LOSSY COMPRESSION OF INDIVIDUAL SIGNALS BASED ON ONE PASS CODEBOOK ADAPTATION

by

Christopher Chan

Memorandum No. UCB/ERL M95/58

1 July 1995
LOSSY COMPRESSION OF INDIVIDUAL SIGNALS BASED ON ONE PASS CODEBOOK ADAPTATION

by

Christopher Chan

Memorandum No. UCB/ERL M95/58

1 July 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720
LOSSY COMPRESSION OF INDIVIDUAL SIGNALS BASED ON ONE PASS CODEBOOK ADAPTATION

by

Christopher Chan

Memorandum No. UCB/ERL M95/58

1 July 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720
ABSTRACT

Lossy Compression of Individual Signals based on One Pass Codebook Adaptation

Christopher Chan

Universal lossless codes have been proven to exist [5], and practical universal lossless coding schemes have been constructed [27, 6]. Given the attractive property of a universal code, namely that it can achieve an asymptotic rate approaching the entropy of the source, without knowledge about the source distribution a priori, a lossy counterpart of a universal code is highly desired. In this work, our objective is to develop a practical one pass lossy compression algorithm that approaches this universality property.

We have addressed the problem within the framework of vector quantization (VQ). Traditionally, vector quantization relies on a codebook that has to be "trained" in advance on a set of signals, called the training sequence, generated from the target source. This assumes knowledge of the source distribution a priori. To release this restrictive assumption, we have investigated on methods to adapt the VQ codebook to the actual signal to be coded.

In the lossless domain, Ziv and Lempel have introduced a simple scheme for universal data compression, which later gives rise to a whole class of algorithms. This class of algorithms is based on a dynamic dictionary of source sequences parsed as strings, and replacing subsequent occurrences of these strings by pointers. Generalizing such exact string matching to approximate string matching, i.e. one with distortion, we have incorporated the Lempel-Ziv idea into adaptive VQ, and developed a universal lossy compression algorithm.

The essence of the adaptive algorithm is the criterion it uses to encode the source vectors and adapt the codebook. A source vector can be simply encoded by a
codevector in the current codebook, or it can invoke the addition of a new codevector. Current codevectors can be moved around or even be deleted as the source distribution varies. These actions are taken to minimize the Lagrangian $R + \lambda D$, where $R$ is the bit rate involved in taking an action, $D$ is the distortion it introduces, and $\lambda$ is a parameter chosen to control the operating point of the algorithm on the operational rate-distortion curve. In a sense, bits are "traded" for a reduction in distortion by modifying the codebook. Hence, the technique is called rate-distortion Lempel-Ziv (RDLZ).

Experiments are performed on various sources, namely stationary Gaussian sources, "switching" (time-varying) Gaussian sources, standard test images, and composite images. Advantages of adaptation using the RDLZ algorithm are demonstrated in these experiments. When a VQ codebook is mismatched with the actual source signal, significant improvement in rate-distortion performance is achieved with the adaptive algorithm. Empirical evidence on the universality of the algorithm is also given.
Contents

List of Figures ii

1 Introduction and Motivation 1
1.1 Universal Lossy Source Coding 1
1.2 Lossy Counterpart of Lempel-Ziv Coding 3
1.3 Vector Quantization 4
1.4 Outline of the Report 7

2 Practical One Pass Algorithm 8
2.1 One Pass Adaptive Vector Quantization 8
2.2 Definitions and Notations 9
2.3 Overview of the Algorithm 10
2.4 Codebook Adaptation 12
   2.4.1 Adding Codevectors 13
   2.4.2 Splitting Codevectors 15
   2.4.3 Moving Codevectors 16
   2.4.4 Deleting Codevectors 17
   2.4.5 Updating the Index Codes 18
2.5 Quantization Issues 19
2.6 Syntax of the Encoded Bitstream 20
2.7 Complexity of the Algorithm 21

3 Experimental Results 23
3.1 Performance Gain with Adaptation 23
3.2 Universality of the Algorithm 24
3.3 Performance Penalty on Stationary Sources 26
3.4 Results on Images 27

4 Conclusion and Future Work 36

Bibliography 38
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Adaptation of the quantizer can achieve better R-D performance.</td>
<td>9</td>
</tr>
<tr>
<td>2-2</td>
<td>Schematic of the codebook adaptation. The decoder keeps track of the encoder codebook changes</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>through side information and n_j, the count of samples that fall in each codevector cell over</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a period of time.</td>
<td></td>
</tr>
<tr>
<td>2-3</td>
<td>Minimizing the Lagrangian $J = R + \lambda D$ as the codebook adaptation criterion.</td>
<td>11</td>
</tr>
<tr>
<td>2-4</td>
<td>The encoded bitstream consisting of codevector indices as well as codebook updates.</td>
<td>12</td>
</tr>
<tr>
<td>2-5</td>
<td>Several ways of codebook adaptation: Add, Split, Move and Delete.</td>
<td>13</td>
</tr>
<tr>
<td>2-6</td>
<td>Modified entropy coding of the codevector indices. In this example, Huffman coding is used</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>for codevectors in the hotlist plus the prefix of the rest of the codebook ($s_{cold}$).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$C_{hot} = {s_0, \ldots, s_{10}}$, $C_{cold} = {s_{11}, \ldots, s_{18}}$.</td>
<td></td>
</tr>
<tr>
<td>3-1</td>
<td>Source mismatched with the initial codebook. (a) Distribution of the source. Signal source</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>is correlated Gaussian source with $\rho = 0.95$, $\sigma^2 = 1$. (b) Initial codebook</td>
<td></td>
</tr>
<tr>
<td></td>
<td>designed on training sequence from an i.i.d. Gaussian source with $\sigma^2 = 0.25$.</td>
<td></td>
</tr>
<tr>
<td>3-2</td>
<td>Comparison of RDLZ and static codebook VQ on a source mismatched with the initial codebook.</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Performance with optimal codebook is also shown.</td>
<td></td>
</tr>
<tr>
<td>3-3</td>
<td>(a) Source distribution at two different time instants $t_1$ and $t_2$. (b) Codevectors with</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>shortest-index codes at $t_1$ and $t_2$.</td>
<td></td>
</tr>
<tr>
<td>3-4</td>
<td>Comparison of RDLZ and static codebook VQ on a time-varying source. The source is switching</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>back and forth between correlated Gaussian sources with 1) $\rho = 0.3$, and 2) $\rho = 0.95$.</td>
<td></td>
</tr>
<tr>
<td>3-5</td>
<td>Universality of RDLZ. The source and initial codebook are identical to that in Fig. 3-1.</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Results on 160k and 800k samples from the source are shown.</td>
<td></td>
</tr>
</tbody>
</table>
Performance penalty due to adaptation on an i.i.d. Gaussian source. —
is the performance of the optimal vector quantizer, × are operating
points of RDLZ with initial codebooks of different size. 29

