocks are then predicted, and the difference between the original signal and the prediction, called the residual signal, is transmitted. This prediction step is done to remove as much information as possible from the previously processed blocks. The residual signal is then processed, and the processed residual signal is transmitted. In image and video coding, we predict and process the signal to have the smallest mean-squared error (MSE) under a small bit-rate constraint achieved through transform coding. We then represent the signal in bits through entropy coding, so we can transmit it. This block-based procedure is used because it can take into account the different local characteristics of the original signal.

In modern video compression standards, such as H.264 [1] and HEVC [2], pre-
diction may be done in two ways. First we may use previously encoded blocks of the current frame only, called intra-prediction. Second we may predict from blocks of other encoded frames, called inter-prediction or motion compensation. A frame exclusively intra-predicted is called an i-frame. A frame predicted that is motion compensated from the previous frames and intra-predicted from blocks in the current frame is called a p-frame. A frame motion compensated from the previous and future frames, as well as intra-predicted from blocks in the current frame, is called a b-frame.

Since prediction usually leads to a non-zero error, we need to transmit the residual of this prediction. To send the largest amount of energy with as few coefficients as possible, we transform the residual block. In many video coding systems, transforms such as the two-dimensional (2-D) discrete cosine transform (DCT) [3] or the integer DCT [4] are used. After transforming the residual signal, we quantize the obtained coefficients. Entropy coding follows, where the coefficients are converted to a small number of bits for transmission. Figure 1-1 shows a picture of the general block-based video encoder.

![General scheme for block-based video encoder.](image)

**Figure 1-1:** General scheme for block-based video encoder.

### 1.1 Energy Compaction

In the context of transform coding, a natural question is how to evaluate a transform with respect of encoding the signal. A reasonable way to evaluate a transform is to compute its energy as a function of the number of coefficients that are kept over all the blocks of this signal. For many image and video signals, an appropriate transform will accurately represent most of the energy of this signal with a small fraction of the coefficients. A transform that can compact most of the energy of a signal using only a small number of coefficients is frequently desirable for compression systems. This property is called the energy compaction property [5].
In many image compression systems, the 2-D DCT is used since it provides good energy compaction. As an example, we consider dividing the image in figure 1-2a into $8 \times 8$ blocks, and transforming each block using the $8 \times 8$ normalized 2-D DCT. We then keep the largest coefficients, and evaluate our compression as a function of the fraction of coefficients kept. In figure 1-2b, we see that there is barely any visible difference if we keep the 15% largest coefficients. In figure 1-2c, we see that keeping the 5% largest coefficients will still give us a clear image, although we can now notice some encoding artifacts, such as on the shoulder. In figure 1-2d, we see that the MSE decreases rapidly as we increase the number of coefficients.

We now express this mathematically. Let $x$ be our original signal and $T$ be our transform. Throughout this thesis, we will assume all of our transforms to be orthonormal. We consider the problem of finding the approximation $\hat{x}$ of $x$ with the smallest mean-square error, given a sparsity constraint on $\hat{x}$. This sparsity constraint is given by a maximum number $c$ of non-zero coefficients in $\hat{x}$ under the transform $T$. We let $||a||_0$ be the number of non-zero elements in $a$. In this case, our problem becomes:

$$\min_{||T\hat{x}||_0\leq c} ||x - \hat{x}||_2^2$$

(1.1)

The solution to this problem is found by applying the energy equation for the orthonormal transform. We can rewrite our problem as:

$$\min_{||T\hat{x}||_0\leq c} ||T(x - \hat{x})||_2^2$$

(1.2)

In this case, we then note the solution will simply be to keep the $c$ largest coefficients in magnitude-squared from the transformed signal to find $\hat{x}$. Similarly, if we consider that we now have $x$ as formed by many blocks $x_i$, and transform each block by the same transform $T$, then we write our optimization problem as:

$$\min_{\sum_i ||T\hat{x}_i||_0\leq c} \sum_i ||x_i - \hat{x}_i||_2^2$$

(1.3)
Using the same method as before to solve this problem, we find that the same solution will hold by keeping the $c$ largest coefficients in magnitude-squared over all transformed blocks.

Throughout this thesis, we use the mean-squared error with respect to how many
coefficients are kept as a metric to evaluate our methods. There are many disadvantages in doing so. First, the number of bits obtained by compression is not directly proportional to the number of coefficients, since it is also influenced by the quantizer used and by the number of bits reduced with entropy coding. We note truncation may not be the best model for quantization, but it is often used due to mathematical simplicity and may be used in other applications. If a transform does not have a good performance with respect to truncations, it will most likely not be a good transform for the general signal encoding with quantization. We will in this thesis see how we can adapt our results to a bit-rate constraint instead.

Second, we note the MSE is not an ideal metric of performance in many applications. Consider image compression, and let us examine two ways to modify a 512×512 image. The first method removes 7 from every pixel in the image, which incurs an MSE of $7^2 \times 512 \times 512 = 12845056$. This change of the image mean will not significantly affect our view of the image, since it will just make it slightly darker. The second method removes an 8×8 block of the image, which implies an MSE smaller than $256^2 \times 8 \times 8 = 4194304$, but will be noticed by any viewer. In this example, the first modification is less harmful than the second, although it has a significantly larger MSE.

Many metrics have been proposed that take user experience into account [6, 7]. We will not focus on these metrics since they are harder to treat mathematically. Many of these metrics are related to or based on weighted mean-square error, which one may in the future adapt these methods to.

1.2 Multiple-Transform Signal Encoding

In multiple transform encoding, we consider instead the case where we have $K$ orthonormal transforms, denoted by $T^{(1)}, \ldots, T^{(K)}$. For each block $i$ of our signal, denoted by $x_i$, we wish to choose the transform $T_i$ and the approximation $\hat{x}_i$, that best represents our signal. By appropriately choosing a set of transforms, we may have a significant improvement in the performance of our coding system [8, 9]. Under this
scheme, we are transmitting not only the transformed coefficients for each block, but also an index indicating the chosen transform. This scheme is illustrated in figure 1-3.

We now see an example of why such a scheme may be useful. Assume we have two transforms, $T(1)$ given by the impulse basis and $T(2)$ as the normalized 1-D DCT, and that the multiple-transform scheme is used to encode two blocks of a one-dimensional signal $x$ divided into blocks of length $M = 10$. If the first block of $x$ is $x_1(n) = \delta(n-3)$, then we can transmit it with one coefficient using the impulse basis, but need ten coefficients if we use the 1-D DCT basis. If the second block of $x$ is $x_2(n) = 3$ for all $n$, then we can transmit it with one coefficient using the 1-D DCT, but need ten coefficients for the impulse basis. In the multiple transform scheme, we only use two coefficients to encode the two blocks with no error. Meanwhile, choosing to use a fixed transform will require eleven coefficients to achieve the same error.

A natural question that arises is why we use multiple transforms, instead of finding statistical characteristics of an image and using a transform such as the Karhunen-Loeve transform (KLT) [5]. The KLT uses the statistical properties of the signal to find the $k$ orthonormal coefficients that will on average best represent this signal. There are many reasons why multiple-transforms may be more successful than the KLT. First, we note the KLT is the best transform given that we can only keep $k$ fixed coefficients, not varying for each block. Second, we note in many signals we do not have globally stationary statistical properties to find a best transform. In this sense, by choosing the transforms appropriately, multiple transforms may take into account the KLT of many possible auto-correlation functions from different blocks of our signal, requiring the random process to be only locally stationary.
Another question one may ask is what the difference is between using multiple-transforms and using an overcomplete representation [10] to encode the signal. An overcomplete representation for an N-dimensional signal is a spanning set \( \{ \phi_1, \ldots, \phi_M \} \) for this signal space. We note an overcomplete representation is equivalent to multiple-transforms where a transform represents each possible combination of vectors from the spanning set. This gives us a total of \( \binom{M}{N} \) transforms. This assumes that we do not restrict our transforms to be orthonormal. For signal compression, the main disadvantage of using an overcomplete representation is that the number of vectors in the overcomplete basis is very large. Therefore too many bits are spent to indicate the vectors used in the approximation.

A significant issue one faces in the multiple-transform signal compression scenario is how to determine what the best transform is for a given block. Mathematically, the objective is to find the transform in each block \( i, T_i \) from the set \( \{ T^{(1)}, \ldots, T^{(K)} \} \), and the number of coefficients \( c_i \) that should be allocated for each block, such that the total number of coefficients is less than \( c \). In this sense, we can find \( \tilde{x} \) to minimize the mean-square error. This may then be written as the following optimization problem:

\[
\min_{T_1, \ldots, T_N, \sum_i \|T_i\|_0 \leq c} \sum_i \|x_i - \tilde{x}_i\|_2^2
\]  

This problem has led to two algorithms in [11], that solve it under different optimality conditions. This thesis will present two new algorithms to solve it.

1.3 Overview of The Thesis

Chapter 2 presents an overview of the previous research in multiple-transform signal encoding. We first present the system in [12], and show some of the results found when encoding motion compensation residuals using multiple-transforms. We also present the two different algorithms used to determine the best multiple transforms for a full signal presented in [11], called algorithm A and algorithm B.

In chapter 3, we consider an alternative approach to algorithm B based on thresh-
olding, called algorithm C. In chapter 4, we consider a more general algorithm to solve the multiple-transform problem under a fixed number of coefficients based on dynamic programming and the multiple-choice knapsack problem. In chapter 5, we compare these algorithms, and present our results. In chapter 6, we study how to change these algorithms to take quantization into account.

Finally, in chapter 7, we conclude by presenting possible directions of future work for multiple-transform encoding.
Chapter 2

Previous Research

In this chapter, we first consider an example where multiple transforms are used to encode a video sequence. We then review two algorithms designed to solve the problem of selecting the transforms and coefficients for each block.

2.1 Multiple-Transform Video Encoding

A significant application of multiple-transform encoding is for encoding video signals. In [12], multiple-transforms are used to encode motion compensation residuals.

