Fast and Efficient Algorithms for text and Video Compression

Dzung Tien Hoang
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-97-06
May 1997

Fast and Efficient Algorithms for Text and Video Compression[1]

Dzung Tien Hoang[2]
Department of Computer Science
Brown University
Providence, RI 02912–1910

Abstract

There is a tradeoff between the speed of a data compressor and the level of compression it can achieve. Improving compression generally requires more computation; and improving speed generally sacrifices compression. In this thesis, we examine a range of tradeoffs for text and video.

In text compression, we attempt to bridge the gap between statistical techniques, which exhibit a greater amount of compression but are computationally intensive, and dictionary-based techniques, which give less compression but run faster. We combine the context modeling of statistical coding with dynamic dictionaries into a hybrid coding scheme we call *Dictionary by Partial Matching*.

In low-bit-rate video compression, we explore the speed-compression tradeoffs with a range of motion estimation techniques operating within the H.261 video coding standard. We initially consider algorithms that explicitly minimizes bit rate and combination of rate and distortion. With insights gained from the explicit minimization algorithms, we propose a new technique for motion estimation that minimizes an efficiently computed heuristic function. The new technique gives compression efficiency comparable to the explicit-minimization algorithms while running much faster. We also explore bit-minimization in a non-standard quadtree-based video coder that codes motion information hierarchically using variable-sized blocks.

For video coding at medium-to-high bit rates, we propose a framework that casts rate control as a resource allocation problem with continuous variables, non-linear constraints, and a novel lexicographic optimality criterion motivated for near-constant-quality video. With this framework, we redefine the concept of efficiency to better reflect the constancy in quality generally desired from a video coder. Rigorous analysis within this framework reveals elegant conditions for optimality, which leads to polynomial-time algorithms. Simulation studies confirm the theoretical analysis and produce encodings that are more constant in quality than that achieved with existing rate control methods. As evidence of the flexibility of the framework, we show how to extend it to allocate bits among multiple variable-bit-rate bitstreams for transmission over a constant-bit-rate channel.

---

[1] A similar version of this report was submitted in fulfillment of the dissertation requirement for Dzung Tien Hoang's Ph.D.

[2] Support was provided in part by a National Science Foundation Graduate Fellowship; by Air Force Office of Strategic Research grants F49620–92–J–0515 and F49620–94–I–0217; and by Army Research Office grant DAAH04–93–G–0076.

Fast and Efficient Algorithms for Text and Video Compression

by

Dzung Tien Hoang

B.S. Computer Science, Tulane University, 1990

B.S. Electrical Engineering, Tulane University, 1990

Sc.M. Computer Science, Brown University, 1992

Thesis

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in the Department of Computer Science

at Brown University.

May 1997

Vita

Dzung Tien Hoang was born on April 20, 1968 in Nha Trang, Vietnam. He immigrated to the United States of America in 1975 with his parents, Dzuyet D. Hoang and Tien T. Tran, and two sisters. He now has three sisters and one brother. They have been living in Harvey, Louisiana.

After graduating in 1986 from the Louisiana School for Math, Science and the Arts, a public residential high school in Natchitoches, Louisiana, he attended Tulane University in New Orleans with a full-tuition Dean's Honor Scholarship and graduated in 1990 with Bachelor of Science degrees in Electrical Engineering and Computer Science, both with Summa Cum Laude honors.

He joined the Computer Science Department at Brown University in Providence, Rhode Island, in 1990 under a University Fellowship and later under a National Science Foundation Graduate Fellowship. He received a Master of Science in Computer Science from Brown University in 1992.

From 1993 to 1996, he was a visiting scholar and a research assistant at Duke University in Durham, North Carolina. From 1991 to 1995, he spent summers working at the Frederick National Cancer Research Facility, the Supercomputing Research Center, and the IBM T. J. Watson Research Center.

In August 1996, he joined Digital Video Systems, in Santa Clara, California, as a Senior Software Engineer.

Preface

In today's information-driven society, text, audio, image, video and other forms of information are being increasingly generated, manipulated, and transmitted in digital form. This trend is manifested in the increased level of automation in businesses, the ubiquity of personal computers, the explosive growth of the Internet and the World Wide Web, and the growing library of multimedia software that incorporates digital audio, images, and video. Within the past decade, we have seen secondary storage on desktop personal computers mushroom from a mere 20 MB to 2 GB and beyond. Modem technology has pushed the transmission bandwidth through plain telephone lines from 300 bits/s to 33.6 kbits/s. Even with technological improvements in transmission bandwidth and storage capacity, the information explosion is quickly making current technologies seem inadequate. For example, application software that once fit onto a few floppy disks now demands multi-megabytes of disk space.

Crucial to the management of digital information are data compression techniques that help make more efficient use of the limited transmission and storage resources available. For storage applications, data compression increases the effective storage space, allowing more data to be stored on a given storage device. For transmission applications, data compression increases the effective bandwidth, allowing a higher volume of data to be transmitted over a given transmission medium. Data compression can be viewed as a logical transformation of the data and is independent of the underlying transmission or storage technology. Data compression will not be made obsolete by advances in these technologies, as there will always be a need for even more storage and even greater bandwidth.

Acknowledgements

This dissertation is the culmination of seven years of graduate studies and four years of undergraduate education. I am grateful to Johnette Hassell, Mark Benard, and Boumediene Belkhouche from Tulane University for encouraging me to pursue a graduate degree in Computer Science.

At Brown University, my research career began on firm ground under the direction of Daniel Lopresti, who guided me to my first conference publication and my first summer research position. I thank John Savage of Brown University for his patience and support during my search for an academic focus.

Under the steady guidance of Jeff Vitter I have learned much about research and scholarship. Jeff is a demanding advisor, sparing in approval or praise. A note of "Good Work" and a few sparse markings on a recent draft research paper was sure sign of my progress. I am proud to have earned Jeff's trust.

During my residence at Brown University and Duke University, my life has been enriched by interaction with fellow students, especially T. Murali (tmax), Swarup Acharya, Dimitrios Michailidis, Darren Vengroff, P. Krishnan (pk), Choh-Man Teng, Thomas Alexander, Min Wang, Hongyan Wang, and Apostol Natsev. Much of my work at Duke was catalyzed by Phil Long who provided a patient ear and injected much enthusiasm during our many brainstorming sessions.

I would like to thank Morten Schultz and Bruce Shapiro from the Frederick National Cancer Research Facility; Duncan Buell, Maya Gokhale, and Jeff Arnold from the Center for Computing Sciences; and Prasoon Tiwari, Elliot Linzer, and Cesar Gonzales from the IBM T. J. Watson Research Center for making my summers most stimulating and rewarding.

Research is often a collaborative effort. The work described in Chapters 2, 4, and 5 was performed jointly with Phil Long and Jeff Vitter. Chapters 6 to 8 resulted from joint work with Elliot Linzer and Jeff Vitter. Chapter 10 was joint work with Jeff Vitter.

I am indebted to the National Science Foundation for supporting me with an NSF Graduate Fellowship. This award has afforded me the freedom to pursue a research career in the field of my choice.

I would be quite lost without the skills of the administrative staff at the Departments of Computer Science at Brown University and Duke University. Special tribute is due to Mary Andrade at Brown and Cathy Caimano at Duke for their tireless administrative assistance.

I appreciate the kindness of Digital Video Systems for giving me the time and resources to finish writing my dissertation.

Above all, I thank my parents, my brother, and my sisters for their constant love and support.