"Barbara" 30

(a) Original "Ariel". (b) "Ariel" RDLZ encoded at 0.5 bpp, PSNR =
23.5 dB. 30

Comparison of RDLZ and static codebook VQ on "Ariel". Vector
dimension is 4 × 4. Codebooks trained on "Barbara". 31

Original composite image. 32

Composite image coded with static codebook VQ at 0.43 bpp. 33

Composite image coded with RDLZ at 0.43 bpp. 34

Comparison of RDLZ and static codebook VQ on composite image.
Vector dimension is 4 × 4. Initial codebook trained on "Barbara". —
x— is the performance of static codebook VQ, ····· is the operational
rate-distortion curves of RDLZ with the labelled initial codebook size,
—o— is the convex hull of these curves. As a comparison,—|— is
the performance of static codebook VQ with codebook trained on the
composite image. 35
Acknowledgements

The original concept of this work is conceived by my research advisor, Prof. Martin Vetterli, whose insights and enthusiasm have sparked my interest in pursuing it down the road. There have been ups and downs all along, and without Martin’s continual intellectual and mental (and of course, financial) support, I may not have persisted. I deeply appreciate his commitment to research and to the intellectual growth of his students.

At various stages, many other people have given valuable comments and suggestions. For this, I would like to thank Philip Chou, Zoran Cvetković, Masoud Khansari, Antonio Ortega, Maximilian Ott, Kannan Ramchandran, Sivarama Venkatesan and Gregory Yovanof. Their inputs have certainly helped shaping this work into its current form. I would also like to acknowledge Tom Lookabaugh, Eve Riskin and other people who have provided the VQ and TSVQ programs in the decaf.stanford.edu ftp site. In the same manner, I would like to thank Ton Kalker for sharing his MATLAB utilities with me.

Special thanks are given to Prof. Avideh Zakhor for being my second reader, Heather Brown for her tireless effort in insulating the graduate students from the bureaucracy, and Talal Shamoon for offering to proofread this report, although I am running too late for that. I am grateful to all my friends and colleagues at Cory 307 and 319, and everybody in the Wavelet Group, who have brought so much warmth and fun to the work environment. Their cooperation and encouragements are especially appreciated. Of course, I am indebted to the Infopad project for its support of this work.

Finally, my heartfelt thanks go to my friends near and far, to Cindy, and to my parents, for supporting and inspiring me in every aspect of my life; and to God, for everything comes from Him.
Chapter 1

Introduction and Motivation

1.1 Universal Lossy Source Coding

Rate distortion theory has provided a theoretical framework that defines the achievable optimality of lossy source coding, and establishes the existence of source codes performing near the optimum. These codes, however, assumes \textit{a priori} the knowledge of the source distribution. And even with this assumption, the design of the code is a non-trivial process.

For instance, in the vector quantization (VQ) framework, some necessary optimality conditions have been proved [9]. If a vector quantizer $Q^*$ is optimal over any vector quantizer $Q$ with a particular vector dimension $\ell$, then it must satisfy these conditions. Two well known optimality conditions are the \textit{nearest neighbor condition} and the \textit{centroid condition}. Based on these conditions, various versions of an iterative method for designing quantizers have been introduced. They are called variably the generalized Lloyd algorithm [9], k-means algorithm [17], or the Linde-Buzo-Gray (LBG) algorithm [14]. These iterative algorithms, although guaranteed to converge, do not always converge to the optimal quantizer. More likely than not, they will go to some local minima. Hence, additional techniques have been introduced to evade local minima, such as \textit{simulated annealing} [8, 21], but they add to the computational complexity significantly.

Another problem in designing the quantizer using these iterative algorithms is
that the distribution of the source $X$ is unknown. Hence, the designer has to rely on a training sequence $(X_1, \ldots, X_m)$ of vectors, where the $X_i$'s are assumed to have the same distribution as $X$. In coding of portrait images, for instance, a large set of portrait images will be used as the training sequence. However, it is not difficult to conceive of situations where this assumption is too restrictive and unreasonable. In our example, the images to be coded may be landscapes, or brain CATs.

This motivates the work on developing universal codes for lossy source coding. A block code is universal if it achieves the rate-distortion bound as the blocklength approaches infinity, on all sources from a class of sources. In other words, the code "learns" from the actual source signals and adapts to them. Consider a stationary non-ergodic source, where the actual signal can be generated from one of several stationary ergodic sources chosen arbitrarily at the beginning. Asymptotically, the code should achieve the optimal performance of the corresponding stationary ergodic source.

Previous works in this area fall into two categories: 1) existence proof and code construction, and 2) practical, computationally efficient algorithms. In works by Ziv [25], Neuhoff et al. [18], MacKenthun and Pursley [16], Kieffer [11], Linder et al. [15], Yu and Speed [22] and others, the existence of universal lossy source codes is established under various assumptions about the class of sources, the distortion measure, and the type of convergence of the rates to the rate-distortion bound. The codes constructed in these proofs, however, cannot be implemented in practical coding situations, due to their high computational complexity and/or intensive memory requirements. Hence, work is still in progress towards the goal of developing a practical algorithm for finding a universal lossy code.

Two different approaches have been taken in constructing the practical universal code. The first approach involves iterations on the source signal, requiring the encoder to look at the source signal more than once in determining the adaptation of the code. Algorithms taking this approach are called two pass algorithms. In [10], a VQ codebook is modified by splitting codevectors with the largest partial distortion in encoding a source signal, and running the LBG algorithm on the same signal to determine the final locations of the new codevectors. In [7], the encoder finds
the optimal codebook for the part of the source signal seen thus far at increasing
intervals, and refines its initial codebook by transmitting codebook changes with
increasing accuracy. The tradeoff between spending bits on transmitting updates of
the codebook and specifying codevector index is addressed in [23], which also involves
finding the optimal codebook periodically. Lightstone and Mitra [13] then addressed
the rate-distortion tradeoff by using the entropy-constrained framework. One common
characteristic of these algorithms is that some kind of iterative techniques, such as
the LBG algorithm, is involved at certain steps of the algorithms.

The second approach, described as one pass, adapts the code as encoding pro-
ceeds without using iterative techniques. The source signal is hence parsed through
just once. Steinberg and Gutman [20] proposed an algorithm based on string match-
ing with distortion, which achieves $R(D/2)$ for a large class of stationary sources
and distortion measures. Zhang and Wei [24] introduced the “gold-washing” method,
which sequentially updates the set of codevectors. In [19], the source distribution
function is estimated based on the occurrence counts of the scalar quantizer bins, and
the quantizer adapts without requiring side information. Our work adopts the one
pass approach.

Advantages of one pass algorithms include simplicity, computational efficiency,
and low encoding delay. In lossless source coding, a family of universal coding algo-
rithms which exhibits these advantages is Lempel-Ziv coding [26, 27]. It is proved to
be asymptotically optimal, and because of its speed and efficiency, it has become the
standard algorithm for file compression on computers.