We model an image \( x(n_1, n_2) \) as a sample of a stationary random process \( X(n_1, n_2) \). We assume without loss of generality that all stationary random processes here described have zero mean, since we may subtract the mean otherwise. The auto-correlation function of this random process is defined by:

\[
R_X(n_1, n_2) \triangleq E[X(k_1, k_2)X(k_1 + n_1, k_2 + n_2)]
\]

(2.1)

Images and video frames may be well-modeled by a two-dimensional Markov-1 auto-correlation function. This auto-correlation function is given by, for \( \sigma > 0 \) and some \( \rho_1 \) and \( \rho_2 \) between \(-1\) and \(+1\):

\[
R_X(n_1, n_2) = \sigma^2 \rho_1^{[n_1]} \rho_2^{[n_2]}
\]

(2.2)
We can then compute an analytic expression for the KLT to find the best basis for this signal. By using separability, we note this converges to the 2-D DCT as $\rho_1 \to 1$ and $\rho_2 \to 1$ [3]. For most images, we find that $\rho_1 \approx \rho_2 \approx 0.95$. Therefore, the 2-D DCT is very close to their actual KLT.

When considering motion compensation residuals, we find that the values of $\rho_1$ and $\rho_2$ are much lower, since inter-prediction reduces the correlation between pixels. Therefore the 2-D DCT will not necessarily be close to the best transform in every block.

As an example, in the image in figure 2-1a, the energy is distributed over all the pixels, with a high correlation between neighboring pixels. However, in the motion-compensated residuals for the same image, shown in figure 2-1b, the correlation between neighbors is not as high and the energy is focused in directional features, such as the edges of the players. While the 2-D DCT is best used to compress global and non-directional features (when $\rho_1 \to 1$ and $\rho_2 \to 1$), a transform that can take these directional features into account will be better for motion compensation residuals.

![Figure 2-1: Example of frame and motion-compensated residuals for frame.](image)

For motion compensation residuals, a better model for the auto-correlation function can be obtained by taking into account the angle $\theta$ of the edge. In this case, the auto-correlation function for $X(n_1, n_2)$ will be given by:
\[ R_X(n_1, n_2) = \sigma^2 \rho_1 \left| n_1 \cos(\theta) + n_2 \sin(\theta) \right| \rho_2 \left| -n_1 \sin(\theta) + n_2 \cos(\theta) \right| \]

(2.3)

This operation is simply applying a rotation to the Markov-1 model. By choosing \( \theta \) appropriately, the auto-correlation function may be approximated by \( \rho_1 \approx 1 \) and \( \rho_2 \approx 0 \). The KLT will simply be the 1-D DCT in the angle \( \theta \) of the edge.

In [12], a set of transforms is designed for the multiple-transform scheme to take this into account. This set of transforms uses the possible edge directions together with the 2-D DCT. For 4 x 4 blocks of the image, the set of directions used is shown in figure 2-2. Similarly, the set of possible directions used for 8 x 8 blocks of the image is shown in figure 2-3. Using this scheme, there is an improvement in the Bjontegaard-Delta bit-rate metric [13] between 5% and 15% when compared to only using the 2-D DCT.

![Figure 2-2: Possible directions for 4 x 4 directional 1-D DCT.](image)

Using this set of transforms requires sending many bits of side-information. When using all possible directional transforms for every block, side-information accounts for approximately 5% of the bit-rate. To reduce this side-information bit-rate, [14] uses only horizontal and vertical 1-D DCTs together with the 2-D DCT. This is based on the observation that the 2-D DCT, horizontal 1-D DCT, and vertical 1-D DCT are
chosen significantly more often than the other transforms. Applying these transforms to encode H.264 motion compensation residuals had no significant change in the Bjontegaard-Delta bit-rate metric when compared to using all directional transforms.

2.2 Algorithm A: Iterative Method

We now consider the problem given in equation (1.4). We have $N$ blocks of size $M$, and a set of $K$ orthonormal transforms, $T = \{T^{(1)}, ..., T^{(K)}\}$. For each block $x_i$, we
choose one $T_i \in \mathcal{T}$ and form an approximation $\hat{x}_i$ with $\|T_i \hat{x}_i\|_0 = c_i$. We let $c$ denote the vector of the $c_i$, $T$ denote the vector of the $T_i$, and $\hat{x}$ denote the vector of the $\hat{x}_i$. This gives us a constraint $\sum c_i \leq c$. This can then be written as:

$$\min_T \min_{c, ||c||_1 = c} \sum_i \|x_i - \hat{x}_i\|_2^2 = \min_{c, ||c||_1 = c} \min_T \sum_i \|x_i - \hat{x}_i\|_2^2$$ (2.4)

Based on this equality, [11] proposes an iterative algorithm to find $T$ and $c$. This algorithm iterates between choosing the best transform and the appropriate number of coefficients for each block. It initializes with some initial guess $T = T_0$ and then iterates the following two steps until convergence.

Step A1: Fix $T$, find the best choice of $c$. In other words, solve:

$$\min_{c, ||c||_1 = c} \sum_i \|x_i - \hat{x}_i\|_2^2$$ (2.5)

Step A2: Fix $c$, find the best choice of $T$. In other words, solve:

$$\min_T \sum_i \|x_i - \hat{x}_i\|_2^2$$ (2.6)

This iteration may be seen as taking alternating projections between the signals with a fixed number of coefficients in each block and the set of signals with fixed transforms in each block. There is no theoretical guarantee that Algorithm A will converge to a globally optimal solution. In many practical examples the solution will only be locally optimal. This will be studied in more detail in chapter 5.

There are two main advantages of using algorithm A. First, it is fairly efficient, having complexity $O(KNM)$ if we ignore the step to compute the transforms of all blocks, which we will do for all our algorithms. Second, it will give us a solution for a specific fixed value of $c$, even though this solution may not be globally optimal.
2.3 Algorithm B: Rate-Distortion Optimization

A second algorithm is then presented in [11]. Instead of solving the optimization problem we had before, we have now a rate-distortion trade-off, based in [15], which gives us:

$$\min_{T,c} \sum_i ||x_i - \hat{x}_i||_2^2 + \lambda ||c||_1$$  \hspace{1cm} (2.7)

This adds a cost term $\lambda > 0$, which is the cost of adding a coefficient. This $\lambda$ is referred to as marginal cost. Alternatively, one can see this problem as optimizing local rate-distortion metrics. We will define a local metric $\mu_i(\lambda)$ for block $i$ as:

$$\mu_i(\lambda) \triangleq ||x_i - \hat{x}_i||_2^2 + \lambda c_i$$  \hspace{1cm} (2.8)

Then construct a global metric $\mu(\lambda)$ as the sum of the previous metrics:

$$\mu(\lambda) \triangleq \sum_i \mu_i(\lambda)$$  \hspace{1cm} (2.9)

Under this framework, the minimization problem in equation (2.7) becomes:

$$\min_{T,c} \mu(\lambda)$$  \hspace{1cm} (2.10)

Any solution to this problem is a truncation of $\hat{x}$ under a transform $T_i$. This problem is therefore equivalent to optimizing over all truncations $\hat{x}$ of the signal $x$. In this thesis, a truncation is defined as, for every natural numbers $i$ and $n$, we must have $T_i\hat{x}_i(n)$ either equal to $T_ix_i(n)$ or 0. In this case, this problem becomes, for truncations $\hat{x}$:

$$\max_{T,c} \sum_i ||\hat{x}_i||_2^2 - \lambda c_i$$  \hspace{1cm} (2.11)

Define a local metric $\nu_i(\lambda)$ for this alternate problem:

$$\nu_i(\lambda) \triangleq ||\hat{x}_i||_2^2 - \lambda c_i$$  \hspace{1cm} (2.12)
Then we can define a global metric to maximize over all truncations as:

$$
\nu(\lambda) = \sum_i \nu_i(\lambda)
$$

(2.13)

Note if we find a pair \((T, c)\) that solves this optimization problem, then it also solves the previous problem (1.4) for \(c = ||c||_1\). This property is formally stated and proven in the lemma below.

**Lemma 2.3.1** If \((T, c)\) is a pair of transforms and coefficients that optimizes equation (2.7), then it also optimizes equation (1.4) for \(c = ||c||_1\).

**Proof** Suppose this pair \((T, c)\) does not optimize equation (1.4). Let \((\tilde{T}, \tilde{c})\) be the pair that solves equation (1.4) for \(c = ||c||_1\), with associated approximation \(\tilde{x}\). Let \(\hat{x}\) be the best approximation associated with \((T, c)\). Then we note that \(\sum_i ||x_i - \hat{x}_i||_2^2 < \sum_i ||x_i - \tilde{x}_i||_2^2\), and therefore \(\sum_i ||x_i - \tilde{x}_i||_2^2 + \lambda c < \sum_i ||x_i - \hat{x}_i||_2^2 + \lambda c\). But this is a contradiction, since it implies that \(\tilde{x}\) is a better solution for (2.7). Therefore \((T, c)\) must optimize equation (1.4).

Therefore, solving the rate-distortion problem solves the original problem for many values of \(c\).

To solve this problem, one can first define the block-optimal energy function. Take a block of the signal, \(x_i\), and denote by \(E_i(c)\) the block-optimal energy function of block \(i\). This function indicates for each value of the number of coefficients in this block, \(c \in \{0, \ldots, M\}\), the largest energy that we can keep in our truncation using only \(c\) coefficients and the best possible transform.

For example, assume we have a four-sized block of a one-dimensional signal \(x\), with transform coefficients given in magnitude-squared in table 2.1.

If \(c = 1\), either \(E_1(1) = 10\) from the first transform or \(E_1(1) = 15\) from the second transform. Since it is the largest possible value over all transforms, \(E_1(1) = 15\). Repeat this procedure over all possible values of the number of coefficients \(c \in \{0, \ldots, 4\}\), to find that the block-optimal energy function for this block is given in table 2.2.
Table 2.1: Example of transform coefficients in magnitude-squared.

| n  | \( |T^{(1)}x_1(n)|^2 \) | \( |T^{(2)}x_1(n)|^2 \) |
|----|----------------|----------------|
| 1  | 10            | 1.7            |
| 2  | 6             | 2              |
| 3  | 1             | 1.3            |
| 4  | 3             | 15             |

Table 2.2: Example of block optimal energy function for signal in table 2.1.

<table>
<thead>
<tr>
<th>c</th>
<th>( E_i(c) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>

Algorithm B defines for each block the profit function \( f_i(c) \) given by:

\[
f_i(c) = E_i(c) - \lambda c = \nu_i(\lambda)
\]  

(2.14)

This algorithm then solves the problem in equation (2.11) by applying the following two steps.

Step B1: For each block, compute its block-optimal energy function \( E_i(c) \):

\[
E_i(c) = \max_{T_i} ||\hat{x}_i||_2^2
\]  

(2.15)

Step B2: For each block, find \( c_i \) that maximizes the profit function, or:

\[
c_i = \arg \max_{c} f_i(c) = \arg \max_{c} E_i(c) - \lambda c
\]  

(2.16)

The block diagram in figure 2-4 implements this algorithm for one block. In this diagram, \( E^i \) denotes the block-optimal energy function using only transform \( i \).