Contents

List of Tables

List of Figures

Introduction

In this thesis, we restrict our attention to the compression of *digital data* that consist of a sequence of symbols chosen from a *finite* alphabet. In order for data compression to be meaningful, we assume that there is a standard representation for the uncompressed data that codes each symbol using the same number of bits. For example, text documents are commonly represented as sequences of characters (alphanumeric and control symbols) that are encoded in ASCII format using eight bits. Digital images are composed of pixels, which are typically represented using a binary code of a fixed length. Compression is achieved when the data can be represented with an average length per symbol that is less than that of the standard representation.

Not all forms of information are digital in nature. For example, audio, image, and video exist at some point as waveforms that are continuous both in amplitude and in time. Information of this kind is referred to as *analog* signals. In order to be representable in the digital domain, analog signals must be discretized in both time and amplitude. This process is referred to as *digital sampling*. Digitally sampled data is therefore only an approximation of the original analog signal.

Data compression methods can be classified into two broad categories: *lossless* and *lossy*. As its name suggests, in lossless coding, information is preserved by the compression and subsequent decompression operations. The types of data that are typically compressed losslessly include natural language texts, database files, sensitive medical images, scientific data, and binary executables. Of course, lossless compression techniques can be applied to any type of digital data; however, there is no guarantee that compression will actually be achieved for all cases. Although digitally sampled analog data is inherently lossy, no additional loss is incurred when lossless compression is applied.

On the other hand, lossy coding does not preserve information. In lossy coding, the amount of compression is typically variable and is dependent on the amount of loss that can be tolerated. Lossy coding is typically applied to digitally sampled data or other types of data where some amount of loss can be tolerated. The amount of loss that can be tolerated is dependent to the type of data being compressed, and quantifying tolerable loss is an important research area in itself.

By accepting a modest amount of loss, a much higher level of compression can be achieved with lossy methods than lossless ones. For example, a digital color image can typically be compressed losslessly by a factor of roughly two to four. Lossy techniques can compress the same image by a factor of 20 to 40, with little or no noticeable distortion. For less critical applications, the amount of compression can be increased even further by accepting a higher level of distortion.

To stress the importance of data compression, it should be noted that some applications would not be realizable without data compression. For example, a two-hour movie would require about

149 gigabytes to be stored digitally without compression. The proposed Digital Video Disk (DVD) technology would store the same movie in compressed form using only 4.7 gigabytes on a single-sided optical disk. The efficacy of DVD, therefore, relies on the technology to compress digital video and associated audio with a compression ratio of about 32:1, while still delivering satisfactory fidelity.

A basic idea in data compression is that most information sources of practical interest are not random, but possess some structure. Recognizing and exploiting this structure is a major theme in data compression. The amount of compression that is achievable depends on the amount of redundancy or structure present in the data that can be recognized and exploited. For example, by noting that certain letters or words in English texts appear more frequently than others, we can represent them using fewer bits than the less frequently occurring letters or words. This is exactly the idea behind Morse Code, which represents letters using a varying number of dots and dashes. The recognition and exploitation of statistical properties of a data source are ideas that form the basis for much of lossless data compression, which is the subject of Chapters 1 and 2.

In lossy coding, there is a direct relationship between the length of an encoding and the amount of loss, or distortion, that is incurred. Redundancy exists when an information source exhibits properties that allow it to be coded with fewer bits with little or no perceived distortion. For example, in coding speech, distortion in high frequency bands is not as perceptible as that in lower frequency bands. As a result, the high frequency bands can be coded with less precision using fewer bits. The nature of redundancy for lossy coding, especially as it relates to video coding, is the subject of Section 3.3.

Contributions of the Thesis

In data compression, there is a natural tradeoff between the speed of a compressor and the level of compression that it can achieve. In order to achieve greater compression, we generally require more complex and time-consuming algorithms. In this thesis, we examine a range of operational points within the tradeoff possibilities.

Text Compression

In text compression, statistical coding techniques exhibit state-of-the-art compression, but are computationally intensive. Dictionary-based methods, on the other hand, give less compression while running faster. In Chapter 2, we attempt to bridge the gap between statistical and dictionary-based techniques. We combine the context modeling of statistical coding with dynamic dictionaries into a hybrid coding scheme that we call *Dictionary by Partial Matching* (DPM).

We investigate the DPM technique with three dictionary-based methods and observe varying levels of improvement. The amount of improvement appears to be related to how fast the dictionary is adapted. Generally speaking, a dictionary coder that grows its dictionary quickly is likely to get a smaller improvement with DPM than a coder that is more selective in adapting its dictionary.

Motion Estimation at Low Bit Rates

In Chapter 4, we explore the speed-compression tradeoffs possible with a range of motion estimation techniques operating within a low-bit-rate video coder that adheres to the H.261 international standard for video coding. At very low rates, hybrid video coders that employ motion compensation in conjunction with transform coding of the residual typically spends a significant portion of the bandwidth to code the motion information. We focus on motion estimation with hopes of improving the compression performance.

Initially, we construct motion estimation algorithms that explicitly minimizes bit rate and combination of rate and distortion. In coding experiments, these computationally intensive algorithms produce better compression (with comparable quality) compared to the standard motion estimation algorithm, which does not require as much computation. Based on insights gained from the explicit minimization algorithms, we propose a new technique for motion estimation that minimizes a quickly computed heuristic function of rate and distortion. The new technique gives compression efficiency comparable to the computationally intensive explicit-minimization algorithms while running almost as fast as the standard algorithm.

In Chapter 5, the bit-minimization philosophy is further applied to a non-standard quadtree-based video coder that codes motion information hierarchically using variable-sized blocks. The motivation is to explore the limits of bit-minimization when applied to an efficient scheme for coding motion vectors. By designing the quadtree encoding so that a subtree is coded independently of other disjoint subtrees, we are able to compute motion vectors that globally minimize the total bit rate using a dynamic programming algorithm. Experimental results confirm that the quadtree-based coder gives additional gains over the H.261-based coders.

Optimal Rate Control

In Chapters 6 through 10, we focus our attention on optimal rate control algorithms for video coders. Existing optimal rate control techniques typically regulate the coding rate to minimize a sum-distortion measure. While these techniques can leverage the wealth of tools from least-mean-square optimization theory, they do not guarantee constant-quality video, an objective often mentioned in the literature. We propose a framework that casts rate control as a resource allocation problem with continuous variables, non-linear constraints, and a novel lexicographic optimality criterion that is motivated for uniform video quality. With this framework, we redefine the concept of coding efficiency to better reflect the constancy in quality that is generally desired from a video coder.

Rigorous analysis within this framework reveals a set of necessary and sufficient conditions for optimality for coding at both constant and variable bit rates. With these conditions, we are able to construct polynomial-time algorithms for optimal rate control. Experimental implementations of these algorithms confirm the theoretical analysis and produce encodings that are more uniform in quality than that achieved with existing rate control methods. As evidence of the generality and flexibility of the framework, we show how to extend the framework to allocate bits among multiple variable-bit-rate bitstreams that are to be transmitted over a common constant-bit-rate channel.

Part I

Text Compression

Chapter 1

Introduction to Text Compression

There are two main classes of text compression methods: statistical and dictionary-based. Generally speaking, statistical methods process the input one character at a time and capture redundancy using probabilistic models. The probabilistic models predict the current character to be coded based on previously coded characters. The prediction is used in a process called *entropy coding* that codes the identity of the current character efficiently using a variable number of bits. Dictionary (or substitution) methods, on the other hand, use a dictionary of stored phrases (sequences of characters) and substitute substrings in the input text that match some dictionary entry by pointers (or index) into the dictionary. In this way, a dictionary coder is able to process multiple characters in each encoding step.

The effectiveness of statistical methods depends upon the accuracy of the probabilistic models and the efficiency of the entropy coder. Since sophisticated modeling and efficient entropy coding techniques exist today, statistical coding can give state-of-the-art compression. However, these sophisticated techniques are computationally expensive.