1.2 Lossy Counterpart of Lempel-Ziv Coding

Since Lempel-Ziv coding provides an elegant and simple way for doing universal
source coding in the lossless domain, it is natural to ask the question: is there a lossy
counterpart of Lempel-Ziv coding? This work is motivated by this question.

The basic idea behind Lempel-Ziv (LZ) coding is to replace a repeated block of
source symbols, or phrase, by a reference to a previous occurrence. Particularly, in
the approach proposed by Ziv and Lempel in 1978, labelled $LZ78$, the source sequence
seen thus far is parsed into phrases, where each phrase is the longest matching string seen previously plus one extra source symbol. These phrases constitute the dictionary.

To extend this idea to lossy source coding, phrases from the source are matched to the dictionary with distortion. Along these lines, Steinberg and Gutman proposed an algorithm that achieves a rate of $R(D/2)$ given an average distortion $D > 0$ for a large class of sources and distortion measures [20]. Koga and Arimoto [12] further proved that the algorithm achieves the rate-distortion bound asymptotically for certain sources and fidelity criteria. These works establish the theoretical foundation and motivation for a lossy version of LZ coding.

In this work, a lossy coding algorithm that has the flavor of LZ coding is developed. The algorithm, which we call the rate-distortion Lempel-Ziv (RDLZ) algorithm, is developed using the framework of adaptive vector quantization. LZ coding involves building a dictionary by parsing in phrases from the source sequence, and replacing subsequent occurrences of the phrases (via string matching) by pointers to the dictionary. If now we constrain the phrases to be of constant length $\ell$, and if string matching with distortion is performed, then the problem can be translated to a VQ context. The LZ dictionary is analogous to the VQ codebook. Phrases in the dictionary are the codevectors. Approximate string matching becomes quantization, where an $\ell$-block of source symbols is replaced by the "best" codevector according to some cost function. Here, the cost function we have adopted is the Lagrangian $J = R + \lambda D$, where $R$ and $D$ are the bit rate and distortion respectively. The essence of the problem is to construct the codebook in a one pass manner, based on the actual signal to be encoded. This point will be further elaborated in Chapter 2.

1.3 Vector Quantization

Quantization, in its simplest, is to approximate a single number by the "nearest" number (according to some distortion measure) from a predetermined finite set. If this quantization process is performed on each sample from a signal sequence separately, then it is called scalar quantization. Consider analog-to-digital conversion, which is basically scalar quantization of real values onto a finite set of numbers. The real line
(or a closed interval on it) is partitioned into cells, and each of the cells is mapped to a reproduction point within the cell.

Vector quantization is a generalization of scalar quantization. A vector is an ordered set of numbers, which is typically a block of source symbols. Instead of the real line, the multi-dimensional Euclidean space is being partitioned. Mathematically, the problem is formulated as follows. A vector quantizer $Q$ of dimension $\ell$ and size $M$ is a mapping from $K^\ell$ into a finite set $C$ containing $M$ reproduction points, called codevectors. That is,

$$Q: K^\ell \rightarrow C,$$

where $C = \{c_1, \ldots, c_M\}$, $c_1, \ldots, c_M \in K^\ell$. The set $C$ is called the codebook. A quantizer $Q$ is defined by a codebook, and associated with it, a partition of $K^\ell$ into cells $C_i = \{x \in K^\ell : Q(x) = c_i\}$, $i = 1, \ldots, M$. Each codevector in the codebook is encoded by a source code which maps its index $i \in \{1, \ldots, M\}$ into a binary string $s_i \in \{0,1\}^*$, where $\{0,1\}^*$ denotes the set of finite length binary sequences. The code is fixed rate or variable rate depending on whether $s_1, \ldots, s_M$ have the same length or not. For the latter case, the code has to be prefix free.

The average rate per vector is defined to be

$$R = \sum_{i=1}^{M} p_i |s_i|$$

where $p_i = \Pr\{Q(X) = c_i\}$ for a random source $X$. Let $d_\ell(x,y)$ be the distortion between vectors $x$ and $y$, where $x,y \in K^\ell$. Then the average distortion per vector between the source and its reproduction is

$$D = E[d_\ell(X,Q(X))].$$

First consider a vector quantizer using a fixed rate code for the codevector indices. The quantizer is optimal if the average distortion $D$ is minimized. Some necessary conditions for a quantizer to be optimal have been proposed and proved. Given a fixed decoder, the optimal encoder would partition $K^\ell$ such that the mapping from vectors in each cell to the reproduction point is a minimum distortion, or nearest
neighbor mapping. That is,

\[ d_e(x, Q(x)) = \min_{c_i \in C} d_e(x, c_i). \]

If an input vector does not have a unique nearest neighbor, it can be assigned arbitrarily to any of its nearest neighbors.

Now, given a fixed encoder, which specifies the partition of \( \mathcal{R}^e \), the optimal decoder would place the reproduction points at the centroid of each cell, so that distortion is minimized. Hence,

\[ c_i = E[X|X \in C_i]. \]

Based on these two optimality conditions, iterative codebook improvement algorithms have been developed [9, 17, 14]. Starting from an arbitrary initial quantizer, they alternately optimize the encoder and decoder using the nearest neighbor and centroid conditions respectively.

For a variable rate vector quantizer, codes for the codevector indices are not of the same length, hence the strategy mentioned above that only minimizes distortion may not be optimal in the rate-distortion sense. Using entropy-constrained vector quantization (ECVQ) [4], vector quantizers can be designed which have minimum distortion subject to an entropy constraint. It is based on minimizing the Lagrangian cost function \( J = R + \lambda D \). The two step iterative codebook improvement algorithm for fixed rate VQ becomes a three step process:

1. Given the variable rate index codes \( \{s_i\} \) and codevectors \( \{c_i\}, i = 1, \ldots, M \), find the partition of \( \mathcal{R}^e \) that minimizes \( J \).

2. Given the updated partition of \( \mathcal{R}^e \), find the optimal lossless codes \( \{s_i\} \) for the codevector indices.

3. Given the updated partition of \( \mathcal{R}^e \), find the reproduction codevectors \( \{c_i\} \) that minimizes \( J \).

Note that if codevector updates do not have to be transmitted, Step 3 is simply the centroid condition that minimizes distortion for each cell \( C_i \).
As mentioned earlier, a problem with these iterative quantizer design methods is that the source distribution is typically unknown. Hence, a training sequence \((X_1, \ldots, X_m)\) generated from source \(X\) has to be used to design the VQ codebook. If the actual signal to be encoded has a source mismatch with \(X\), then modifying the codebook to adapt to the actual source would improve performance in the rate-distortion sense. A one pass algorithm to adapt the codebook by "trading" bit rates for distortion will be presented in this work.