As an example, suppose we have the single block in the previous example with energy function \( E_1 \) given before and take \( \lambda = 4 \). Then the profit function \( f_1 \) for this block will be \( f_1(c) = E_1(c) - 4c \), and is given in table 2.3. The best choice is to keep one coefficient, and have \( c_1 = 1 \). This best coefficient changes with \( \lambda \). Let
us consider the same block. If $\lambda$ is very large, such as $\lambda = 20$, the best choice is to keep no coefficient. If $\lambda$ is very small, such as $\lambda = 0.5$, the best choice is to keep all coefficients.

This algorithm is globally optimal. A proof of optimality for this algorithm is presented in [11], based in convex optimization. Note this algorithm does not solve the original problem for a specific value of $c$. If we are looking for a specific value of $c$, then we need to search over different values of $\lambda$. There is also no simple association between $\lambda$ and values of $c$. Furthermore, some values of $c$ may not be associated with any value of $\lambda$, as we will see later.

In terms of complexity, computing the block-optimal energy function requires sorting the transform coefficients in magnitude-squared for each transform. Therefore step B1 will take $O(KNM \log(M))$ operations. This step is followed by simply computing $f_i(c)$ for all values of $c$ in each block, which takes $O(NM)$ operation. One iteration of this algorithm will thus take $O(KNM \log(M))$ operations. If we need to search for a value of $\lambda$, additional iterations will only require $O(NM)$ operations, since we only need to repeat step B2.

Finally, we note this algorithm is implemented independently for all blocks, therefore parallel implementations can be used to significantly increase the performance.

Table 2.3: Example of profit function for signal in table 2.1 when $\lambda = 4$. 

<table>
<thead>
<tr>
<th>$c$</th>
<th>$f_1(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
This will reduce the overall complexity by a factor of $N$.

2.4 Motivation of Thesis

Considering the setup from previous research, this thesis first considers two new algorithms to choose the best transforms. The first algorithm is a globally optimal algorithm which is an alternate version of algorithm B. The second algorithm is a globally optimal algorithm that finds a solution for a specific value of $c$. Until now, this framework did not take quantization into account. This thesis will study how we can change these algorithms if we follow truncation by a quantization step.
Chapter 3

Algorithm C: Thresholding

Algorithm C considers the same rate-distortion problem as algorithm B, from equation (2.7). In algorithm B, the solution for a fixed block always consists of keeping the coefficients greater than \( \lambda \) in magnitude-squared for some transform. Therefore, a simple solution to the rate-distortion problem would be given by the following steps:

Step C1: For each block \( i \) and each possible transform, threshold the coefficients of the transformed block in magnitude-squared by \( \lambda \), and compute the metric \( \mu_i(\lambda) \).

Step C2: For each block, choose the transform with the smallest metric \( \mu_i(\lambda) \).

This algorithm is implemented as a block diagram in figure 3-1, where \( \mu^{(i)} \) is the metric of the current block when transform \( T^{(i)} \) is used.

As an example, let us consider the previous single block \( x_1 \) with two transforms given in table 2.1. Then we note for \( \lambda = 4 \) the solution would be given by keeping the coefficients which have magnitude squared greater than 4. For the first transform, we have a total energy of 16, while for transform 2 we have a total energy of 15. In
this case, this means the metric \( \mu_i(\lambda) \) will be \( 16 - 8 = 8 \) for the first transform and 
\( 15 - 4 = 11 \) for the second transform. Therefore, the solution will be to keep only one coefficient from the second transform. This algorithm solves the same problem as algorithm B, and will result in the same solution for all values of \( \lambda \). This algorithm's output also varies significantly with \( \lambda \). As a consistency check, we can see that for extremely small values of \( \lambda \) we take all non-zero coefficients, so we keep all coefficients, as expected. Meanwhile, for extremely large values of \( \lambda \), no coefficient will be kept for any transform, as expected.

3.1 Optimality

We will now show that this algorithm results in a globally optimal solution for equation (2.7). From lemma 2.3.1, this solution is enough to solve the original problem for many possible values of \( \lambda \).

We begin our proof with a simple lemma that argues that for a fixed block and fixed transform, the best solution is always given by keeping the coefficients larger than \( \lambda \) in magnitude-squared. This proof is based on the fact that adding a coefficient should be done as long as the reduction in MSE is greater than the value of the marginal cost \( \lambda \).

**Lemma 3.1.1** Fix a block \( i \) and an orthonormal transform \( T_i \). Let \( \hat{x}_i \) be a truncation of our signal \( x_i \). Then \( \hat{x}_i \) is a solution for the optimization problem below:

\[
\min_{c_i} ||x_i - \hat{x}_i||_2^2 + \lambda c_i = \min_{c_i} \mu_i(\lambda) \quad (3.1)
\]

with only \( c_i \) coefficients under \( T_i \) if and only if the following two conditions are true:

(a) If \( |T_i x_i(n)|^2 > \lambda \), then \( T_i \hat{x}_i(n) = T_i x_i(n) \).

(b) If \( |T_i x_i(n)|^2 < \lambda \), then \( T_i \hat{x}_i(n) = 0 \).

**Proof** Before our proof, we note it suffices to take \( \hat{x}_i \) to be a truncation of \( x_i \) under the transform \( T_i \), because any non-truncation will be sub-optimal.
We first show that optimality under equation (3.1) will imply that the two conditions are true. We begin by showing optimality implies condition (a) by contradiction. Let $S_i$ be the set of coefficients used in block $i$ for the optimal solution, with cardinality $c_i = |S_i|$ and associated approximation $\tilde{x}_i$. Assume there exists $n$ such that $|T_i x_i(n)|^2 > \lambda$ and $n \notin S_i$.

Let $\tilde{S}_i = S_i \cup \{n\}$; in other words, add $n$ to the set of chosen coefficients in our approximation. Denote this new approximation by $\tilde{x}_i$. Then the new number of coefficients $\tilde{c}_i$ will be $c_i + 1$. Similarly, from the energy equation, we find:

$$||x_i - \tilde{x}_i||_2^2 = ||T_i x_i - T_i \tilde{x}_i||_2^2 = \sum_{m \notin \tilde{S}_i} |T_i x_i(m) - T_i \tilde{x}_i(m)|^2$$  \hspace{1cm} (3.2)

Writing this in function of $S_i$ instead:

$$||x_i - \tilde{x}_i||_2^2 = \sum_{m \notin S_i} |T_i x_i(m) - T_i \tilde{x}_i(m)|^2 - |T_i x_i(n)|^2$$  \hspace{1cm} (3.3)

From the energy equation:

$$||x_i - \tilde{x}_i||_2^2 = ||T_i x_i - T_i \tilde{x}_i||_2^2 - |T_i x_i(n)|^2 = ||x_i - \tilde{x}_i||_2^2 - |T_i x_i(n)|^2$$  \hspace{1cm} (3.4)

Therefore, we find the new metric $\tilde{\mu}_i(\lambda)$ will be:

$$\tilde{\mu}_i(\lambda) = ||x_i - \tilde{x}_i||_2^2 + \lambda \tilde{c}_i = ||x_i - \tilde{x}_i||_2^2 - |T_i x_i(n)|^2 + \lambda (c_i + 1)$$  \hspace{1cm} (3.5)

Using the fact that $|T_i x_i(n)|^2 > \lambda$ together with the definition of $\mu_i(\lambda)$, we then note:

$$\tilde{\mu}_i(\lambda) = \mu_i(\lambda) - |T_i x_i(n)|^2 + \lambda < \mu_i(\lambda)$$  \hspace{1cm} (3.6)

But this means the approximation $\tilde{x}$ is not optimal, which is a contradiction. So condition (a) must be implied by any solution to equation (3.1). We prove condition (b) using a very similar argument. We now assume there exists $n$ such that $n \in S_i$ but $|T_i x_i(n)|^2 < \lambda$, and generate the approximation for $\tilde{S}_i = S_i \setminus \{n\}$ Then we note
that the number of coefficients $c_i$ will be $c_i - 1$ and the mean-square error will increase by $|T_i x_i(n)|^2$. Therefore our new metric $\tilde{\mu}_i(\lambda)$ will be:

$$\tilde{\mu}_i(\lambda) = ||x_i - \tilde{x}_i||^2 + |T_i x_i(n)|^2 + \lambda(c_i - 1)$$  \hspace{1cm} (3.7)

This in turn gives us, using the fact that $|T_i x_i(n)|^2 < \lambda$:

$$\tilde{\mu}_i(\lambda) = \mu_i(\lambda) + |T_i x_i(n)|^2 - \lambda < \mu_i(\lambda)$$  \hspace{1cm} (3.8)

This is a contradiction to the optimality of $\tilde{x}_i$, and we conclude condition (b) must be true. Now, to see that these two conditions suffice for a solution to be optimal, we note that an optimal solution exists, since there are a finite number of possibilities, and so it must satisfy both of the previous conditions. Now all that is left is to add or remove coefficients with magnitude-squared $|T_i x_i(n)|^2 = \lambda$. But this will not change the metric $\mu_i(\lambda)$, from the previous argument. Therefore any solution satisfying these two conditions will solve equation (3.1). 

This lemma applies only for one transform and one block, but we can easily extend this result for many different blocks. This justifies the optimality of our algorithm, and is shown in the following theorem.

**Theorem 3.1.2** Any approximation $\tilde{x}$ given by algorithm C solves equation (2.7). Furthermore, any solution to equation (2.7) can be obtained by this algorithm, by choosing if coefficients with value equal to $\lambda$ are kept or not.

**Proof** We note that we can separate our optimization problem into independent problems, as in:

$$\min_{T, C} \sum_i ||x_i - \tilde{x}_i||^2 + \lambda c_i = \sum_i \min_{T, c_i} ||x_i - \tilde{x}_i||^2 + \lambda c_i$$  \hspace{1cm} (3.9)

As long as we can solve the problem for each block, we can find the globally optimal solution. Furthermore, we note that:

34
\[
\min_{\mathbf{T}_i, \mathbf{c}_i} ||x_i - \hat{x}_i||^2_2 + \lambda c_i = \min_{\mathbf{c}_i} \left( \min_{\mathbf{T}_i} ||x_i - \hat{x}_i||^2_2 + \lambda c_i \right)
\]  
(3.10)

From lemma 3.1.1, the solution for a particular block and fixed transform can be obtained by thresholding by \(\lambda\) and choosing whether or not to keep the coefficients that have magnitude-squared equal to \(\lambda\). And so, a solution to the general problem will always be given by taking the transform with the smallest metric \(\mu_i(\lambda)\) after thresholding the coefficients by \(\lambda\) in each block, and choosing whether or not to keep coefficients that have magnitude-squared equal to \(\lambda\).