Dictionary methods, on the other hand, are inherently faster since several characters can be coded in each encoding step. Also, efficient dictionary data structures and search algorithms are available that allow for the implementation of very fast encoding and decoding systems. In fact, many real-time lossless data compression applications use some form of dictionary coding. However, since dictionary coders do not provide for as flexible data modeling, they do not give as much compression as statistical coders.

1.1 Statistical Coding

A basic idea in data compression is that most information sources of practical interest are not random, but possess some structure. Shannon recognized and formalized this idea in his seminal work on information theory [96], in which he shows how to code a source described by a statistical model arbitrarily close to its theoretical limit, which he defined to be the *entropy* of the source. Since then, much work has been done on statistical source modeling and efficient entropy coding techniques. The logical separation of statistical compression into source modeling and entropy coding has been proposed by Rissanen and Langdon [93]. We adopt this separation and discuss modeling and entropy coding separately.

1.1.1 Probability Modeling

One way to formally characterize the structure of an information source is with a probabilistic model, where a probability of occurrence is assigned to each symbol at each position in the text.[1] There are many ways in which a probabilistic model can be constructed. The simplest is to assume that every symbol is equally probable. In this case, each symbol can be coded with the same number of bits using a binary code.[2] If we assume that the probability of each symbol occurring is independent of position and of the surrounding text, we have a zero-order model. We can also consider models in which the probability of the current symbol being coded depends upon the identity of one previous symbol (first-order model), two previous symbols (second-order model), and so on.[3]

1.1.2 Entropy Coding

Given a probability model of an information source, how do we code instances of the source with the fewest number of bits per symbol on average? This question was answered by Shannon in his seminal paper on information theory [96]. Shannon defined a quantity called *entropy* that is a measure of the information content of a source. For a zero-order information source with an alphabet of size N and symbol probabilities $P = \{p_1, p_2, \ldots, p_N\}$, the entropy $H(P)$ of the source is defined to be:

$$H(P) = -\sum_{i=1}^{N} p_i \log_2 p_i.$$

The entropy can be interpreted as a *lower bound* on the average number of bits per symbol that can be used to encode the source. A source with high entropy cannot be compressed as well as one with lower entropy.

Early entropy coders, as exemplified by Huffman [44] coding and Shannon-Fano [26] coding, assign close to $\lceil -\log_2 p_i \rceil$ bits to code the ith symbol in an attempt to code a source near its entropy limit. Since these approaches code each symbol with an integral number of bits, the entropy limit cannot be reached exactly in general. In fact, if the probability of the most probably symbol is close to 1, Huffman coding can waste as much as 1 bit per symbol on average [2].

1.1.2a Arithmetic Codes

Addressing this limitation, Shannon showed that, by grouping (blocking) characters and coding them together, we can code a source arbitrarily close to its entropy. This approach is taken to

[1] In a compression system, the encoder and decoder share the same probability model so that the encoded message can be decoded exactly. The model may change dynamically, but the decoder's model always matches the encoder's.

[2] This is strictly true only if the size, N, of the alphabet is a power of two. Otherwise, one can use $\lfloor \log_2 N \rfloor$ bits for some symbols and $\lceil \log_2 N \rceil$ bits for others.

[3] There is some disagreement in the literature about how to number the order of a model. Some authors prefer to count the symbol to be predicted as well as the symbols used for prediction, so that their numbering is one more than ours.

the limit in *arithmetic coding*, where the entire input text is coded as a unit. The result is that arithmetic coding is able code arbitrarily close to the entropy limit, assuming perfect arithmetic.

Practical implementations of arithmetic coding exist that operate using fixed-precision arithmetic with little loss in compression efficiency [41, 82]. We will not discuss arithmetic coding in further detail. Since efficient solutions exist for entropy coding, the remaining avenue for improving statistical coding is with probability modeling.

1.1.2b Binary and Phased-In Codes

A source with uniform probability distribution does not benefit much from arithmetic coding. Instead, simpler and faster coding systems can often be used with little loss in compression. The simplest system is *binary coding*, where a fixed number of bits is used to represent each symbol: if N is the size of the alphabet, then one can simply use $\lceil \log_2 N \rceil$ bits for each symbol. If N is a power of 2, then all patterns of $\log_2 N$ bits are assigned to code a symbol. If N is not a power of 2, however, some binary patterns are unused. In this case, some symbols can be coded with one fewer bit than others. Let $k = \lceil \log_2 N \rceil$. Then $2^k - N$ symbols can be coded with $k-1$ bits and the rest with k bits. There are several ways to construct a code with the above properties. In fact, any Huffman code constructed from the uniform distribution will do. In Appendix A of [2], an efficient scheme is given and is called a *phased-in code*.

1.1.3 Context Modeling

While a zero-order model is easy to describe and code, many sources of practical interest can be better described with higher-order models. For example, the zero-order entropy of English texts has been estimated to be around 4.0 bits per symbol, while the limit using higher-order models (in experiments using human subjects) seems to be between 1.3 to 1.7 [97, 51, 2]. This suggests that higher-order models are needed if we wish to code English texts well. Other natural language texts also exhibit high-order properties.

In a general order-n model, the probability distribution of the current character depends upon an arbitrary (but fixed) set of n previously coded characters. This set is called the *context* of the model. There are many possible ways in which a context can be defined, especially if many more than n characters have already been coded. Since text is an inherently one-dimensional sequence of characters, intuition suggests that a good choice for the context is the sequence of n characters *immediately preceding* the current character. This choice for the context classifies the model as a *Markov* model. One can also formulate models in which the probability distribution for the current character also depends upon past coding events. Models of this kind belong to the class of *finite-state* models. Markov models that do not require memory of past actions are called *finite-context* models. In a finite-context model of order 3, therefore, the current character only depends upon the three preceding characters. Finite-context models are more popular than finite-state models for data compression since they are simpler and can model real-world data sufficiently well.

A major obstacles facing the use of high-order context models is that, as the order increases, the number of possible contexts increases exponentially. For example, with a binary alphabet, there can be 2^n order-n contexts requiring as many probability distributions. Therefore, the problem of estimating the probability distribution becomes intractable as the order increases.

1.1.4 Static, Semi-Adaptive, and Adaptive Models

If we assume that the probability model is known and fixed for a given source, we have a *static* model. Creating good static models requires extensive knowledge of the input source, knowledge which can be difficult to acquire in practice. Also, a static model tuned for a particular source, say English texts, may perform poorly for a different source, say Spanish texts.

A model can also be constructed by analyzing a message and creating a model that best encodes the message. Since the decoder needs to know the model before it can decode the message, the model needs to be transmitted. This scheme is called *semi-adaptive* modeling. Besides having to transmit the model explicitly, another drawback to semi-adaptive modeling is that the input message needs to be read twice, once to construct the model and again to encode it.

An alternative to static and semi-adaptive modeling is *adaptive* modeling. With adaptive modeling, the encoder and decoder initially start with the same model. As characters are encoded (respectively, decoded), the encoder (decoder) modifies the model based on the characters that have been encoded (decoded). This scheme works as long as the encoder and decoder make the same modifications at each step. A common approach is to base the model on frequency counts of already encoded text. Adaptive models have been shown in the literature to perform well compared to static and semi-adaptive models [19, 2].

1.1.5 Prediction by Partial Matching

One of the more popular and successful adaptive compression algorithms based on finite-context models is the Prediction by Partial Matching (PPM) algorithm of Cleary and Witten [20]. As discussed earlier, one of the difficulties of using a high-order context model is the large number of possible contexts. An adaptive compression scheme would have to process many symbols before it can construct a reasonably accurate high-order context model. Instead of waiting to gather enough statistics before using a high-order context, PPM combines the prediction of finite-context models of varying order, up to a maximum order o, in a process which we now describe.