1.4 Outline of the Report

Chapter 2 describes the practical one pass algorithm we propose. Sections 2.2 and 2.3 give the basic definitions and notations, and an overview of the algorithm. Section 2.4 describes the algorithm in detail, particularly on how codebook adaptation is carried out. Some detailed issues of the algorithm are then explained in Sections 2.5 and 2.6, followed by a discussion on the algorithmic complexity in Section 2.7.

Chapter 3 presents experimental results of the algorithm on various sources. Section 3.1 demonstrates the benefits of adaptation on stationary sources with mismatched initial codebook, as well as time-varying sources. Section 3.2 provides empirical evidence on the universality of the algorithm. Section 3.3 discusses the performance penalty due to adaptation on a stationary source. Results on images are given in Section 3.4.

Finally, Chapter 4 gives the conclusion, and states the future directions for this project.
Chapter 2

Practical One Pass Algorithm

2.1 One Pass Adaptive Vector Quantization

When there is a mismatch between the statistics of the source and that of the training sequence from which the VQ codebook is derived, or when the source statistics is slowly varying in time, adapting the codebook to local statistical characteristics of the source signal is likely to improve the coding performance. Such a vector quantizer is described as adaptive.

In most cases, modifying the codebook involves transmission of side information, which costs bits. Hence, the idea is that bits can be spent wisely to result in an overall reduction in distortion which is worthwhile in the rate-distortion sense (Fig. 2-1).

Our objective is to develop a codebook adaptation scheme which is simple, computationally efficient, and one pass. This is in line with the spirit of Lempel-Ziv coding as mentioned in the previous chapter. Being one pass, it introduces minimal encoding delay. In fact, without looking ahead into the future during encoding, the delay is just the vector dimension $\ell$. Codebook adaptation is based on "learning" from the encoding history of the source signal, which is characterized by a small set of statistical measures (Fig. 2-2).
2.2 Definitions and Notations

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, $x_i \in A^\ell$, be a realization of the random $\ell$-dimensional vector source $X$. The source alphabet $A$ can be finite or continuous. In the discussion given in Chapter 1, $A$ was taken to be the set of reals $\mathcal{R}$. Also define $\hat{A}$ to be the reconstruction alphabet, which may or may not be the same as $A$. Let $d_e(x, y)$ be a distortion measure on $A^\ell \times \hat{A}^\ell$. The codebook $C$ is a set of reproduction codevectors \{c_j \in \hat{A}^\ell, j = 1, \ldots, M\}, each of which is associated with a cell $C_j = \{x_i | \hat{x}_i = c_j\}$, where $\hat{x}_i$ is the quantized version of $x_i$, i.e. $\hat{x}_i = Q(x_i)$. Each codevector is also associated with an index code $s_j$, of rate $r_j = |s_j|$ (length of the index codes in bits).

To estimate the empirical distribution of the source, non-overlapping windows $\tau_k$, $k = 1, 2, \ldots$ of size $L$ are defined. $L$ is the update interval, at which the local empirical distribution of the source is re-evaluated, to trace the variation of the source statistics in time. During $\tau_k$, the count of samples that fall in each codevector cell is recorded. This is denoted by $n_j^{(k)}$, the occurrence count for the $j$-th codevector during the $k$-th update window. In the same way, the empirical centroid $\hat{c}_j^{(k)}$ and average distortion $D^{(k)}(C_j, c_j)$ of cell $C_j$ with respect to codevector $c_j$ for each $j$ are
Figure 2-2: Schematic of the codebook adaptation. The decoder keeps track of the encoder codebook changes through side information and \( n_j \), the count of samples that fall in each codevector cell over a period of time.

estimated.

When a new codevector is introduced, or an existing codevector is modified (as will be described in Section 2.4), side information about the position of that codevector has to be sent. The total number of bits spent on encoding the codevector position is denoted \( B(v) \), where \( v \) is the codevector position, or if differential encoding is used, the positional difference between the new and the original codevectors.

2.3 Overview of the Algorithm

Encoding a source signal \( x \) using a fixed codebook designed on a training sequence gives an operating point on the R-D plane. We want to adapt the codebook so that the operating point moves towards the R-D bound for that source. This can be formulated as minimizing a cost function, which in the current context is the Lagrangian \( J = R + \lambda D \). Fig. 2-3 illustrates the idea of minimizing the Lagrangian as the codebook adaptation criterion.

Being one pass, the algorithm "learns" from the source signal as encoding pro-
ceeds to streamline the codebook. Hence, codebook modification occurs continuously during encoding, and it is crucial that the decoder can keep track of the codebook dynamics from the encoded bitstream. As shown in Fig. 2-4, the bitstream is consists of codevector indices \((s_j, \text{ to be precise})\) and side information regarding codebook updates, in the chronological order as codebook adaptation occurs. This is somewhat similar to Lempel-Ziv coding, where the encoding history is recorded in the bitstream.

As an overview, the algorithm goes as follows:

1. Start with an initial codebook.

2. For each source vector \(x_i\), decide whether it should initiate a new codevector to the codebook. If so, describe the new codevector in the bitstream. If not, send the index code of the nearest codevector.

3. At intervals of \(L\), decide which existing codevectors should be moved towards their respective empirical centroids \(\hat{c}_j\), and which should be deleted from the codebook. Send side information for codevector movements.

4. Also at intervals of \(L\), update the index codes \(s_j\) for the codevectors.
2.4 Codebook Adaptation

In [4], Chou et al. introduced the entropy-constrained vector quantization (ECVQ) algorithm to design vector quantizers that minimize average distortion subject to an entropy constraint. Refer back to Section 1.3 for the three steps involved in each iteration of the algorithm. The algorithm iterates these three steps until convergence is reached.

For a stationary ergodic source, \( x \) can be divided into blocks of length \( L \), i.e.

\[
(x_1 \ldots x_L), (x_{L+1} \ldots x_{2L}), \ldots, (x_{(K-1)L+1} \ldots x_{KL}),
\]

assuming that \( N = KL \) for some integer \( K \). Steps 1, 2 and 3 can then be applied to these blocks in turn. In other words, instead of iterating on the same signal many times, this scheme "iterates" on different parts of the signal, updating the codebook one step at a time. As \( L, K \to \infty \), convergence to the optimal quantizer is expected. One observation is that the overhead in sending the codebook update is negligible as \( L \to \infty \).

However, if the source exhibits non-stationary behavior, or if the source signal is of finite duration, such a naive one pass block updating scheme is no longer feasible. It fails to consider the tradeoff between the overhead in terms of bit rate and the resulting effects on distortion, and it does not adapt to variations in the source statistics adequately. We have hence studied the effects of adapting the codebook

Figure 2-4: The encoded bitstream consisting of codevector indices as well as codebook updates.
by “trading” bit rates for distortion, through adding, splitting, deleting and moving codevectors (Fig. 2-5).

2.4.1 Adding Codevectors

For a signal $x = (x_1, x_2, \ldots, x_N)$, there may be times when a source vector $x_i$ is too far from any existing codevector. This may reflect a mismatch between the statistics of the source and that of the training sequence, a change in the source distribution in time, or simply a rare event. In any case, $x_i$ is a candidate for being a new codevector.