An important comment about this algorithm is that it solves the same problem as algorithm B, so we cannot expect it to solve the general problem in equation (1.4). On the other hand, as shown in the previous theorem, it will generate every optimal solution to equation (2.7).

Based on this theorem, we present one property that may be useful in the future research, as well as when searching for optimal values of \(\lambda\). This property says that any optimal solution for the previous problem, must have \(c_i\) that is non-decreasing with decreasing values of \(\lambda\) in every block \(i\). The proof to this lies on the fact that the metric \(\mu_i(\lambda)\) always increases with \(\lambda\) and its derivative with respect to \(\lambda\) is given by the \(c_i\) of the current transform. Therefore, when we are changing transforms, the new transform must have a higher derivative, and we can conclude it will always result in increasing the number of coefficients.

**Corollary 3.1.3** Let \(c_i(\lambda_0)\) be the number of coefficients used in block \(i\) to solve (2.7) for \(\lambda = \lambda_0\). Then we note \(c_i(\lambda)\) is non-decreasing with decreasing values of \(\lambda\). Furthermore, \(c(\lambda) = \sum_i c_i(\lambda)\) also is non-decreasing with decreasing values of \(\lambda\).

### 3.2 Complexity

In terms of complexity, this algorithm first needs to compute all transforms for all blocks. We do not consider this step in our calculations, since it is present in all the algorithms examined. The next step is thresholding, which is done for all coefficients
in all transforms, in all blocks, and therefore has a total of $O(KMN)$ operations. Following this step we compute the metric $\mu_i(\lambda)$ for each transform, which also takes $O(KMN)$ operations. Finally we need to find the maximum $\mu_i(\lambda)$ for each block. This requires $O(KN)$ operations. Therefore, this algorithm’s complexity is overall $O(KMN)$.

Note this algorithm must be repeated as a whole if we want to examine different values of $\lambda$. Therefore any subsequent computations will also take $O(KMN)$. Comparing to algorithm B, we note that as long as we need to compute this for a small number of values of $\lambda$, algorithm C will be better. If we need to examine many different values of $\lambda$, however, algorithm B will be better.

This algorithm is parallelizable, since we may execute it separately and independently for every block. This gives us a reduction in complexity by a factor of $N$ when implemented in parallel.
Chapter 4

Algorithm D: Dynamic Programming

The previous algorithm solves the problem for every possible value of \( \lambda \), but a value of \( \lambda \) is not associated with each value of \( c \). We consider solving the problem in equation (1.4). We examine the equivalent problem of finding the best truncation and transform combination to maximize the energy of this truncation given a total number of coefficients \( c \). To solve this problem, let \( T^{(m)} \) denote the first \( m \) elements of \( T \). Note that we can write our optimization problem as:

\[
\max_{T_M,C_M} \left( \max_{T^{(M-1)}, \|c^{(M-1)}\|_1 = c - c_M} \|\hat{x}\|^2 \right) \tag{4.1}
\]

This gives us a recursive method to solve this problem. Let the energy of the best possible coefficient allocation for the first \( m \) blocks with \( c \) coefficients be denoted by \( S_m(c) \), and let the block optimal energy function of block \( j \) be denoted by \( E_j(c) \). Then we can write:

\[
S_m(c) = \max_{k \in \{0,1,\ldots,M\}} S_{m-1}(c-k) + E_m(k) \tag{4.2}
\]

This recursive implementation would not be polynomial, so instead we take a bottom-up approach. This approach is implemented starting at block \( m = 1 \) with \( S_0(0) = 0 \), and by following the steps below:
Step D1: First compute $E_m(c)$ for $c \in \{0, ..., M\}$.

Step D2: Update $S_m(c)$ for every $c \in \{0, ..., mM\}$ according to equation:

$$S_m(c) = \max_{k \in \{0,1,\ldots,M\}} S_{m-1}(c - k) + E_m(k) \quad (4.3)$$

Step D3: If $m < N$, increment $m$ by one and go back to step D1.

We now consider two examples. First we take a signal with a single block and transforms $T^{(1)}$ and $T^{(2)}$. Their coefficients are given in magnitude-squared in table 4.1.

| $c$ | $|T^{(1)}x_1(n)|^2$ | $|T^{(2)}x_1(n)|^2$ |
|-----|-----------------|-----------------|
| 1   | 11              | 6               |
| 2   | 2               | 6               |
| 3   | 2               | 4               |
| 4   | 2               | 1               |

Table 4.1: Example of transform coefficients in magnitude-squared.

In this case, we can compute the block-optimal energy function, and find the result in table 4.2. This block optimal energy function is not concave. For concave block-optimal energy functions, it is shown in [11] that algorithm B will give the optimal solution for every $c$.

<table>
<thead>
<tr>
<th>$c$</th>
<th>$E_1(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 4.2: Example of block optimal energy function for signal in table 4.1.

By running algorithm D, we find the trivial solution that $S_1(c) = E_1(c)$. For one block, finding the optimal coefficient allocation is the same as just finding the block-optimal energy function, by definition of block optimal energy function.

Now we examine the outputs of algorithm C as we vary the value of $\lambda$. For $\lambda > 11$, the solution is to keep no coefficient. Second, as long as $6 < \lambda < 11$, the best solution
is to keep one coefficient from transform \( T^{(1)} \). Third, as long as \( 4 < \lambda < 6 \), we note 
\( T^{(2)} \) will keep two coefficients and have metric \( \mu(\lambda) = 5 + 2\lambda \) and \( T^{(1)} \) will have metric 
\( \mu(\lambda) = 6 + \lambda \), so it will be best to choose \( T^{(1)} \). Fourth, as long as \( 2 < \lambda < 4 \), then 
the metric for transform \( T^{(1)} \) will be \( \mu(\lambda) = 6 + \lambda \) and for transform \( T^{(2)} \) will be 
\( \mu(\lambda) = 1 + 3\lambda \). Therefore, as long as \( \lambda > 2.5 \), we will choose \( T^{(1)} \), but otherwise 
we will choose \( T^{(2)} \). We note this jump will necessarily be from 1 coefficient to 3 
coefficients as we move lambda, and no solution here will use 2 coefficients. In fact, 
for \( 1 < \lambda < 2.5 \) we will choose \( T^{(2)} \) and keep 3 coefficients. Finally, for \( \lambda < 1 \), we can 
choose any transform since we are keeping all the coefficients. This is summarized in 
table 4.3. Note algorithm C will never give us a solution with 2 coefficients, while 
algorithm D will indeed find this solution. We can see that algorithm B will never 
choose this solution, since its solution is a subset of those produced by algorithm C.

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>( c ) for ( T^{(1)} )</th>
<th>( c ) for ( T^{(2)} )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda &gt; 11 )</td>
<td>0</td>
<td>0</td>
<td>Any</td>
</tr>
<tr>
<td>( 6 &lt; \lambda &lt; 11 )</td>
<td>1</td>
<td>0</td>
<td>( T^{(1)} )</td>
</tr>
<tr>
<td>( 4 &lt; \lambda &lt; 6 )</td>
<td>1</td>
<td>2</td>
<td>( T^{(1)} )</td>
</tr>
<tr>
<td>( 2.5 &lt; \lambda &lt; 4 )</td>
<td>1</td>
<td>3</td>
<td>( T^{(1)} )</td>
</tr>
<tr>
<td>( 2 &lt; \lambda &lt; 2.5 )</td>
<td>1</td>
<td>3</td>
<td>( T^{(2)} )</td>
</tr>
<tr>
<td>( 1 &lt; \lambda &lt; 2 )</td>
<td>4</td>
<td>3</td>
<td>( T^{(2)} )</td>
</tr>
<tr>
<td>( 0 &lt; \lambda &lt; 1 )</td>
<td>4</td>
<td>4</td>
<td>Any</td>
</tr>
</tbody>
</table>

Table 4.3: Example of algorithm C for signal in table 4.1.

We now consider another example with two blocks. Suppose we have a second 
block with transforms given in table 4.4.

| \( c \) | \( \left| T^{(1)}x_2(n) \right|^2 \) | \( \left| T^{(2)}x_2(n) \right|^2 \) |
|-------|-----------------|-----------------|
| 1 | 2.4 | 2 |
| 2 | 1.5 | 2 |
| 3 | 1.9 | 2 |
| 4 | 2.2 | 2 |

Table 4.4: Example of transform coefficients for second block in magnitude-squared.

Our objective now is to determine the best allocation in each block using algorithm 
D. We first find the block optimal energy function for the first block, as given in 4.2.
This gives us $S_1(c)$ as we had before. We now find the block optimal energy function for the second block, shown in table 4.5.

<table>
<thead>
<tr>
<th>$c$</th>
<th>$E_2(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2.4</td>
</tr>
<tr>
<td>2</td>
<td>4.6</td>
</tr>
<tr>
<td>3</td>
<td>6.5</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4.5: Example of block optimal energy function for signal in table 4.4.

In this case, we consider each possible value of $c$, and use equation (4.2) to find $S_2(c)$. For example, for $c = 1$, we compare $E_2(1) = 2.4$ with $S_1(1) = 11$, so we take one coefficient from block 1 and no coefficient from block 2. Similarly, for $c = 2$, our options are $E_2(2) = 4.6$, $E_2(1) + S_1(1) = 13.4$ or $S_1(2) = 13$, so we take one coefficient from each block. Repeating this procedure will give us the allocation in table 4.6.

<table>
<thead>
<tr>
<th>$c$</th>
<th>$c_1$</th>
<th>$T_1$</th>
<th>$c_2$</th>
<th>$T_2$</th>
<th>$S_2(c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Any</td>
<td>0</td>
<td>Any</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$T^{(1)}$</td>
<td>0</td>
<td>Any</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$T^{(1)}$</td>
<td>1</td>
<td>$T^{(1)}$</td>
<td>13.4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>$T^{(2)}$</td>
<td>0</td>
<td>Any</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>$T^{(2)}$</td>
<td>1</td>
<td>$T^{(1)}$</td>
<td>18.4</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>$T^{(2)}$</td>
<td>2</td>
<td>$T^{(1)}$</td>
<td>20.6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>$T^{(2)}$</td>
<td>3</td>
<td>$T^{(1)}$</td>
<td>22.5</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>$T^{(2)}$</td>
<td>4</td>
<td>Any</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>Any</td>
<td>4</td>
<td>Any</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 4.6: Example of algorithm D for two blocks.