At a high level, with $o = 3$, the PPM algorithm works as follows. For each sequence σ of three characters, the algorithm maintains counts of the number of times each character c occurred when the previous three characters were σ. The same is done for each sequence of two characters, and one character. There is also a zero-order context that keeps frequency counts for each character. When the encoder is at a particular position in the file encoding a given character c, it looks at the previous three characters, and encodes the current character using arithmetic coding with a probability estimate given by the fraction of time c has occurred given that the context was the previous three characters. If c has not occurred previously in the given three character context, the encoder sends a special *escape* symbol (whose probability estimation technique is a parameter of the basic algorithm), and recurses to using a two character context. If it turns out that c has not occurred in any context, even the zero-order context, then it is coded assuming a uniform acaabbacbcabcbbca|acbcabccbabcba...

(a) Input text

symbol	count	Pr
a	0	0
b	1	$\frac{1}{2}$
c	0	0
⟨Esc⟩	1	$\frac{1}{2}$

(b) Context bca

symbol	count	Pr
a	1	$\frac{1}{3}$
b	1	$\frac{1}{3}$
c	0	0
⟨Esc⟩	1	$\frac{1}{3}$

(c) Context ca

symbol	count	Pr
a	1	$\frac{1}{6}$
b	2	$\frac{1}{3}$
c	2	$\frac{1}{3}$
⟨Esc⟩	1	$\frac{1}{6}$

(d) Context a

Figure 1.1: Example of context models created by the PPM method. A portion of the input text is show in (a). The current coding position is indicated by the vertical bar, and the current contexts are underlined. Frequency tables for the current contexts are shown in (b), (c), and (d).

probability distribution.

As an example, consider the context models and input in Figure 1.1 for a three-symbol alphabet with the maximum order $o = 3$. The current first, second, and third-order contexts are a, ca, and bca, respectively. The table for context bca keeps count of the number of times that each symbol follows bca in the previous text. In this example, the escape symbol gets a fixed count of 1. Similar counts are shown for the contexts ca and a. The current character, a, is coded by first consulting the table for context bca. Since a has not yet occurred previously in this context, an escape symbol is sent with a probability value of $\frac{1}{2}$ (using $\log_2 2 = 1$ bit). The escape tells the decoder to shift to the order-2 context. Since a has previously occurred in the order-2 context, the encoder transmits the code for a with probability value of $\frac{1}{3}$ (using $\log_2 3$ bits). The counts in the tables are updated to reflect the occurrence of a, and coding proceeds with contexts caa, aa, and a.

The estimation of the probability of the escape symbol is an important parameter to the PPM algorithm. Several techniques for assigning a probability to the escape symbol in a PPM-style coder have been proposed and the resulting coders labeled PPMA, PPMB, PPMC, PPMD, PPMP, and PPMX [2, 20, 72, 111]. There is no theoretical basis for the preference of any method over another. Therefore, the only basis for comparison is empirical. This problem of assigning a probability to a novel event is known in the literature as the *zero-frequency* problem, with roots in philosophy as detailed in [2].

In [2, 20], experiments show that PPM works best for typical text files with a maximum order o of about 4. For non-text files, PPM peaks when o is 1 or 2.

1.2 Dictionary Coding

While statistical coders can give excellent compression with sophisticated modeling and entropy coding techniques, they are limited in speed since they code one character at a time. Dictionary coders, on the other hand, can code several characters at a time. They work by dividing the input into *phrases* in a process called *parsing*. The input text is usually parsed from left to right. In parsing, the text starting at the current coding position is searched against a dictionary of stored phrases. If a phrase in the dictionary matches a prefix of the text starting at the current position, the identity of that phrase is encoded, usually using a fixed number of bits. The index of the matching phrase is commonly called a *pointer* or *codeword*. If no match is found, the encoder has to inform the decoder of this fact and then send the next character. Of course, if the input alphabet is included in the dictionary, a matching phrase can always be found.

There may be more than one entry in the dictionary that matches a prefix of the text at the current position. With greedy parsing, the longest such match is chosen. Greedy parsing, however, is optimal only for certain dictionary schemes, such as a dictionary whose entries consist of at most two characters. Although optimal parsing can be performed [102, 33], greedy parsing is fast and works well in practice.

Early dictionary coders usually operate with a static dictionary that is fixed for all input. Static dictionary methods suffer the same limitations as static statistical methods, as discussed in Section 1.1.4. As with statistical methods, one can construct a custom dictionary for each input (semi-adaptive) or periodically change the dictionary to reflect the recent past input (adaptive). Also, adaptive dictionary coding has been shown [2, 113, 114] (by analysis in some cases and by experiments in others) to perform as well if not better than static and semi-adaptive coding.

1.2.1 Lempel-Ziv Coding

In 1977 and 1978, Ziv and Lempel [113, 114] introduced two adaptive dictionary schemes that has revolutionized the practice of data compression. Besides introducing practical coders, Ziv and Lempel proved that their adaptive dictionary coders are asymptotically optimal for a general source model.

The coding schemes of Ziv and Lempel have come to be known as Lempel-Ziv coding. The two schemes introduced in 1977 and 1978 are known as LZ77 and LZ78, respectively. Although introduced only a year apart, the two schemes are significantly different and gave rise to two classes of dictionary coders; however, adaptive dictionary algorithms based on either of these techniques are collectively known as Lempel-Ziv coders. Since their introduction, many variants of LZ77 and LZ78 have been proposed and some have even been patented. Due to their speed and simplicity of both encoding and decoding, Lempel-Ziv coders have been used for real-time compression for storage and communications applications and are integral to several international standards.

1.2.2 LZ77-Class Coders

We first describe the LZ77 class of dictionary coders. At a high level, an LZ77 coder works by replacing a repeated substring with a pointer to an earlier occurrence in the input. The pointer is represented by the pair (d, l), where d is a displacement that indicates the position of the matching phrase relative to the current position, and l is the length of the substring (which may be empty). After transmitting the pointer, the encoder transmits the character following the coded substring `acaabbacbcabcbbcaac|bcabccbabcba...`

(a) Input text

`a,c,aa,b,ba,cb,cab,cbb,caac`

(b) Parsed phrases

`bcabcc`

(c) Next phrase

Figure 1.2: Snapshot of LZ77 coder. A portion of the input is shown in (a). The current coding position is indicated by the vertical bar. The previously parsed phrases are listed in (b), and (c) shows the next phrase, corresponding to the longest match of `bcabc` followed by the literal `c`.

literally, that is, as it is coded in the input.[4] The concatenation of the matched substring and the coded character is called a *phrase*. In the original LZ77 coder, the displacement d and length l are limited in range and are represented using a fixed number of bits for each. The effective dictionary, therefore, consists of all substrings, within the specified lengths, that occur previously in the input within a finite range of the current position. Conceptually, one can think of the dictionary as being defined by a *sliding window*, equivalently first-in-first-out (FIFO) queue, of the immediate past input. This is how LZ77 was originally presented and how many LZ77-based coders have been implemented.

We illustrate the LZ77 algorithm with an example in Figure 1.2. From the current position, the longest prefix matching a substring of the previous encoded text is `bcabc`. This is coded by the pair $(11, 5)$. The following character, `c`, is coded literally.

1.2.3 LZ78-Class Coders

Whereas the LZ77 coder includes in its dictionary all substrings in the sliding window below a maximum size, the LZ78 coder described in [114] builds its dictionary in a slower, more structured manner. Initially, the dictionary contains only the empty string. From the current position, the dictionary is searched for the longest entry that matches a prefix of the input starting at the current position. (The longest match may be the empty string.) The index of the match is transmitted using $\lceil \log_2 L \rceil$ bits, where L is the number of entries in the dictionary. Then the character following the match in the input is coded literally. A new phrase that is the concatenation of the match and the literal character is added to the dictionary. It is easy to verify that if a phrase is in the dictionary, then all its prefixes are also in the dictionary.