The adaptation criterion, as mentioned above, is to minimize $R + \lambda D$. A greedy approach is adopted to decide whether to add a new codevector or not. That is, for each $i$, $x_i$ is encoded in such a way which gives the smallest $R_i + \lambda D_i$, where $R_i$ is the number of bits to encode $x_i$, and $D_i$ is the distortion thus introduced, i.e. $d_t(x_i, \hat{x}_i)$. Two options are available to encode $x_i$: 1) use the nearest codevector, or 2) send a quantized version of $x_i$ as a new codevector.
For 1),

\[ R_i = r_j^* \quad (2.1) \]
\[ D_i = d_t(x_i, c_j^*) \quad (2.2) \]

where

\[ j^* = \arg \min_{j \in \{1, \ldots, M\}} r_j + \lambda d_t(x_i, c_j). \]

For 2), differential coding is used to describe the new codevector \( c_{\text{new}} \). In Section 2.5, issues on quantizing \( x_i \) to \( c_{\text{new}} \) will be discussed, which we will skip over for the time being. An escape code is needed, to distinguish it from an ordinary codevector index, followed by the index code of the nearest existing codevector \( c_{j'} \) and the offset \( v = c_{\text{new}} - c_{j'} \). Let \( E \) be the length of the escape code, then we have

\[ R_i = r_j' + B(v) + E \quad (2.3) \]
\[ D_i = d_t(x_i, c_{\text{new}}) \quad (2.4) \]

where

\[ j' = \arg \min_{j \in \{1, \ldots, M\}} d_t(x_i, c_j). \]

Now, define \( \Delta R \) to be the increase in bit rate if adaptation (i.e. adding \( c_{\text{new}} \) for this case) is taken as opposed to no adaptation, and \( \Delta D \) to be the reduction in distortion in doing so. Based on the criterion to minimize \( J = R + \lambda D \), adaptation is taken if

\[ \lambda \Delta D > \Delta R. \quad (2.5) \]

To estimate \( \Delta R \) and \( \Delta D \) for adding codevectors, we need to consider some further details. Since the index code for \( c_{\text{new}} \) does not exist before, the index code of the nearest codevector, i.e. \( s_{j'} \), is appended an extra bit to accommodate for \( s_{\text{new}} \). Since \( s_j \) is prefix free, so appending a bit does not affect its decodability. Visualize this as splitting a leaf node of the Huffman tree. Hence, the vectors mapped to \( c_{j'} \) subsequently will suffer from an increase of index code length by 1. Denote this effect by \( \hat{n}_{j'} \), which is an estimation of \( n_{j'} \) during the current update window. As a rule of thumb, \( \hat{n}_{j'} \) can be taken to be \( n_{j'} \) in the previous update window.
Adding \( c_{\text{new}} \) will reduce the distortion in encoding \( x_i \), as well as other subsequent source vectors mapped to \( c_{\text{new}} \). To derive an operating formula for \( \Delta R \) and \( \Delta D \), we need to estimate the occurrence count \( n_{\text{new}} \) for the new codevector in the current update window. Since there is no basis for estimating this, we take a conservative approach, and assume it to be small. In our experiments, we have used \( n_{\text{new}} = 1 \) or 2. We also assume that \( c_{\text{new}} \) gives the same reduction in distortion for all subsequent cases as in encoding \( x_i \). With these assumptions,

\[
\Delta R = r_j - r^* + \frac{B(v) + E + \hat{n}_j}{\hat{n}_{\text{new}}}
\]

\[
\Delta D = d_{\ell}(x_i, c_{\text{new}}) - d_{\ell}(x_i, c_j)
\]

### 2.4.2 Splitting Codevectors

In [10], it is proposed that the codevector with the highest partial distortion should split into two codevectors. The partial distortion of the \( j \)-th codevector is defined to be \( p_j D(C_j, c_j) \), where \( p_j \) is the probability that \( x_i \) is mapped to \( c_j \). In our work, since the criterion is to minimize \( R + \lambda D \), a different approach is taken.

At the end of an update window \( \tau_k \), the empirical partial distortion of a codevector is estimated via \( n_j^{(k)} \) and \( D^{(k)}(C_j, c_j) \). The codevectors (there can be several, e.g. 10) with the highest empirical partial distortion are potential candidates to split, but we also need to consider the tradeoff between \( \Delta R \) and \( \Delta D \).

Consider the source vectors that are mapped to \( c_j \). Let \( R_j \) be the rate to encode the source vectors mapped to \( c_j \), and \( D_j \) be the average distortion thus introduced. Without splitting,

\[
R_j = r_j
\]

\[
D_j = D(C_j, c_j)
\]

Upon splitting, the codevector index code is lengthened by 1 bit, in the same way as adding a codevector. Hence, assuming that the combined occurrence counts of the two children codevectors is the same as that of the original codevector, then \( R_j \) becomes \( r_j + 1 + \text{overhead} \). Using the same scheme to transmit overhead as described
in 2.4.1,

\[ R_j = r_j + 1 + \frac{B(v) + E}{n_j^{(k)}} \]  

(2.10)

where \( v \) is the offset between the new codevector \( c_{\text{new}} \) and the original codevector \( c_j \), and \( n_j^{(k)} \) is the empirical occurrence count of \( c_j \) before splitting.

The average distortion will drop, but the extent to which it drops is not obvious. This depends on the geometry of the cell \( C_j \), and how \( c_{\text{new}} \) is placed relative to \( c_j \). A heuristic estimate is given by

\[ D_j = \kappa D(C_j, c_j) \]  

(2.11)

where \( \kappa < 1 \). In our experiments, \( 0.5 < \kappa < 0.75 \).

Hence,

\[ \Delta R = 1 + \frac{B(v) + E}{n_j^{(k)}} \]  

(2.12)

\[ \Delta D = (1 - \kappa)D(C_j, c_j) \]  

(2.13)

and \( c_j \) is a candidate for splitting if (2.5) holds.

After determining \( c_j \) to be a candidate for splitting, it is not actually split until a subsequent source vector \( x_i \) mapped to \( c_j \) is at a distance within a range from it, which is not too close nor too far. Heuristically, we have taken the range to be \( D(C_j, c_j) < d_s(x_i, c_j) < 4D(C_j, c_j) \). Then, a quantized version of \( x_i \) is treated as a new codevector and is transmitted in the bitstream. \( c_j \) is unchanged in the process.

### 2.4.3 Moving Codevectors

The empirical centroid \( \hat{c}_j^{(k)} \) is calculated for each codevector during \( \tau_k \). At the end of \( \tau_k \), decisions are made as to which \( c_j \) should be moved towards \( \hat{c}_j^{(k)} \). They do not necessarily move to exactly \( \hat{c}_j^{(k)} \), since codevector offsets are quantized before transmission.