Not all values of $c$ will have an optimal solution given by the algorithm C, since we can see that $c_1$ may decrease as we increase $c$, but both of these are strictly non-decreasing with decreasing $\lambda$. An important point to comment when comparing these algorithms is that algorithm D produces the solution for all values of $c$. This is a property of algorithms implemented by using dynamic programming [16].
4.1 Optimality

Our objective now is to prove that algorithm D gives us a globally optimal solution. The proof is given in the following theorem.

**Theorem 4.1.1** The function $S_N(c)$ produced by algorithm D for a signal with $N$ blocks of size $M$ gives us the optimal energy to be kept in each block. In other words, for all possible values of $c \in \{0, ..., MN\}$ and truncations $\tilde{x}$, we have:

$$S_N(c) = \max_{T, ||c||_1 = c} ||\tilde{x}||_2^2$$

(4.4)

**Proof** First we show that some approximation $\tilde{x}(n)$ exists and achieves such $S_N(n)$ by constructing this approximation. We know that this approximation is easily found by knowing the number of coefficients $c_i$ in each block $i$. Define each $c_i$ as:

$$c_i = \arg \max_{k \in \{0, ..., M\}} S_{i-1}(c - \sum_{j=i+1}^{N} c_j - k) + E_i(k)$$

(4.5)

We let $\tilde{x}$ be the approximation given by taking the most energy when keeping $c_i$ coefficients in each block $i$. Then we note that:

$$\sum_{i} \sum_{n} |\tilde{x}_i(n)|^2 = \sum_{i} E_i(c_i) = S_N \left( \sum_{i} (c_i) \right) = S_N(c)$$

(4.6)

We now need to prove that for any truncation $\tilde{x}$ the following inequality holds:

$$S_N(c) \geq ||\tilde{x}||_2^2 = \sum_{i=1}^{N} ||\tilde{x}_i||_2^2$$

(4.7)

We will show this by induction over the number of blocks $N$.

Inductive basis: First, assume $N = 1$. Then we have for any $c \in \{0, \ldots, M\}$, as $E_i(c)$ is increasing in $c$:

$$S_1(c) = \max_{k \in \{0, \ldots, M\}} S_0(c - k) + E_1(k) = E_1(c)$$

(4.8)

But, by definition of $E_1(c)$, it is the greatest energy we can have for block 1 if we
keep $c$ coefficients in our set of transforms. So, for any other transform in the set and keeping $c$ coefficients, we must have:

$$S_1(c) = E_1(c) \geq ||x_1(n)||_2^2$$  (4.9)

Inductive hypothesis: If we have, for all $c \in \{0, \ldots, MN\}$ and for all truncations $\hat{x}$ using $c$ coefficients in the first $N$ blocks:

$$S_N(c) \geq \sum_{i=1}^{N} ||\hat{x}_i||_2^2$$  (4.10)

Then we wish to show that for all $c \in \{0, \ldots, M(N + 1)\}$ and for all truncations $\hat{x}$ using $c$ coefficients in the first $N + 1$ blocks:

$$S_{N+1}(c) \geq \sum_{i=1}^{N+1} ||\hat{x}_i||_2^2$$  (4.11)

We begin by noting that, for any truncation $\hat{x}$ of the first $N + 1$ blocks:

$$\sum_{i=1}^{N+1} ||\hat{x}_i||_2^2 = \sum_{i=1}^{N} ||\hat{x}_i||_2^2 + ||\hat{x}_{N+1}||_2^2$$  (4.12)

From our inductive hypothesis we note that $\hat{x}_1(n)$ upto $\hat{x}_N(n)$ are using some number of coefficients $\bar{c} \leq c$ to encode the coefficients, so we can write:

$$\sum_{i=1}^{N+1} ||\hat{x}_i||_2^2 \leq S_N(\bar{c}) + ||\hat{x}_{N+1}||_2^2$$  (4.13)

Define $k = c - \bar{c}$, where $0 \leq k \leq N$. Furthermore, note that we need to use $k$ coefficients to encode the last term of our previous sum. So, by definition of $E_{N+1}(k)$, we have:

$$\sum_{i=1}^{N+1} ||\hat{x}_i||_2^2 \leq S_N(\bar{c}) + E_{N+1}(k) \leq \max_k S_N(c - k) + E_{N+1}(k)$$  (4.14)

But, as we defined for algorithm D, we have:
\[ \sum_{i=1}^{N+1} \| \tilde{x}_i \|_2^2 \leq S_{N+1}(c) \]  

(4.15)

And we thus conclude that our inequality holds.

We note this algorithm can be used for other metrics. Let \( \| x \| \) be any metric we wish to compute that we can write it as a sum of individual block metrics, or:

\[ \| x \| = \sum_i \| x_i \| \]  

(4.16)

Assume that we can compute the block-optimal metric function \( E_i(c) \) given by maximizing \( \| x_i \| \) in each block \( i \) using only \( c \) coefficients. Then algorithm D will give us the optimal coefficient allocation for that metric. The issue of using different metrics is then related to the complexity of computing these \( E_i(c) \).

### 4.2 Complexity

The complexity of algorithm D comes from two sources. First we have the operations to compute the block-optimal energy function. In this case, the complexity is that of computing the transforms and sorting its values. We ignore the complexity to compute the transforms since all of the algorithms require it. This will result in a complexity of \( O(KNM \log(M)) \) for this first step. Second, we observe all values of \( c \) for each block, and so we have a resulting complexity of \( O(M^2N) \) operations for each block. By adding over all blocks, we obtain a total of \( O(M^2N^2) \) operations. Note that \( c \leq MN \) and \( K \ll N \), therefore we say this algorithm’s complexity is \( O(M^2N^2) \).

This algorithm solves a more general problem than the three previous algorithms, and always gives us a globally optimal solution for all possible values of \( c \). If we wish to find all possible combinations of \( c \) with the optimal MSE, this algorithm only requires one iteration, and will take \( O(MN) \) operations per output. This may be especially useful if we need to transmit the video sequence to many users at many different quality rates.
4.3 Multiple-choice Knapsack Problem

Having described this problem as two steps, we can now understand the inherent structure in which this is based. Note the second step’s objective is to solve a simple optimization problem, given by:

$$\max_{\|e\|=c} \sum_{i} E_i(c_i)$$  \hfill (4.17)

This structure is similar to the multiple-choice knapsack problem. We will here see two different formulations of the knapsack problem and how they relate to our problem.

4.3.1 The 0-1 Knapsack Problem

The knapsack problem has been studied thoroughly in the operational research literature\textsuperscript{17, 18}. The traditional knapsack problem is known as 0-1 knapsack problem.

We begin with a set of objects \( \{X_1, ..., X_N\} \). For each object, we associate a profit \( p_i \) and a weight \( w_i \). Assume all weights \( w_i \) are integer values. For each object, we can choose to either add it to the knapsack or not. The knapsack has a profit given by the sum of the profits of the objects added, and a weight given by the sum of the weights of the objects added. The knapsack problem consists of maximizing the total profit of the knapsack, while ensuring the total weight is smaller than a maximum weight \( W \).

This problem is mathematically stated by introducing indexes \( x_i \) which indicate whether an object is in the knapsack or not. We then write it as:

$$\max_{x} \sum_{i=1}^{N} x_ip_i$$

subject to \( \sum_{i} w_ix_i \leq W, x_i \in \{0, 1\} \)

Dynamic programming is used to solve the 0-1 knapsack problem\textsuperscript{18}. Intuitively, this solution comes from noticing that solving for \( N \) objects means we either take
the $N$-th object and find the best combination for the previous $N - 1$ blocks with a weight of $W - w_N$, or we do not take it and use the best solution for the previous $N - 1$ blocks with a weight of $W$. We start by defining the partial profit function, $P_i(w)$, for $i$ from $0$ to $N$. We let $P_0(w) = 0$ for all $w \geq 0$ and $P_0(w) = -\infty$ for all $w < 0$. Our objective is then to find $P_N(W)$, by using the following update rule:

$$P_N(w) = \max\{P_{N-1}(w), P_{N-1}(w - w_N) + p_N\} \quad (4.18)$$

All we need to do in this case is to take each element $X_i$, and compute $P_i(w)$ for all values of $w \leq W$. In this case, our complexity is given by $O(NW)$. If $W$ is fixed, then it is polynomial in $N$, but, in many cases, $W$ may increase exponentially with $N$. This algorithm is then classified as pseudo-polynomial.

Note the coefficient allocation problem with $M = 1$ is an instance of the 0-1 knapsack problem. In this case, we would have $w_i = 1$ and $p_i = E_i(1)$.

### 4.3.2 The Multiple-choice Knapsack Problem

We now consider another alternate form of the knapsack problem, known as the multiple-choice knapsack problem. In this variant, we consider instead that we have many classes of objects $C_i$, each with size $M_i$. For each object $X_{i,j} \in C_i$, we have an object with associated weight $w_{i,j}$ and profit $p_{i,j}$. For each class, we add one and only one of the objects to the knapsack. We again maximize the total profit under a constraint on the total weight.

The multiple-choice knapsack problem has been studied in the signal processing literature. It has been used to characterize the bit-allocation problem [19]. Many approximations to its solution have also been derived and implemented in the context of image coding and quantizer selection [20, 21], as well as in video encoding [22, 23].

To represent this problem mathematically, let $x_{i,j}$ be the indicator of whether object $X_{i,j}$ is in the knapsack or not. In this case, we need the condition that $\sum_j x_{i,j} = 1$, so that we can guarantee only one element from each class is taken. In this case, we have:
\[
\begin{align*}
\text{maximize} \quad & \sum_i \sum_j x_{i,j} p_{i,j} \\
\text{subject to} \quad & \sum_i \sum_j w_{i,j} x_{i,j} \leq W, \quad \sum_j x_{i,j} = 1, \quad x_{i,j} \in \{0, 1\}
\end{align*}
\]

This problem is a more general version of the 0-1 knapsack problem. The two are equal when we have only two elements in each class, and these elements correspond to either keeping or not the original object.