We illustrate the LZ78 algorithm with an example in Figure 1.3. From the current position, the longest prefix matching an entry in the dictionary is `cb`. The corresponding index, 6, is coded

[4] Ziv and Lempel group the displacement, length, and character into one codeword. We adopt a different perspective here to show the concept of an implicit dictionary.

`acaabbacbcabcbbcaa|cbcabccbabcba...`

(a) Input text

`cbc`

(b) Next phrase

index	string
0	
1	`a`
2	`c`
3	`aa`
4	`b`
5	`ba`
6	`cb`
7	`ca`
8	`bc`
9	`ab`
10	`cbb`
11	`caa`

(c) Dictionary

Figure 1.3: Snapshot of LZ78 coder. A portion of the input is shown in (a). The current coding position is indicated by the vertical bar. The current dictionary is listed in (c), where the first entry is the empty string. The next phrase is shown in (b), corresponding to the longest match of `cb` followed by the literal `c`.

using $\lceil \log_2 12 \rceil = 4$ bits. The following character, `c`, is coded literally. The new phrase, `cbc`, is then inserted into the dictionary.

1.2.4 LZFG Coder

In LZ77, the longest matching substring may be found in more than one location in the sliding window; that is, a phrase may be coded by any of a number of codewords. This is a source of coding inefficiency and is referred to as *pointer redundancy*. In 1989, Fiala and Greene [27] introduced a dictionary coder that eliminates pointer redundancy by a judicious choice of dictionary data structure and pointer encoding scheme. Introduced as the C2 coder, it has become known as LZFG [2] in honor of its inventors.

LZFG eliminates pointer redundancy by coding a substring match as a pointer into a digital search tree, *trie* for short, that stores substrings that have occurred previously in the text. An important property of a trie is that each substring stored in the trie has a unique pointer. LZFG is similar to LZ78 in that stored substrings start at boundaries of previously parsed phrases. Unlike LZ78, the effective LZFG dictionary consists of all prefixes (up to a maximum length) that start at boundaries of previously parsed phrases. It is similar to LZ77 in its use of a sliding window. The expressiveness of the effective dictionary is between that of LZ77 and LZ78.

Central to the LZFG coder is its use of the PATRICIA trie [73, 54] data structure. A trie is a tree used for searching where each branch is labeled with a character. Each node in the trie stores an item whose key is the sequence of characters encountered on the path from the root to that node. A PATRICIA trie is a trie in which a sequence of unary branches is compressed into a single branch. A PATRICIA trie saves space by not storing every character explicitly in the nodes.

Instead, each node stores a position/length pointer (p, ℓ) to a string, which is stored externally in a buffer. For a leaf node, the corresponding string extends indefinitely; i.e., $\ell = \infty$. We can view the arcs between nodes in a PATRICIA trie as being labeled with strings such that the concatenation of the labels on the path from the root to a node gives the string pointed to by that node. In LZFG, each internal node stores a string previously matched by the coder.

The LZFG coder uses a PATRICIA trie both for searching and for coding the identity of coded phrases. An example of a PATRICIA trie data structure that is built by the LZFG coder is shown in Figure 1.4. From the current input position, the trie is traversed starting at the root node. The longest matching substring, `caa` in this example, is found by traversing from the root to node 3 (matching `c`), from there to node 4 (matching `b`), and finally to leaf 1 (matching `a`). The first mismatch occurs in the branch from node 4 to leaf 1. This illustrates the search phase of the encoding algorithm.

LZFG identifies an element s of the dictionary encoded by its PATRICIA trie as follows. First, it informs the decoder, using one bit, whether the first node *following* the position in the trie corresponding to s is a leaf or an internal node. If it is an internal node, the identity of the "following" node is transmitted using a phased-in code and the position of the last matched character in the branch leading to the given node is also transmitted using a phased-in code, except if the branch is labeled by a single character, in which case no bits need to be sent. Unlike internal nodes, the substrings represented by a leaf can extend indefinitely. If the "following" node is a leaf, then the position of the last matched character in the extension is coded using yet another static variable-length code. All the static variable-length codes used belong to a parameterized class called start-step-stop codes that can be quickly encoded and decoded. In our example, the longest match, `caa`, is coded with the pointer (L,1,1). The pointer indicates that the "following" node is leaf 1 and that the last matching character is the first character of the branch's label.

The new phrase, `caac`, can now be inserted into the trie by creating a new leaf at the point of the first mismatched character. The new leaf represents the string starting at the current coding position and extending to the end of the input. If the mismatched character is in the middle of a branch, then a new node is created to "split" the branch. The new phrase starts at position 15 (counting from 0) in the input, and this information is "percolated" up to the root. The trie that results from inserting the new phrase in our example is shown in Figure 1.5.

An important observation is that one leaf and at most one node are created every time a new phrase is inserted into the PATRICIA trie. This implies that the storage requirements for the trie is linear in the number of phrases.

In order for the decoder to decipher the pointers transmitted by the encoder, it must maintain a duplicate copy of the PATRICIA trie used for encoding. This requirement results in a decoder that is of about the same complexity as the encoder. This parity in complexity is called the *symmetric* property. This is in contrast to the *asymmetric* property of LZ77 and LZ78, where encoding is more complex than decoding.

1.3 Equivalence of LZ and Statistical Coders

Although dictionary and statistical coding appear at first glance to be quite different, it has been shown [93, 56, 3], that some of the Ziv-Lempel algorithms that use greedy parsing have equivalent statistic coders that give the same compression for all inputs.[5] The connection between LZ and statistical coders goes beyond equivalence of code lengths. A more fundamental result is that an LZ dictionary structure can be decomposed into a set of finite-context statistical models that, when used in conjunction with an appropriate context-selection scheme, gives the same compression as the original dictionary method. At a high level, the statistical equivalent of a dictionary coder initially uses an order-0 context when coding the first character of a corresponding parsed phrase. The statistical coder then uses an order-1 context to code the second character in the phrase, continuing to use increasing order contexts for subsequent characters in the phrase. At the end of the current phrase, the statistical coders resets to using a lower-order context, most of time the order-0 context, for the start of the next phrase.

Figure 1.6(b) shows a finite-state automaton with transition probabilities that is equivalent to the LZ78 dictionary in Figure 1.6(a). The nodes are labeled with the corresponding coding context, where Λ denotes the empty context. The transitions are labeled with probabilities of the corresponding context models. Note the use of an explicit *escape* symbol to indicate a reset to a lower-order context. This is similar to the use of the escape symbol in PPM.