Let \( \hat{c}_j \) be the quantized version of \( \hat{c}_j^{(k)} \), and let \( v \) be the offset \( \hat{c}_j - c_j \). Hence, \( B(v) \) bits are required to encode the offset. Moreover, since not all codevectors are always updated, the codevector indices must be specified also. This overhead is shared among \( n_j^{(k+1)} \) source vectors which will be mapped to \( \hat{c}_j \). At the moment the decision
is made, however, \( n_j^{(k+1)} \) is unknown. As an estimate, it can be taken to be \( n_j^{(k)} \).

Given these considerations, for each codevector \( c_j \),

\[
\Delta R = \frac{r_j + B(v)}{n_j^{(k)}}.
\]  

(2.14)

If \( \hat{c}_j \) is exactly the centroid of the source vectors mapped to \( c_j \) during \( \tau_k \), then the average reduction in distortion if \( \hat{c}_j \) were the codevector instead of \( c_j \) is simply \( d_e(c_j, \hat{c}_j) \). Then, for each \( c_j \),

\[
\Delta D = d_e(c_j, \hat{c}_j).
\]  

(2.15)

As mentioned earlier, \( \hat{c}_j \) may not be exactly the centroid. In that case, the equality in (2.15) no longer holds. In our experiments, however, (2.15) is still being used. In Section 3, we will show that the exact formulation of these decision equations is relatively unimportant.

As before, a codevector is updated if (2.5) holds.

2.4.4 Deleting Codevectors

In [4], codevectors whose cells are unpopulated after several iterations are effectively deleted from the codebook, since the entropy coder will assign an infinite length code to its index. For our purpose, it may be worthwhile to keep the codevectors around, even though they are unpopulated for a period of time, since adding them later on will cost considerable overhead. At the end of \( \tau_k \), codevectors with \( n_j^{(k)} = 0 \) will be assigned a count of 1 before updating the index codes for \( \tau_{j+1} \). A more detailed discussion on this will be given in Section 2.4.5.

If the codebook is growing too large, and memory is running low, then it is inevitable that unpopulated codevectors are deleted. In that case, some of the unpopulated codevectors are flushed out at the end of \( \tau_k \) for each \( k \). Note that codevector deletion is an adaptation that comes for free with the encoded bitstream, since the necessary information for deletion is just \( n_j^{(k)} \), which can be calculated at the decoder based on the stream of codevector indices sent by the encoder. No side information has to be transmitted, and hence no rate-distortion tradeoff is considered here.
2.4.5 Updating the Index Codes

As mentioned before, index codes are updated at the end of $\tau_k$ based on $n_j^{(k)}$, $j = 1, \ldots, M$, gathered during $\tau_k$. $\{n_1^{(k)}, \ldots, n_M^{(k)}\}$ gives an estimate of the local statistics of the source, hence an entropy coder that updates the lossless code based on $\{n_1^{(k)}/L, \ldots, n_M^{(k)}/L\}$ adapts to local statistics of the source. (Note that the update interval $L$ is equal to $\sum_{j=1}^M n_j^{(k)}$.) Since both the encoder and decoder have access to $n_j^{(k)}$, updates of the index codes do not have to be transmitted. Entropy codes such as Huffman code, Shannon-Fano-Elias code, or arithmetic code can be used for this purpose.

When estimating the distribution from $\{n_1^{(k)}, \ldots, n_M^{(k)}\}$, the zero frequency problem [1] arises: if $n_j^{(k)} = 0$ for some $j$, should we infer that the $j$-th cell is of zero probability, and assign an infinite length index code to it (or simply delete it)? Or should we assume that the set of samples is not of sufficient size? In Section 2.4.4, a way to circumvent the zero frequency problem is presented. The occurrence counts of unpopulated codevectors are set to one automatically, to avoid the codevectors from being virtually flushed out of the codebook when the index codes are updated. This, however, still poses a problem. In the algorithm, $x_i$ is encoded by the codevector that incurs the least $R + \lambda D$. The longer a codevector index code is, the less likely that codevector is going to be chosen. Hence, if the codebook size $M$ is large, setting $n_j^{(k)}$ to 1 does not save $c_j$ from being phased out. Moreover, if the update interval $L$ is not much larger than the codebook size $M$, the occurrence counts suffer from insufficient statistics. Losing codevectors just because they have not experienced much usage during $\tau_k$ should be avoided.

To overcome this, the codebook can be divided into two parts. The first part $C_{hot}$ contains the most popular codevectors. This is the hotlist of the codebook. The second part $C_{cold}$ contains the rest of the codebook. Let $n_{cold}^{(k)} = \sum_{(j|c_j \in C_{cold})} n_j^{(k)}$. Then the entropy coder will design the index codes for $\{n_j^{(k)}|c_j \in C_{hot}\} \cup \{n_{cold}^{(k)}\}$. The codevectors in $C_{cold}$ will then be distinguished by a fixed length code appended to $s_{cold}$ assigned by the entropy coder. In other words, $\{s_j|c_j \in C_{hot}\}$ are optimal variable length codes, whereas $\{s_j|c_j \in C_{cold}\}$ are fixed length codes (Fig. 2-6). This prevents
any particular index code from being extraordinarily longer than the others, and also reduces the amount of computation of the entropy coder.

![Figure 2-6: Modified entropy coding of the codevector indices. In this example, Huffman coding is used for codevectors in the hotlist plus the prefix of the rest of the codebook (\(s_{\text{cold}}\)). Fixed length codes then append \(s_{\text{cold}}\). \(C_{\text{hot}} = \{s_0, \ldots, s_{10}\}\), \(C_{\text{cold}} = \{s_{11}, \ldots, s_{18}\}\).]

### 2.5 Quantization Issues

When adding, splitting and moving codevectors, offsets are sent as overhead. Hence, there is a tradeoff in the precision with which the offsets are described, and the number of bits that they cost. For simplicity, independent scalar quantization is performed on each component of the offset. The scalar quantizer is pre-designed and fixed rate at the current stage of our work.

As the codebook adapts to a stationary source and converges to the optimal codebook, the size of the codevector offsets become smaller and smaller. Hence, the quantizer is designed to be adaptive to the dynamic range of offset powers. Given
a fixed rate of $b$ bits, $b_G$ bits are allocated to specify the gain, and the remaining $b_Q = b - b_G$ are allocated for the quantization levels of a uniform quantizer. Hence, $2^{b_Q}$ levels are available, and they are scaled by one of $2^{b_G}$ gain values. For a codevector offset, the gain is determined by its largest component. Hence, the total number of bits to quantize an offset of dimension $l$ is $b_G + lb_Q$. The gain varies from one codevector offset to another.

There are several alternatives to encoding (quantizing) the offsets:

1. Scalar quantization of each component with several scalar quantizers of different resolutions. The quantized version with the best rate-distortion tradeoff is selected, and a selector code that specifies that resolution is transmitted.