This problem may be solved with dynamic programming, similarly to (4.18). We here use the same principle as the previous solution, but instead take into account that all possible elements of a class may be added to the knapsack. Let \( P_N(w) \) be the same partial profit function. We then have the recursion:

\[
P_N(w) = \max_j P_{N-1}(w - w_{N,j}) + p_{N,j}
\]  \hspace{1cm} (4.19)

We implement this algorithm by taking each class \( C_i \) at a time and obtaining \( P_i(w) \) for all values of \( w \leq W \). If all of our classes have the same size \( M \), the complexity will be \( O(NMW) \). This algorithm is pseudo-polynomial, since it will be polynomial as long as \( W \) does not grow exponentially with \( N \).

Alternatively, we can see that this problem may be solved by considering instead the Lagrangian relaxation given by, for some \( \lambda > 0 \):

\[
\begin{align*}
\text{maximize} \quad & \sum_i \sum_j x_{i,j} p_{i,j} - \lambda \sum_i \sum_j x_{i,j} w_{i,j} \\
\text{subject to} \quad & \sum_j x_{i,j} = 1, \quad x_{i,j} \in \{0, 1\}
\end{align*}
\]

In this case, we note this problem can then be broken into the same individual problem in multiple blocks. But this means in each block we take the solution that optimizes the Lagrangian metric \( p_{i,j} - \lambda w_{i,j} \).

In this sense, the coefficient allocation problem may be seen as an instance of the multiple-choice knapsack problem. Our classes consist of each of the blocks. The elements in each class are the numbers of coefficients we choose to keep in each block,
with value from 0 to \( M \). Our cost will then be the same number of coefficients, and the profits are the energies kept with each number of coefficients. In this approach, we take \( j \in \{0, \ldots, M\} \). Their weights are given by \( w_{i,j} = j \), and profits \( p_{i,j} = E_i(j) \).

Algorithm D is the same as the dynamic programming solution to the multiple-choice knapsack problem. If \( W \leq MN \), then the complexity will become \( O(N^2M^2) \). In this application, the number of coefficients kept will be polynomial with \( N \). Similarly, algorithm B is the same as the Lagrangian relaxation to the multiple-choice knapsack problem, where we take the number of coefficients that maximizes the Lagrangian metric.
Chapter 5

Algorithm Comparison

In this chapter, we compare the four different algorithms discussed in chapters 2, 3, and 4. We evaluate the performance of these algorithms through simulations. In section 5.1, we implement these algorithms and present the experimental setup used. In section 5.2, we compare these algorithms in terms of the reconstruction MSE as a function of the number of coefficients kept for video compression. We then summarize this comparison in section 5.3.

5.1 Experimental Setup

We now apply our system to video encoding, using the system presented in [14]. We use only $K = 3$ transforms: 2-D DCT, 1-D Horizontal DCT, and 1-D Vertical DCT. We take the first frame out of motion compensated residuals. We also test our methods in terms of encoding directly from the images. All video sequences we use are grayscale and in CIF format, with a size of $352 \times 288$. In each case we analyze the energy preserved as a function of the number of coefficients kept. Our objective is to compare the four algorithms in the case of $4 \times 4$, $8 \times 8$, and $16 \times 16$ block sizes. We run algorithm A until convergence to capture the best performance, initializing all transforms as 2-D DCT. The video test sequences considered are shown in figure 5-1, and their motion compensated residuals are shown in figure 5-2. Different sequences are used to capture many possible motion characteristics; the bridge sequence does
not have as much motion as the rest, while the football sequence has a lot of motion.

\section{5.2 \hspace{1ex} Video Compression}

We begin by comparing the algorithms when encoding motion compensated residuals by using \(8 \times 8\) blocks. We use these sequences to perform a detailed comparison between algorithms A and D. We then consider using other block sizes. Finally, we evaluate what happens as we encode the signals directly instead of motion-compensated residuals.
Figure 5-2: First frame of motion compensated residuals to be encoded.

5.2.1 Compression of Motion Compensated Residuals

In our first approach, we consider $8 \times 8$ block size residuals. In figure 5-3, the energy preserved is shown as a function of the number of coefficients for each of the sequences.

Many things can be quickly seen from these plots. First, we note that the values of the four algorithms are approximately the same. As expected from our theoretical results, at any point where algorithm B or C has a solution, we note algorithm D will have the same solution. To see this in greater detail, in figure 5-4, we use markers to indicate the points given by algorithm B, and zoom into the curve in figure 5-3c. As we can see, the points for algorithm B are always positioned on the curve given by algorithm D. However, there may be differences among the curves, mostly because of
the fact that we look at a few quantized values of $\lambda$, and at some points we cannot find a value of $\lambda$ to be associated with a specific number of coefficients. This is the major advantage of using algorithm D instead of algorithms B and C.

Observing the different motion-compensated residuals in figure 5-2, we note that the ones with more correlation are the ones that can be compressed the most. Consider the football and carphone sequences. We note these two residuals are very structured. Therefore, choosing appropriate transforms leads to a greater potential for compression. This is exploited by our algorithm, and therefore our curves achieve a higher fraction of the energy with fewer coefficients. For example, we can achieve more than 90% of the energy of the football motion compensated residuals with only
Figure 5-4: Zoom for comparison between algorithms B and D when encoding the football motion compensation residuals from figure 5-3c. Points for algorithm B are on the curve for algorithm D, but not all points were found for algorithm B.

10% of the coefficients. Meanwhile, observing the bridge (close) sequence, we note its motion compensated residuals have a very noisy profile, and therefore we need many more coefficients to achieve high energy levels. In this case, we need more than a quarter of the coefficients to achieve 90% of the energy.

Another important point is that, from these figures, it may appear to be the case that algorithm A is always the same as algorithm D, but that is not the case. Previous research [11] compared algorithms A and B, and noted that algorithm A usually achieves very close performance to algorithm B when we have such a small number of transforms. This comparison was not ideal, since one could not find the exact best representation for every fixed number of coefficients. With algorithm D, we can now make a more fair comparison to algorithm A.

We observe the curves given by algorithms A and D in more detail in figure 5-5,
Figure 5-5: Comparison between algorithms A and D when encoding the football motion compensation residuals and using a small number of coefficients.

for the football motion compensation residuals, where we look at small values of the number of coefficients. As we can see, there is a very small difference between these algorithms. Computing the Bjontegaard-Delta coefficient savings [13], we find that using algorithm D saves approximately 2% of the coefficients used in algorithm A.

### 5.2.2 Block-size Comparison

We now compare the different block-sizes in terms of their performance when encoding the motion compensated residuals. Figures 5-6 and 5-7 show the total preserved energy as a function of the fraction of coefficients kept for $4 \times 4$ and $16 \times 16$ blocks respectively.

First, as we saw before, there is no significant difference among the four algorithms. The $4 \times 4$ case has the most discernible difference among the algorithms, especially for
the mobile video sequence. This comes from the fact that we now need to get a total of \( N = \frac{352 \times 288}{4 \times 4} = 6336 \) transforms correctly, while in \( 8 \times 8 \) and \( 16 \times 16 \) cases we had a quarter or a sixteenth of that number of transforms respectively. Therefore, there is more room for a locally suboptimal solution of algorithm A, as well as missing values of \( \lambda \) for algorithms B and C.

Second, for signals with less correlation, a smaller block-size is better. For the bridge (close) sequence, a smaller block-size will be better than \( 16 \times 16 \). Meanwhile, for sequences with more correlation, such as carphone video sequence, we can see a larger block-size will be better than \( 4 \times 4 \).
Figure 5-7: Fraction of energy kept as a function of the fraction of coefficients kept for $16 \times 16$ blocks of motion compensated residuals.

5.2.3 Image Compression

We now evaluate our method to encode one frame of the video sequences directly, as in usual image compression. The images used are the ones shown in figure 5-1. We implement this experiment for $8 \times 8$ block-size only. The results for keeping $c$ coefficients are shown in figure 5-8.

In terms of analyzing the different algorithms, we can see that they are also really close, with no significant differences among them. More correlation in the original image can be exploited in terms of compressing the residuals. Therefore, a larger fraction of the total energy can be preserved with fewer coefficients. This does not
mean it is better to encode images directly, since the total energy is much more than that of the prediction residuals, and therefore we will need many more bits to encode it.

5.3 Algorithm Comparison

A brief comparison among the four algorithms will now be presented, summarizing the previous results. In terms of performance, for small values of $K$ there is no significant difference among the solutions produced by these algorithms. Although algorithm A is not globally optimal, its difference from the optimal solution is negligible. This
difference increases with the number of transforms $K$ [11]. We note at some points the solutions from algorithms B and C may appear to be greater or smaller than that of algorithm D. This happens because we do not find $\lambda$ to give us some values of $c$, and these are thus represented as a line connecting the achievable points.

We now discuss the different applications in which we could use each of these algorithms. We note that all of them need to compute the transforms for all blocks. After this computation, algorithm A is the one with smallest complexity, so, although it has the lowest performance, it is the best if computation must be done quickly. Algorithms B and C find a globally optimal solution for many, though not all, values of $c$. We note the main difference between them is based on how many times we need to change the value of $\lambda$ to find an approximation achieving the value of $c$ we want. If there is a good guess on the value of $\lambda$, and thus the algorithm will not need to be called many times, then algorithm C will be better. If we are going to work with many different values of $\lambda$, then algorithm B is better since it will only need to change its last part, and will not need to have the first part of its computation done again. Another structural advantage of algorithms B and C in relation to the others is that they may be done independently for every block, and therefore may significantly improve their performance by using parallel implementations. Finally, Algorithm D is the best if we are searching for a specific value of $c$, or if we wish to find all possible values of $c$. The comparison between these algorithms is summarized in table 5.1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Optimality</th>
<th>Convergence</th>
<th>Complexity</th>
<th>Parallelizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Local</td>
<td>For all $c$</td>
<td>$O(KMN)$</td>
<td>Partially</td>
</tr>
<tr>
<td>B</td>
<td>Global</td>
<td>For some $c$</td>
<td>$O(KNM \log M)$</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>Global</td>
<td>For some $c$</td>
<td>$O(KMN)$</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>Global</td>
<td>For all $c$</td>
<td>$O(M^2N^2)$</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.1: Algorithm comparison as a function of type of optimality, convergence, complexity without the transform operation, and parallelizable over different blocks.
Chapter 6

Quantization

To apply our algorithm in a general video compression system, we need to take quantization into account. Quantization is used to map a representation from real transform coefficient values to a finite set of values given by the reconstruction levels. For more information in quantization, see [24]. In this chapter, we consider what changes need to be made if we introduce quantization to our previous system. We consider here that we choose a transform and a truncation before we quantize. This setup is shown in figure 6-1.