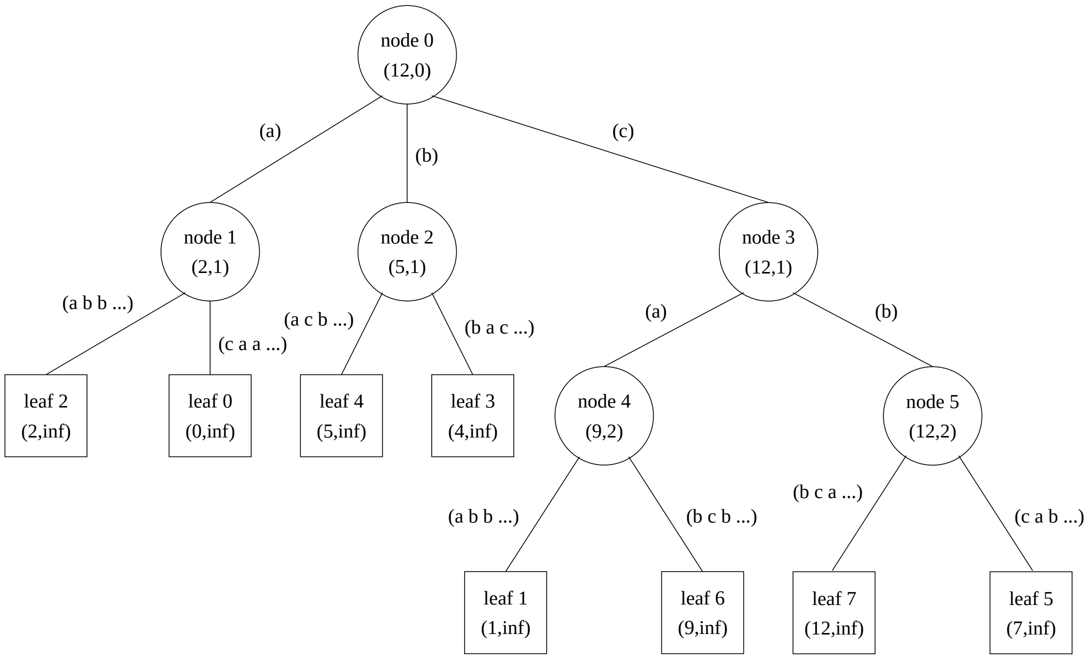
Consider the operation of the statistical coder in Figure 1.6(b) for input `ab`. The initial state is at node Λ. Upon seeing the first input character `a`, the finite-state automaton (FSA) branches to node `a` and transmits $-\log_2 \frac{1}{2} = \log_2 2 = 1$ bit. Reading the next character `b`, the FSA jumps to node `ab` and emits $-\log_2 \frac{3}{4} = \log_2 \frac{4}{3}$ bits. Without reading any input character, the FSA takes the escape transition to node Λ and emits $-\log_2 \frac{2}{3} = \log_2 \frac{3}{2}$ bits. At this point, the coder used $\log_2 2 + \log_2 \frac{4}{3} + \log_2 \frac{3}{2} = 2$ bits to code the string `ab`, the same number of bits that a greedy dictionary coder would use given the dictionary in Figure 1.6(a).

The example in Figure 1.6 demonstrates one source of inefficiency in dictionary coders. Since the string `ab` will never be parsed into two phrases `a,b` by a greedy parser, there is some waste in the bit allocation that allows for this possibility. This redundancy is manifested in the escape probability from node `ab` being less than 1. A more efficient statistical model can be constructed by setting the escape probability to 1.

[5] In a sense, these results imply that statistical coders are more general than dictionary coders and can always give identical or better compression. However, the main advantage of dictionary methods is their faster running time.

acaabbacbcabcbb|caacbcabccbabcba...

(a) Input text



(b) PATRICIA trie

a,c,aa,b,ba,cb,cab,cbb

(c) Parsed phrases

caac

(d) Next phrase

Figure 1.4: Snapshot of LZFG coder. A portion of the input is shown in (a). The current coding position is indicated by the vertical bar. The current PATRICIA trie is shown in (b) and is built from the parsed phrases shown in (c). The next phrase shown in (d) corresponds to the longest match of caa followed by the literal c.

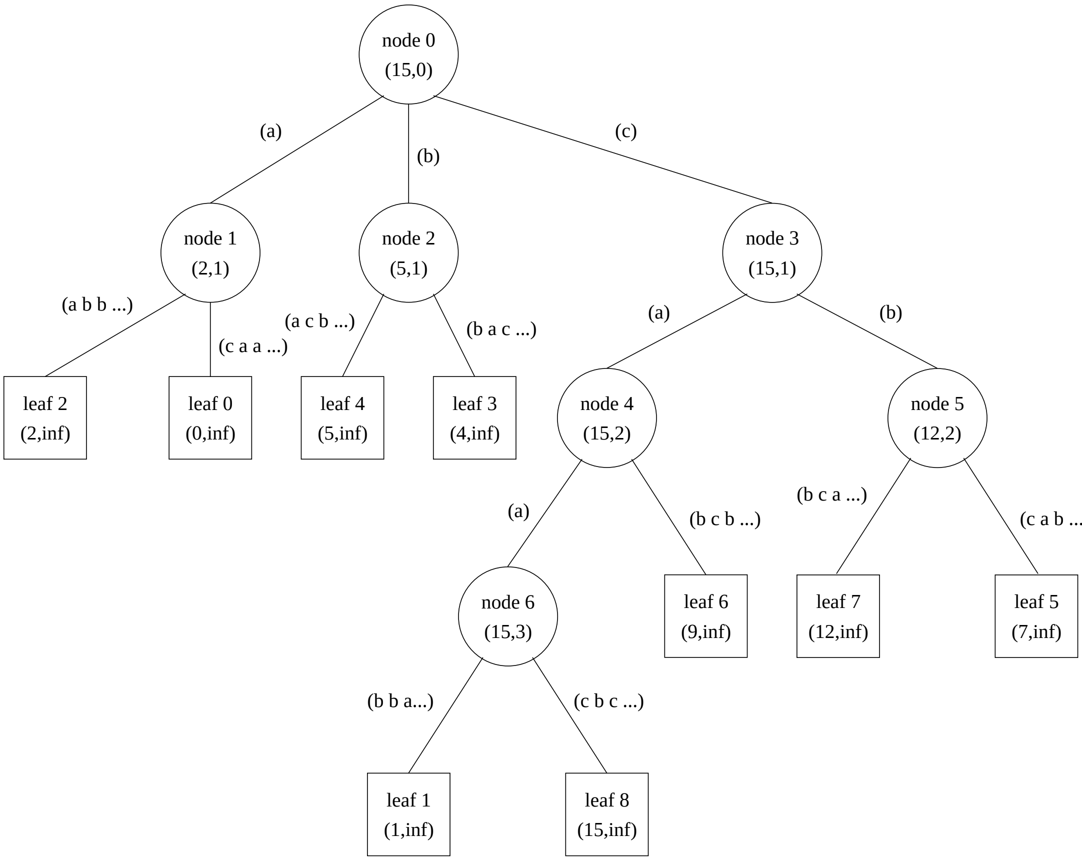
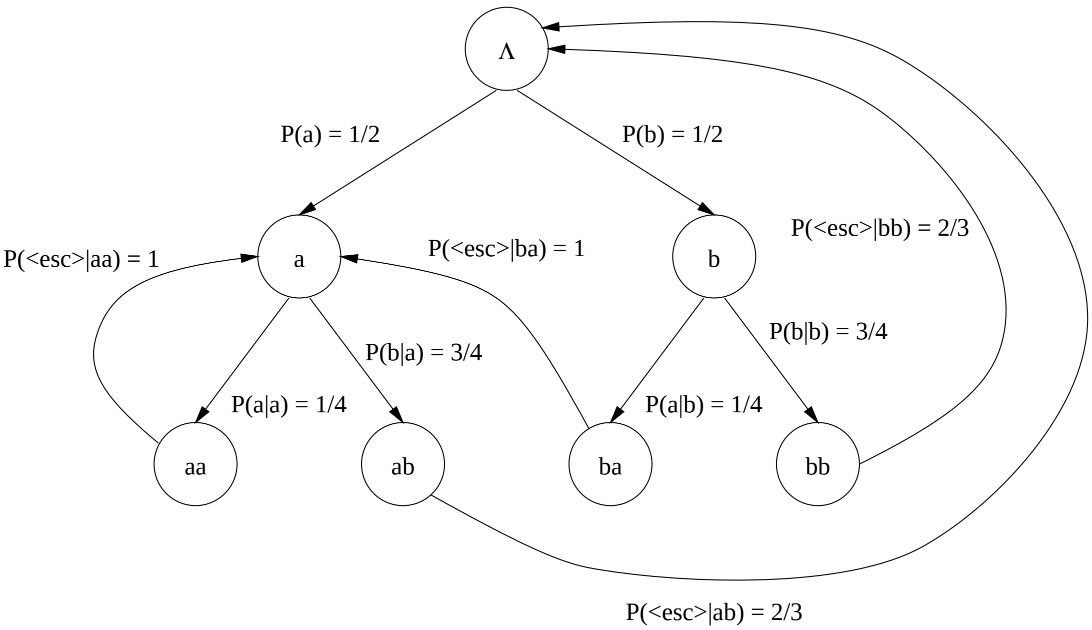


Figure 1.5: PATRICIA trie of Figure 1.4 after insertion of the next phrase, `caac`. In the insertion, node 6 and leaf 8 are created.

index	string
0	a
1	b
2	ab
3	bb

(a) LZ78 dictionary



(b) Statistical equivalent coder

Figure 1.6: Statistical equivalent of a dictionary coder. A finite-state automaton representation is shown in (b) of a statistical coder equivalent to the LZ78 dictionary in (a) that uses two bits to code each entry. The alphabet is $\{a, b\}$.