2. Vector quantization of the offset.

[13] has given a more detailed treatment of quantization strategies for this purpose. The interested reader is referred to it.

2.6 Syntax of the Encoded Bitstream

Encoding of the source is done in a manner similar to Lempel-Ziv coding, that for each $i$, $x_i$ is encoded either by the index code of an existing codevector, or by the description of a new codevector. An escape code is used to distinguish the two cases. Moreover, at the end of each $\tau_k$, whose size $L$ is predetermined, codevector updates are transmitted in a batch, trailed by the escape code. Hence, the encoded bitstream contains all the information the decoder needs to reconstruct the codebook and its dynamics.

The syntax of the bitstream is summarized as follows:

| {index} | {index} ... {index} ...    | ← plain index encoding |
| {esc}  | {index} | {offset} ... | ← new codevector |
| {index} | {index} ... {index}       | ← plain index encoding up to end of $\tau_k$ |
| {index} | {offset} ... {index} | {offset} | {esc} ... | ← codevector updates, trailed by {esc} |

where
\{index\} = index code of the "nearest" existing codevector,  
\{esc\} = escape code, and  
\{offset\} = offset between the new/updated and original codevectors.

2.7 Complexity of the Algorithm

Finally, a brief note on the complexity of the algorithm. For each source vector $x_i$ of dimension $\ell$, encoding it with an existing codevector involves $M$ distortion calculations $d_\ell(x_i, c_j)$, $j = 1, \ldots, M$, and finding the minimum $r_j + \lambda d_\ell(x_i, c_j)$. Hence, the per source symbol complexity is $O(M/\ell)$. At the same time, evaluating whether adding or splitting a codevector is worthwhile involves comparing $\Delta R$ with $\lambda \Delta D$, as described in Sections 2.4.1 and 2.4.2. Using fixed scalar quantization for the codevector offsets, this is an $O(1)$ per source symbol operation. Moreover, several book-keeping operations, such as incrementing occurrence counts, updating centroids and average distortions, are involved. These are also $O(1/\ell)$ operations, and hence the overall per source symbol complexity is $O(M/\ell)$ for large $M$.

Once every $L$ vectors, new index codes are computed, which can be computationally quite intensive for large $M$. With the modified entropy coding scheme described in Section 2.4.5, where the hotlist is kept reasonably small, this task becomes much less computationally intensive. Codevector deletion and batch codevector updates towards centroids are also performed, but they are relatively simple operations. In any case, for large $L$ (such as a few thousands source vectors in practice), these computational overheads become insignificant. Note that the choice of $L$ is a balance of accurate estimation of source statistics (large $L$) and adaptability to local variations of it (small $L$).

Decoding is basically a table look-up plus some book-keeping for codebook modifications. Index codes computation is the same as that at the encoder, which as mentioned earlier, becomes insignificant for large $L$. Hence, the algorithm is of the same order in complexity as static codebook full-search VQ in terms of source encoding and decoding. This is more attractive than two-pass adaptive algorithms, which require more computation per source vector due to their iterative nature.
With the concern for lower encoding complexity, tree-structured VQ (TSVQ) [9] provides an effective way to reduce the search complexity to $O(\log M/\ell)$ per source symbol. A one pass adaptive version of TSVQ would therefore be highly desirable. Work in this direction will be pursued.
Chapter 3

Experimental Results

In this chapter, the performance of the RDLZ algorithm described in Chapter 2 on various sources is evaluated. Sources that are stationary or time-varying, synthetic or real, are considered. In particular, we are concerned with two issues: (1) the performance gain achieved by adaptation for sources that are mismatched with the training sequence on which the codebook is designed, as well as nonstationary sources, and (2) the performance penalty caused by adaptation for stationary sources with optimal initial codebook. Moreover, effects of the choice of parameters such as $L$ (update interval) and $\lambda$ will be discussed.

3.1 Performance Gain with Adaptation

In the first experiment, the adaptive algorithm is applied on a stationary Gaussian source mismatched with the initial codebook. The distribution of the source and the initial codebook are shown in Fig. 3-1. The length of the source is 160000 samples. Vector dimension is 4, but only the first two components are plotted on the Cartesian plane for easy visualization. Update interval $L$ is 2000 vectors. Fig. 3-2 shows the results of the adaptive algorithm initialized with a mismatched codebook, as compared to static codebook vector quantization. The curve is obtained by varying $\lambda$ and initial codebook size, and then finding the convex hull. Significant improvements are achieved with adaptation. When compared to the performance of the optimal vector
quantizer, however, the adaptive algorithm is suboptimal. This is due to the static nature of $L$, which will be discussed in Section 3.2.

![Source distribution](image)

![Mismatched codebook](image)

Figure 3-1: Source mismatched with the initial codebook. (a) Distribution of the source. Signal source is correlated Gaussian source with $\rho = 0.95$, $\sigma^2 = 1$. (b) Initial codebook designed on training sequence from an i.i.d. Gaussian source with $\sigma^2 = 0.25$.

Next, a time-varying source is considered. It is a source which is switching back and forth between two stationary Gaussian processes at random times. This can still be a stationary source if the switching process is stationary, but over a small window of interest, the source exhibits non-stationary behavior. Fig. 3-3a shows the source distribution at two different time instants $t_1$ and $t_2$. We start with an initial codebook which is the optimal for the “average” of the two processes. Results (Fig. 3-4) again show the advantages of adaptation, although they are quite marginal as compared to the previous case. Fig. 3-3b plots the codevectors with the shortest index codes at time instants $t_1$ and $t_2$. Clearly, the codebook (the codevector index codes, to be precise) is adapting to changes in the source distribution.

### 3.2 Universality of the Algorithm

From Fig. 3-2, it appears that although the adaptive algorithm achieves significant improvements over static codebook VQ with mismatched codebook, it is still a
step away from the performance of the optimal VQ codebook. This is due to the fact that $L$ remains constant throughout. As codevectors are moved towards their optimal locations, offsets of codevector updates become smaller and smaller. At some point, the updates are deemed unworthy in the rate-distortion tradeoff by the algorithm, and adaptation ceases. For stationary sources, this can be fixed by expanding the update window as offsets to empirical centroids $c_j^{(t)}$ decrease. Updating becomes cheaper, since the bits spent are spread among more vectors.

The experiment on stationary source with mismatched initial codebook is repeated with some modifications. At the end of $\tau_k$, if the number of codevectors that are updated towards $c_j^{(t)}$ is less than 2, then $L$ will be doubled for $\tau_{k+1}$. Fig. 3-5 shows the convergence of the modified adaptive algorithm to the optimal VQ performance. This gives an empirical evidence of the universality of the algorithm.
3.3 Performance Penalty on Stationary Sources

Given stationary sources, it is expected that adaptation actually hurts the performance, since bits may be unnecessarily spent for marginal reduction in average distortion. In our experiment, we start with the optimal vector quantizer for a stationary memoryless (i.i.d.) Gaussian source. Since the initial codebook is optimal for that particular source, any loss in performance is due to adaptation. The vector dimension is 4, which in this case is irrelevant due to the memoryless nature of the source. There is one trickiness in performing the experiment though, which is the
freedom in choosing $\lambda$, a parameter that controls the degree of adaptation. From Fig. 2-1, we know that $\lambda$ should be chosen as the negative of the slope of the tangent to the operating point on the rate-distortion curve. Hence, a smaller $|\lambda|$ should be chosen at a lower bit rate. This is taken into consideration in the experiment. Fig. 3-6 shows that performance penalty due to adaptation is minimal.