![Diagram showing the sequence of operations: Form Blocks, Transform, Truncation, Quantization.](image)

Figure 6-1: Setup with truncation and quantization.

Quantization is performed for each transform $T^{(i)}$ using a block-quantizer $Q^{(i)}$ that is associated with that transform. We initially keep the model general, by assuming this quantizer is any vector quantizer.

Our objective is to minimize the MSE between the output of the quantizer and the original signal, under a constraint on the number of coefficients possible for each block. This is not a constraint on the number of bits in each block, since that is much harder to compute as it depends on the entropy coder. This will be discussed in more detail later in this thesis.

Let $Q$ be the vector of quantizers, where $Q_i$ is the quantizer associated with block $i$. Our problem here may be written as:
From the energy equation, we do not need to consider the inverse transform step in our optimization. This problem is harder than the one before. As an example, consider the case where we were to keep all the coefficients in a block. Without quantization, the choice of transform would not make a difference, since the total energy is the same. However, in the case with quantization, the transform would matter, since some transforms may have more quantization noise than others. For example, if we are transmitting any integer values, then the impulse basis will not have any quantization noise since its pixels usually take integer values; In this same example, the 2-D DCT will have more noise, since it tends to take real values for most of its coefficients.

A natural question is how much our setup will change. We will study this question by seeing an example in the next section. We then adapt the four algorithms to this case, and to the case where we have a bit-rate constraint.

6.1 Uniform Quantization

An interesting question to ask is: in the case where we use a uniform quantizer with fixed step size for all coefficients under all transforms, is this problem the same as the one we had before? In other words, is the solution to minimize mean-squared error in the quantized output given sparsity constraint the same as the one if we just have the truncation output?

The answer is no. Consider the single block given in table 6.1 and two transforms, \( T^{(1)} \) and \( T^{(2)} \). In this case, we use a mid-thread uniform quantizer with step-size 2. This quantizer's input-output relationship is indicated in figure 6-2. The resulting quantized coefficients when we use this quantizer for our signal are given in table 6.2. There is a small error when quantizing the coefficients of the second transform, while there is a significant error when quantizing those of the first transform.
### Table 6.1: Example of transform coefficients.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$T^{(1)}x_1(n)$</th>
<th>$T^{(2)}x_1(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\sqrt{24} \approx 4.899$</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>$\sqrt{23} \approx 4.795$</td>
<td>$-4$</td>
</tr>
<tr>
<td>3</td>
<td>$\sqrt{1.2} \approx 1.095$</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>$-\sqrt{0.8} \approx -0.8944$</td>
<td>1</td>
</tr>
</tbody>
</table>

We now choose to keep three coefficients from our block. Without quantization, the best choice would be to keep the first three coefficients from the first transform.

### Table 6.2: Example of quantized transform coefficients.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$Q(T^{(1)}x_1(n))$</th>
<th>$Q(T^{(2)}x_1(n))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>$-4$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6-2: Mid-thread uniform quantizer with step-size 2.
But, when we consider the quantization error, the error of using the first transform would be:

\[ \text{Error} \approx 0.899^2 + 0.795^2 + 0.905^2 + 0.8944^2 = 3.0585 \]

The error of using the second transform will be entirely due to not using the fourth coefficient. In this case, this is equal to 1. Therefore, the best transform to use is transform 2.

We now interpret this by observing what is happening to the error. Because \( \hat{x} \) is a truncation of \( x \), \( QT\hat{x}(n) \) must be zero when \( T\hat{x}(n) \) is zero. Therefore, we can write our answer as a sum of two error terms:

\[
||T x - QT\hat{x}||_2^2 = ||T x - T\hat{x}||_2^2 + ||T\hat{x} - QT\hat{x}||_2^2 \triangleq \epsilon_t + \epsilon_q \quad (6.2)
\]

We note we have defined two error terms, \( \epsilon_t \) and \( \epsilon_q \). The first error term is the truncation error, and corresponds to the error given by the truncation with \( c \) coefficients. The second error term is the quantization error, given by the energy of the difference between the original coefficients and the quantized coefficients. In our previous methods, our focus was to minimize \( \epsilon_t \), as we assumed that \( \epsilon_q \approx 0 \). Now we wish to minimize the sum of these two errors.

To understand the quantization error, take a uniform quantizer with step-size equal to \( \Delta \). Then quantizing our coefficients will add an error of up to \( (\frac{\Delta}{2})^2 \) for each coefficient. Therefore, if we keep \( c \) coefficients, we find the bound:

\[
\epsilon_q \leq \left( \frac{\Delta}{2} \right)^2 c \quad (6.3)
\]

For smaller values of \( c \), we have small \( \epsilon_q \), and it will thus be more likely that minimizing \( \epsilon_t \) will suffice. For large values of \( c \), we note \( \epsilon_q \) will significantly increase.

Meanwhile, for a fixed signal \( x \), we note that we can bound \( \epsilon_t \) by arguing that keeping the \( c \) largest coefficients will at most case keep \( \frac{c}{N} \) of the total energy, or:
For large values of $c$ we note $\epsilon_t$ will decrease. Therefore, the problem of finding the optimal solution taking into account the quantization step will be especially significant as we have larger values of $c$.

This is consistent with many of the previous observations. For example, the choice of transform actually makes a difference when we keep all coefficients, since $\epsilon_t = 0$, but $\epsilon_q$ may be large. Meanwhile, the choice of transform will not matter when we keep no coefficients, since $\epsilon_t = ||x||_2^2 = ||Tx||_2^2$ for every transforms $T$ and $\epsilon_q = 0$.

In the case of the previous example, we note for the first transform $\epsilon_t = 0.8$ and $\epsilon_q \approx 2.2585$, but for the second transform $\epsilon_t = 1$ and $\epsilon_q = 0$. The second transform becomes the best due to the large value of $\epsilon_q$ for the first transform. This is enhanced by the fact that we are using a large value of $c$, and consequently smaller values of $\epsilon_t$.

### 6.2 Modifying Algorithms

In this section, we consider how to adapt the previous algorithms so we can take quantization into account. This will attempt to solve the problem in equation (6.1).

We here assume a block is quantized in a coefficient-by-coefficient manner, through scalar quantization. This is done so that quantization complexity of a block for a specific transform becomes $O(M)$, and quantization output may be computed only once for each transform. Combined with transforming the signal before scalar quantization, this is the kind of quantization that is mostly used in practice. For vector quantization, we would need to evaluate all possible combinations of quantized coefficients in a block, giving us a total complexity of $O(2^M)$ which is not fathomable. In all the following sections, we note that a complexity of $O(KNM)$ will be used to quantize all coefficients of all transforms in all blocks.
6.2.1 Algorithm A

Algorithm A will still consist of two steps, but both of these steps will change to take quantization into account.

Step A1: Fix $T$ and $Q$, find the best choice of $c$. In other words, solve:

$$\min_{c,\|c\|=c} \sum_i \|T_i x_i - Q_i T_i \hat{x}_i\|_2^2$$

(6.5)

Step A2: Fix $c$, find the best choice of $T$ and $Q$. In other words, solve:

$$\min_T \sum_i \|T_i x_i - Q_i T_i \hat{x}_i\|_2^2$$

(6.6)

The only significant difference to the previous version of algorithm A is that not only do we need to compute the transforms, but we also need to compute the quantized coefficients associated with each of the transforms. Since quantization may be done in a coefficient-by-coefficient basis, we only need to quantize all the coefficients for each possible transform once. In this case, our complexity will remain equal to $O(KNM)$ per iteration.

6.2.2 Algorithm B

Consider now the new Lagrange optimization problem taking into account the quantized solution:

$$\min_{c,T} \sum_{i=1}^M \|T_i x_i - Q_i T_i \hat{x}_i\|_2^2 + \lambda c_i$$

(6.7)

This metric will also not capture all the possible values of $c$, but, if we find some value of $c$ that solves this problem, it will also solve the original problem in equation (6.1).

Before, we defined the block-optimal energy function as the maximum truncation energy as a function of the number of coefficients kept. We here define the block-optimal error energy function as $\tilde{E}(c)$ denoting the smallest error energy we can get with $c$ coefficients, using the best possible transform and its respective quantizer.
Instead of maximizing the profit as we did before, this time we minimize the cost function $f_i(c)$ in each block defined by:

$$
\tilde{f}_i(c) = \tilde{E}_i(c) + \lambda c
$$

(6.8)

The implementation is going to be the same as the previous one for algorithm B. Instead of maximizing each $f_i(c)$, we minimize each $\tilde{f}_i(c)$. To see that this solution is optimal, all we need to do is to show that the overall metric can be separated in a block-by-block basis, and this proof will follow by definition, as in the proof for theorem 3.1.2.

In this implementation, the main change is to compute the quantized representation of all blocks under all transforms. To compute $\tilde{E}_i(c)$, we will now need to sort the values of the improvements of each transform coefficient in each block, given by $|T_i x_i(n)|^2 - |T_i x_i(n) - Q T_i x_i(n)|^2$. In this case, the complexity will have $O(KN M)$ added, and will result in a total complexity of $O(KN M \log(M))$. In the case where we iterate multiple times to find a specific fraction of the total number of coefficients, each subsequent iteration will not need the quantization, and so our complexity will be $O(N M)$. This algorithm is still parallelizable over all blocks.

### 6.2.3 Algorithm C

Without quantization, algorithm C works by verifying whether the benefit in reduction of MSE of adding a coefficient is greater than or not. We consider optimizing the same problem as in (6.7). We separate this problem in a block-by-block manner, as solving:

$$
\min_{c_i,T_i} ||T_i x_i - Q_i T_i \tilde{x}_i||^2_2 + c_i
$$

(6.9)

To add quantization, we note that adding a coefficient will still increase the second term by $\lambda$, therefore we need the first term to decrease by at least $\lambda$ to add the coefficient. The decrease in the first term by using an approximation $\tilde{x}_i$ is equal to $||T_i x_i||^2_2 - ||T_i x_i - Q_i T_i \tilde{x}_i||^2_2$. Therefore, all we need to do is compare the reduction
$|T_i x_i(n)|^2 - |T_i x_i(n) - Q_i T_i x_i(n)|^2$ to \( \lambda \). This gives us the following lemma, which is similar to lemma 3.1.1, but accounts for quantization.