Chapter 2

Dictionary by Partial Matching

2.1 Introduction

Recently there has been much interest in constructing lossless coders that combine the compression efficiency of statistical coders with the speed of dictionary-based coders. With statistical coders, there has been some work on techniques for arithmetic coding [41, 42, 70] that tradeoff arithmetic precision (and therefore compression) for speed. Furthermore, efficient data structures and coding heuristics have been developed to improve the running time and resource requirements of statistical modeling. While these techniques have succeeded in speeding up statistical coding without sacrificing much compression, they are ultimately limited to coding one character at a time.

From the dictionary coding side, some recent works [32, 75] have attempted to improve the compression of dictionary-based methods without slowing them down too much by maintaining separate dictionaries, one for each source character. A dictionary is chosen for coding based upon the last character coded. In effect, the dictionary used is conditioned on the last character of the last string coded. Since the last character coded is known by both the encoder and decoder, the same dictionary will be used for both encoding and decoding.

One motivation for using multiple dictionaries is that each dictionary can be made smaller, and therefore pointers into each dictionary can be represented more compactly. A drawback is that there is some overhead to managing multiple dictionaries. However, the increase in overhead can potentially be met with a decrease in search time, due to the smaller dictionaries.

Another motivation for using multiple dictionaries comes from the insights gained from the proofs of equivalence between dictionary and statistical coding. In this light, the use multiple dictionaries corresponds to an additional level of context modeling on top of the variable-context mechanism inherent in a greedy dictionary coder. In effect, a statistical equivalent to an order-1 multiple-dictionary scheme would start its cycle by using a context of one previous character, then two, and so on, instead of starting with an order-0 context. This directly addresses the observation in [2] that "the compression achieved by dictionary schemes suffers because of the loss of context at phrase boundaries" since some context is retained at phrase boundaries with context dictionaries.

The work of [32] placed a heavy emphasis on speed, and therefore compression gain was not demonstrated. The compression results of the CSD algorithm of [75], which is based upon the LZW [107] dictionary coder, show more promise. However, the CSD algorithm is limited to using a single-character context.

In this chapter, we take the multiple-dictionary-based approaches of [32, 75] further with a general scheme that combines the finite-context modeling techniques of PPM with various LZ dictionary coders.

2.2 Dictionary by Partial Matching

At a high level, the new hybrid method, which we call Dictionary by Partial Matching, or DPM for short, works as follows. The coder maintains a list of contexts of length 0 up to some upper limit o. For each context in the list, the coder maintains a dictionary containing strings that have previously occurred following the context. To encode the input starting from the current position, the coder first looks for the longest context in its collection that matches a context of the current position: call the context σ. It then looks for the longest string in σ's dictionary that matches a prefix of the input starting at the current position. If there is a match, the identity of the matching string of σ's dictionary is transmitted, otherwise a special *escape* symbol is transmitted, and the process is repeated with the next longest context. If even the length-0 context fails to find a match, a final escape is sent, and the character in the position to be coded is sent literally. If the identity of a string in some dictionary is sent, the position to be coded is advanced to the first position in the input past those “matching” the given string. If a literal is coded, the position is moved forward one. In either case, the dictionaries corresponding to the matching contexts are updated. Optionally, the context dictionaries that were escaped from may be updated as well.

We now illustrate DPM with an example in Figure 2.1. At the current position, indicated by a vertical bar in (a), the available contexts are `qui`, `ui`, `i`, and Λ (the zero-order context). The dictionaries for each context, except for Λ, are shown in (b), (c), and (d). A search of the `qui` dictionary does not yield a match, and so the escape symbol is coded using $\lceil \log_2 5 \rceil = 3$ bits. The `ui` dictionary is then searched, yielding the longest match at index 0. The index is coded with $\lceil \log_2 8 \rceil = 3$ bits.

The above example illustrates the basic operations of DPM. It also demonstrates the savings in coding a dictionary pointer that results from coding with a small high-order dictionary. If the `i` dictionary were used, nine bits would be needed to code a pointer. With the same memory usage, a conventional dictionary coder would require even more bits to code a pointer.

The example leaves out many important details that would go into the design and implementation of an actual DPM coder. In order to get a better idea of what these issues are, we consider three instantiations of the DPM idea, one for each of the dictionary methods described in Chapter 1: LZ77, LZ78, and LZFG. The selection of a dictionary method involves choosing a data structure for maintaining the dictionaries, a policy for adding and removing strings from the dictionaries, and a scheme for coding dictionary pointers and escapes. LZ77-PM, the coder described in Section 2.3, makes these decisions in a manner analogous to the A1 instantiation in [27] of the LZ77 method.

`The␣qui|ck␣brown␣...`

(a) Input text

index	string
0	`t`
1	`z`
2	`et`
3	`lt`
4	⟨Esc⟩

(b) `qui` dictionary

index	string
0	`ck`
1	`de`
2	`ld`
3	`sh`
4	`lty`
5	`dity`
6	`lding`
7	⟨Esc⟩

(c) `ui` dictionary

index	string
0	`d`
1	`n`
2	`t`
3	`ck`
4	`nd`
⋮	⋮
500	`zation`
501	⟨Esc⟩

(d) `i` dictionary

Figure 2.1: Example of a DPM coder with maximum order $o = 3$. At the current position, indicated by a vertical bar in (a), the available contexts are `qui`, `ui`, `i`, and Λ (the zero-order context). The dictionaries for each context, except for Λ, are shown in (b), (c), and (d).

LZW-PM, the coder of Section 2.4, is based upon LZW, an LZ78 variant. LZFG-PM, the coder of Section 2.5, is based upon an implementation of LZFG by Slyz [99].

The LZ77-PM and LZW-PM schemes have been simulated in software with programs that keep count of the number of encoded bits, but do not actually generate a compressed output. Since the main interest is in the compression gain by adopting DPM, no attempt has been made to use efficient data structures or searching techniques, except for a bare minimum to make the simulations run in a reasonable amount of time with the available memory.

The LZFG-PM scheme, on the other hand, has been implemented as a fully functional encoder and decoder. Memory use is limited by fixing the size of the dictionary data structures. This allows for evaluation of DPM in a real-world setting where memory resources are limited.

2.3 LZ77-PM

In an LZ77-style coder, the dictionary is implicitly defined as all substrings, within length restrictions, that are contained in a buffer of the recent past input. In defining a multiple-dictionary extension of LZ77, we seek to preserve the nature of the dictionary. A natural extension of LZ77 to multiple dictionaries, then, is to have the effective dictionary for context σ contain all substrings, within length restrictions, that have occurred in the past following the context σ. This is the approach described in [32] for single-character contexts, where each context dictionary is implemented as a linked list with hashing.