### 3.4 Results on Images

We then compress test images using the adaptive algorithm. Initial codebooks are trained on the image “Barbara” (Fig. 3-7), of size $512 \times 512$. In these experiments, the vector dimension is $4 \times 4$. The “Barbara” codebooks are then used to compress the test image “Ariel”, also of size $512 \times 512$. Result of RDLZ encoding at 0.5 bpp is shown in Fig. 3-8. Fig. 3-9 compares the performance of RDLZ and static codebook VQ at different bit rates.
Figure 3-5: Universality of RDLZ. The source and initial codebook are identical to that in Fig. 3-1. Results on 160k and 800k samples from the source are shown.

A more illustrative example is when the test image is a composite of subimages with drastically different characteristics. Here, the image “Barbara” is appended with a piece of textual image (Fig. 3-10). Using an initial codebook of size 512 trained on “Barbara”, static codebook VQ works very poorly on the textual part, where the text is illegible (Fig. 3-11). On the other hand, RDLZ adapts to the image and modifies the codebook accordingly, giving much more acceptable results (Fig. 3-12).

By varying \( \lambda \) on initial codebooks (trained on “Barbara”) of different size, a set of operational rate-distortion curves are obtained. The convex hull of these curves gives the rate-distortion performance of RDLZ. Fig. 3-13 compares the rate-distortion performance of RDLZ and static codebook VQ in encoding the composite image. As a remark, for most \( \lambda \), a smaller initial codebook gives a rate-distortion performance closer to that of the vector quantizer trained on the composite image itself, but also gives poorer visual quality in the “Barbara” part of the image. This is because with a smaller initial codebook, i.e. shorter average index code length, more bits can be spent on adapting to the textual part of the image, resulting in a better “average” codebook.
Figure 3-6: Performance penalty due to adaptation on an i.i.d. Gaussian source. —— is the performance of the optimal vector quantizer, × are operating points of RDLZ with initial codebooks of different size.

Note that the VQ codebook trained on the composite image is the optimal "average" codebook. With less codevectors optimized for "Barbara", the visual quality in that part is thus affected.
Figure 3-7: “Barbara”

Figure 3-8: (a) Original “Ariel”. (b) “Ariel” RDLZ encoded at 0.5 bpp, PSNR = 23.5 dB.
Figure 3-9: Comparison of RDLZ and static codebook VQ on “Ariel”. Vector dimension is $4 \times 4$. Codebooks trained on “Barbara”.
Since we are considering mobile channels, we need to take care of fading. Assuming an urban cellular environment, where the communication between the cell site and mobile is not only by direct path (sometimes the direct path is not available), but also via many paths. Under these assumptions, each signal path undergoes: i) an attenuation due to path loss obeying the fourth-power law, ii) a log-normal slow shadowing due to terrain features, iii) Rayleigh fading due to multipath with different Doppler shifts. First consider the case where the delay spread of the different paths is small (narrow-band model). A conceptual model of this fading channel under the narrow-band model is as follows:

The transmitted signal is $u(t)$. The channel output is $v(t)$. $A$ is the attenuation on the dominant path, which obeys the fourth-power law. $\tau$ is the propagation delay. $\beta(t)$ is a random process with a log-normal distribution, with a standard deviation of 4 dB for typical urban areas.

Figure 3-10: Original composite image
Figure 3-11: Composite image coded with static codebook VQ at 0.43 bpb.
Figure 3-12: Composite image coded with RDLZ at 0.43 bbp.
Figure 3-13: Comparison of RDLZ and static codebook VQ on composite image. Vector dimension is $4 \times 4$. Initial codebook trained on "Barbara". $-$ and $-$ is the performance of static codebook VQ, $\cdots \cdots$ is the operational rate-distortion curves of RDLZ with the labelled initial codebook size, $-$ is the convex hull of these curves. As a comparison, $-$ is the performance of static codebook VQ with codebook trained on the composite image.
Chapter 4

Conclusion and Future Work

We have presented an adaptive lossy compression algorithm which is one pass and computationally efficient. A lossy version of Lempel-Ziv coding is formulated as a one pass adaptive vector quantization technique. The codebook is modified on line as encoding of the source signal proceeds, and updates are transmitted as side information, so that the decoder can reconstruct the dynamics of the codebook. Codebook adaptation is based on the principal adaptation criterion (2.5), which is to minimize the Lagrangian $J = R + \lambda D$. Since the algorithm is one pass, evaluation of the adaptation criterion involves causal learning and estimating of the source distribution.

Experimental results demonstrate the merits of the adaptive algorithm, in that it improves rate-distortion performance over static codebook VQ in practical situations where the codebook is trained on something other than the source signal. For some cases, improvement is marginal, especially if the initial codebook is reasonably matched to the source distribution. However, in certain cases, improvement can be substantial, as shown in Sections 3.1 – 3.4. Moreover, with slight modifications, the algorithm is shown to converge to the optimal VQ performance on stationary source with mismatched initial codebook.

There are some issues that require further work. The choice of $\lambda$ and update interval $L$ is currently determined more or less by trial and error. An automatic, or learning mechanism to determine these parameters would be advantageous.
As mentioned in Section 2.7, tree-structured codebook provides a way to reduce the search complexity for encoding. In full search VQ, to encode a source vector, the whole codebook has to be searched to find the best codevector. In tree-structured VQ, the search is performed in stages. For an $m$-ary tree-structured codebook, the source vector is compared with at most $m$ codevectors at each stage. This gives an order of $O(\log M/\ell)$ instead of $O(M/\ell)$ per source symbol, which can be a significant reduction in search complexity for large codebook size $M$. Hence, a one pass adaptive tree-structured VQ algorithm would be an interesting and worthwhile entity to pursue. The structure of the codebook, however, poses new complications to the problem. Some previous works of this flavor are [2, 3], which reorganize the structure of the codebook subtree selected from a large pre-designed fixed complete tree-structured codebook at an update interval. These works do not involve sending of side information.

Another direction of future work is a vector quantization scheme with variable vector dimensions. This is even closer in essence to Lempel-Ziv coding, where the dictionary phrases are of variable lengths. Design of the initial codebook, encoding of source signals by the variable dimensional codevectors, adaptability of the codebook, and considerations on visual qualities when applied to image coding are intriguing problems to investigate.
Bibliography