**Lemma 6.2.1** Fix a block \( i \), an orthonormal transform \( T_i \), and a quantizer \( Q_i \). Let \( \tilde{x}_i \) be a truncation of our signal \( x_i \). Then \( \tilde{x}_i \) is a solution for the optimization problem below:

$$
\min_{c_i} \| T_i x_i(n) - Q_i T_i \tilde{x}_i(n) \|^2 + \lambda c_i 
$$

(6.10)

with \( c_i \) coefficients under \( T_i \) if and only if the following two conditions are true:

(a) If $|T_i x_i(n)|^2 - |T_i x_i(n) - Q_i T_i x_i(n)|^2 > \lambda$, then $T_i \tilde{x}_i(n) = T_i x_i(n)$.

(b) If $|T_i x_i(n)|^2 - |T_i x_i(n) - Q_i T_i x_i(n)|^2 < \lambda$, then $T_i \tilde{x}_i(n) = 0$.

Using the same argument as in theorem 3.1.2, we then propose an algorithm by computing each transform and its quantized versions, then keeping the coefficients that satisfy $|T_i x_i(n)|^2 - |T_i x_i(n) - Q_i T_i \tilde{x}_i| \geq \lambda$, and finishing by choosing the transform that will then optimize (6.9). Furthermore, any solution to (6.7) may also be found by using this procedure and choosing to keep or not to keep the coefficients equal to \( \lambda \).

We now discuss the complexity of this algorithm. We do not take into account the complexity of computing the transforms in each block. The next step would be to quantize these blocks, compute the improvements, and compare to \( \lambda \). This gives us a total complexity of \( O(KNM) \). This complexity is not reduced with multiple iterations since it still depends on the value of \( \lambda \). This algorithm will still be parallelizable, since we apply it separately to each block.

**6.2.4 Algorithm D**

We here write the problem again as a function of adding energies. Our objective is now to minimize the sum of each of the block-optimal error energy function, or, mathematically:
We here use the same recursion as before, by defining $S_m(c)$ as the minimum error when using $c$ coefficients in blocks 1 up to $m$, and then writing:

$$S_m(c) = \min_k S_{m-1}(c - k) + \widetilde{E}_m(k)$$

This is similar to the multiple-choice knapsack problem in the situation where we wish to minimize our costs, and the solution will be similar to the one we had before.

In terms of complexity, the total complexity here will be given by that of computing the transforms and its quantizations as well as the block-optimal error energy functions, and then the same complexity as for the previous version of algorithm D of $O(N^2M^2)$. Therefore, this algorithm will have a total complexity of $O(N^2M^2)$.

### 6.3 Bit-rate Constraint

We now discuss solutions for the problem of optimizing the quantized residual under a bit-rate constraint, instead of a constraint in the number of coefficients. We wish to code our signal with at most $B$ bits, and want to minimize the MSE under this bit-rate constraint. This problem is much harder than the one we had before, since it is not necessarily possible to allocate each number of bits $b_i$ to each block $i$.

As an example, if we always use the same number of bits $b_c$ per coefficient, then we note that this problem is the same as keeping the most energy with only $c$ coefficients, where the number of coefficients $c$ is given by:

$$c = \frac{B}{b_c}$$

We here assume that all the bit-rate is used to transmit the quantized coefficients. We ignore the additional bits used to transmit the location of transmitted coefficients, as well as the bits used to transmit the used transform. We let $B_{max}$ be the maximum
number of bits that can be allocated to a block.

Our algorithms to associate the multiple transforms and number of coefficients to each block could be adapted to solve this problem instead. We begin by noting that algorithm A would be changed by iterating first over the transforms with a fixed number of bits in each block, and then finding the optimal number of bits to be allocated in each block given the used transforms. In this case, it will still be locally optimal, and the solution will have a complexity of \( O(N^2 B_{\text{max}}^2 + KNMB_{\text{max}}) = O(N^2 B_{\text{max}}^2) \) per iteration. This is slow when compared to the other algorithms.

To modify algorithms B, C, and D, we need to define the block-optimal bit error function, \( \epsilon_i(b) \). This function indicates what is the smallest error energy that we may achieve using \( b \) bits or less in block \( i \).

In optimizing our new metric, we now wish to minimize the energy of the error under a rate-distortion scenario. In this case, this problem may still be split into separate independent optimization problems in each block. For each block, we solve:

\[
b_i = \arg \min_b (\epsilon_i(b) + \lambda b)
\]

Algorithm B solves this by computing all possible \( \epsilon_i(b) \) for each block, and then observing which value of \( b \) will make the metric \( \epsilon_i(b) + \lambda b \) minimal. We still need to find all block optimal bit error functions, as well as optimize each of these functions. The step of computing the block optimal bit error functions for a fixed block and transform will still take \( O(MB_{\text{max}}) \), since this problem is equivalent to the 0-1 knapsack problem. Evaluating what the best transform is for a fixed block, and every value of \( b \) requires \( O(KB_{\text{max}}) \) operations. Furthermore, finding the optimal \( b_i \) in every block will take \( O(NB_{\text{max}}) \). Therefore, this gives us a total complexity of \( O(KNMB_{\text{max}}) \). Further steps will take \( O(NB_{\text{max}}) \) if we iterate our algorithm for many values of \( \lambda \), since we only need to repeat the last step.

For algorithm C, we take each possible transform and each possible coefficient, and we consider the improvement of adding a coefficient to a block in terms of the improvement per bit, defined by the difference in energy \( |T_i x_i(n)|^2 - |T_i x_i(n) - Q_i T_i x_i(n)|^2 \)
divided by the number of bits used to encode this coefficient. We choose to add a coefficient if its improvement per bit is greater than $\lambda$, and not add a coefficient if its improvement per bit is smaller than $\lambda$. Finally, we choose the best transform by choosing the one that minimizes the metric in equation (6.14). This algorithm is globally optimal, and its proof follows the same strategy as before, thus we choose to omit it. This is a fast algorithm, with the same complexity as before of $O(KNM)$.

When comparing algorithms B and C, algorithm C will be better even for further iterations, as long as $K \leq \frac{B_{\text{max}}}{M}$, or if the number of transforms is smaller than the average number of bits per coefficient.

Algorithm D will remain optimal, but we now have to compute the block-optimal bit error function $\epsilon_i(b)$, and then find the best number of bits to be allocated for each block. We again make use of the representation as the multiple-choice knapsack problem, and the fact that we can write our problem as:

$$\min \sum_i \epsilon_i(b_i)$$

This will be a globally optimal solution to the previous problem. In this case, the algorithm will have a total complexity of $O(KNM B_{\text{max}} + N^2 B_{\text{max}}^2) = O(N^2 B_{\text{max}}^2)$.

Table 6.3 summarizes the differences among the algorithms considering quantization error with a bit-rate constraint. As we can see, the only major change is that algorithm A is now much slower than algorithms B and C, since it must ensure a specific total of bits.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Optimality</th>
<th>Convergence</th>
<th>Complexity</th>
<th>Parallelizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Local</td>
<td>For all $c$</td>
<td>$O(N^2 B_{\text{max}}^2)$</td>
<td>Partially</td>
</tr>
<tr>
<td>B</td>
<td>Global</td>
<td>For some $c$</td>
<td>$O(KNM B_{\text{max}})$</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>Global</td>
<td>For some $c$</td>
<td>$O(KMN)$</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>Global</td>
<td>For all $c$</td>
<td>$O(N^2 B_{\text{max}}^2)$</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6.3: Algorithm comparison with quantization and under a constraint on the number of bits as a function of type of optimality, convergence, complexity without the transform operation, and parallelizable over different blocks.
Chapter 7

Conclusions

7.1 Summary

Transform coding has been used in many signal coding applications. In H.264 and HEVC video coding, transforms are used to find representations where the original signal is as sparse as possible, while keeping as much energy as possible. In this case, the correlation function is well represented by a two-dimensional Markov-1 model. Therefore, the 2-D DCT is typically used in many video and image sequences, since it is very close to the KLT of these signals.

For motion compensation residuals, the energy is not well represented by a Markov-1 model. The energy tends to be focused in one direction, following edges of the original image. Due to this, multiple-transform video encoding with 1-D directional DCTs has been developed so it can use the smallest number of coefficients to encode motion compensated residuals. In this thesis, we considered one main issue when using multiple-transforms to encode a signal: given a constraint on the number of coefficients, it is non-trivial to find the optimal transform and number of coefficients to be allocated for each block.

Two algorithms had been developed to deal with this issue. Algorithm A is an iterative algorithm with low complexity, that finds a locally optimal solution, but it does not necessarily converge to a globally optimal solution. Algorithm B is a globally optimal algorithm that solves the previous problem for many values of the number
of coefficients, by dealing with a rate-distortion metric instead. This algorithm has a higher complexity, but may be implemented in parallel for each block.

In chapter 3, we proposed an alternate version for algorithm B, called algorithm C, based on thresholding. We then showed that algorithm C would always give us an optimal solution, and that any solution for the rate-distortion optimization would be obtained in some variant of algorithm C. This algorithm is very fast computationally and may be implemented in parallel for each block. Therefore, it may be used in many practical applications.

In chapter 4, we proposed another algorithm to solve the same problem, called algorithm D. This algorithm can be found by representing our problem as the multiple-choice knapsack problem and solving it with dynamic programming. This algorithm solves the original optimization problem of finding the best representation for any number of coefficients. This algorithm is slower than the previous three, although it is able to solve a problem the others have not solved before.

In chapter 5, we compared the different algorithms for video compression. We realized there is not much difference between the solutions, apart from the computational complexity to obtain them. In chapter 6, we considered how these algorithms would change to take quantization into account, as well as a bit-rate constraint.

### 7.2 Future Research

There are many possible future directions of work in multiple-transform signal compression.

First, we note that there is still ample room to develop faster algorithms to find the best transform associated with each block. All algorithms that we found until now require that we compute the transforms for all blocks. A solution that does not require these computations could be extremely computationally efficient.

We also note that we could easily adapt these algorithms in the case of weighted $\ell_2$ norm, as long as we redefined the meaning of orthonormal basis for this new metric. These norms are especially useful for different kinds of perceptual based error
metrics, and therefore adapting the algorithms could be extremely helpful. A possible
direction of development would be to adapt these algorithms so they can take basis
that are not orthonormal, or take frames instead of basis [10], in an approach closer
to compressive sensing.

Another area of development for multiple-transform video compression is to reduce
the side-information associated with sending the optimal transform. We believe this
may be done using estimation [25]. This is an area of our current research.

There are also many other signal processing problems that may be related to
variants of the knapsack problem, as well as to many classical problems in theoretical
computer science. To find such problems and understand this equivalence, as well
as to find more applications between the fields, will be another useful direction for
future research.
Bibliography