We now describe an PM modification of a simple LZ77 coder that is almost identical to the A1 coder from [27].

2.3.1 Baseline Coder

The baseline (non-PM) coder works as follows. First, it divides the input into buffers of size 256K (K=1,024 characters), and codes each buffer separately. While coding a particular buffer, the dictionary at any given time consists of all strings in the buffer preceding the current position that are of length at least 2 and at most 16. At each coding step, the encoder finds the longest string in the dictionary that is identical to the string of the same length obtained by reading forward in the buffer starting at the current position. If p is the buffer position being coded (counting from 1), the encoder tells the decoder which string "matched" by giving the position earlier in the buffer where the match occurred along with the length of the match. The position is encoded using $\lceil \log_2 p \rceil$ bits. The length is encoded in binary using four bits. If there is no matching string in the dictionary, the remaining 4-bit codeword (the first 15 were used to encode lengths 2 through 16) is transmitted, and the character at position p is coded literally; i.e., as it appears in the input.

2.3.2 PM Modification

Our PM modification involves the addition of context dictionaries together with a PPM-style escape mechanism. As in the baseline coder, the input is coded in 256K buffers. The context dictionaries are not explicitly maintained. Instead, when searching for a match in context σ, σ is included as the prefix of the search string. For example, let α be the input string of length 16 (the maximal match length) that starts at the current coding position. A search in context σ would involve searching for the longest prefix of $\sigma\alpha$ that has occurred in the buffer before the current position. In conducting this search, the number of times σ has occurred previously in the buffer is recorded by counter n. The position k of the longest match is noted as the value of n the first time the string is matched. In essence, the search in context σ returns the length and index of the longest match within the implicit context dictionary. Let N denote the final value of counter n, β the longest prefix of α that is matched, l_β the length of β, and k the value of n the first time β was matched. If N is above a given threshold T_σ, the encoder uses the current context for coding. Otherwise, the encoder escapes to a lower-order context. Since the decoder can perform the same test, no escape symbol needs to be sent. If $N > T_\sigma$ and no matching string is found, the encoder sends an escape symbol and escapes to the next lower-order context. If $N > T_\sigma$ and a match is found, the index k and the length l_β of the match are transmitted using $\lceil \log_2 N \rceil$ and 4 bits, respectively.

By introducing the threshold T_σ, we are delaying the use of a context dictionary until it reaches a certain size. A reason for this is that a small dictionary is not likely to yield a match. Therefore, the expense of having to transmit an escape symbol every time there is no match is great relative to the potential savings in coding by using the dictionary when there is a match. Although a similar problem exists in statistical coding, the implications for dictionary coding are more severe since the escape symbol must be coded in a more restricted setting where an integral number of bits is used. For example, the escape symbol may be coded as a flag bit, as a special length value, or as a special index value.

In the baseline coder, only strings with at least 2 characters are coded by dictionary reference. This is done since it is more economical to code a one-character string as a literal using 12 bits than as a dictionary pointer using $4 + \lceil \log_2 p \rceil$ bits when $p > 256$, where p is the current coding position in the buffer. We face a similar situation with the PM modification. Experiments show that coding with a minimum length of 2 is better than with a minimum of length of 1.

We have explored several ways to encode the escape symbol. The first is to code the escape as a length-0 match without a match index, using 4 bits, as is done in the baseline coder when coding a literal. The second is to code the escape with a match index of 0 without a length field, using $\lceil \log_2 N + 1 \rceil$ bits. In experiments with text files in the Calgary corpus [2], coding the escape as a length-0 match and using a minimum match-length of 2 yielded consistently better compression.

2.3.3 Experimental Results

We performed coding experiments to observe the effects of the threshold T_σ and the maximum context order in coding a test file (`paper2`) from the Calgary corpus. In one experiment, we limited the maximum context order to 1 and varied the threshold for order-1 contexts. In a second experiment, we fixed the threshold for order-1 contexts to 300 and varied the threshold for order-2 contexts. The compression results are summarized in Table 2.1 and are shown in more detail in Figure 2.2 as plots of the distribution of coding bits versus the threshold value. As expected, the proportion of bits spent coding in the highest context order decreases as the threshold is increased.

For the order-1 LZ77-PM coder, the compression efficiency peaks when the threshold is about 300. For the order-2 LZ77-PM coder, the compression rate approaches, but does not beat, that of the order-1 LZ77-PM coder, even when the threshold is 2000. This suggests that a single-character context is adequate to restore context at phrase boundaries in an LZ77-style coder.

Complete results for coding the Calgary corpus with an order-1 LZ77-PM coder are shown in Table 2.2. Since both the base LZ77 and LZ77-PM coders are limited to matches of length 16 or less, there is a upper bound of the compression ratio that is achievable. For example, this limit is reached for the file `pic`, a digital image that can be compressed fairly well with the other dictionary coders.

On average, DPM improves the compression rate by about 2%. It should be noted, however, that for the text files in the corpus, the improvement is about 3.4%. The non-text files in the corpus generally perform worse with DPM than without.

2.4 LZW-PM

In the LZ78 coder, a literal character is transmitted after every dictionary pointer. One reason for this is to insure that at least one character is coded with each encoding step. In [101], this is referred to as *pointer guaranteed progress.* The LZW coder [107] eliminates the need to transmit a literal after every match by initializing the dictionary to contain the source alphabet, that is, all strings of length 1. In this way, only dictionary pointers are coded. Since all characters are in the dictionary, a match will always be found. This is referred to as *dictionary guaranteed progress* in [101].

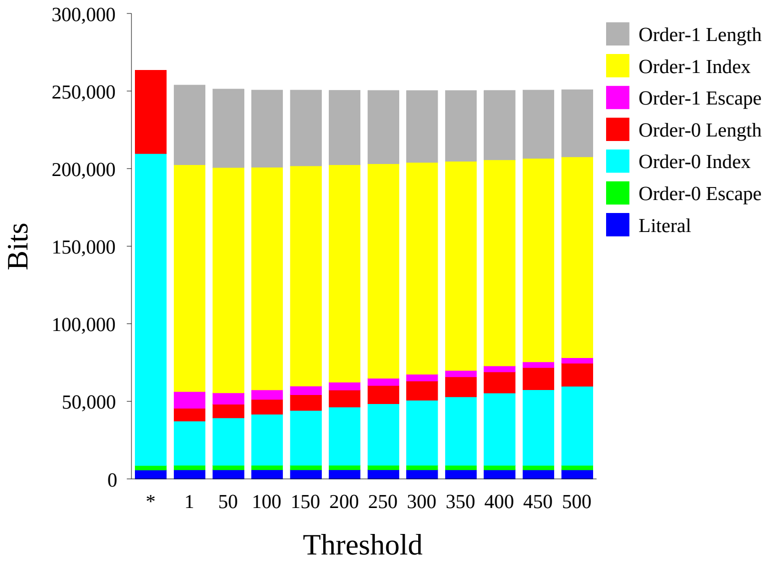
Threshold	Bits for Literal	Bits for Order 0	Bits for Order 1	Total Bits	Total Bits/Symbol
(∗)	5744	257099	N/A	262843	3.198
1	5864	39639	208335	253838	3.088
50	5848	42310	203126	251284	3.057
100	5872	45379	199363	250614	3.049
150	5864	48510	196226	250600	3.049
200	5880	51403	193176	250459	3.047
250	5872	54297	190202	250371	3.046
300	5864	57215	187244	250323	3.045
350	5840	59948	184541	250329	3.045
400	5816	63143	181435	250394	3.046
450	5808	65921	178865	250594	3.049
500	5816	68730	176297	250843	3.052

(a) Order-1 LZ77-PM

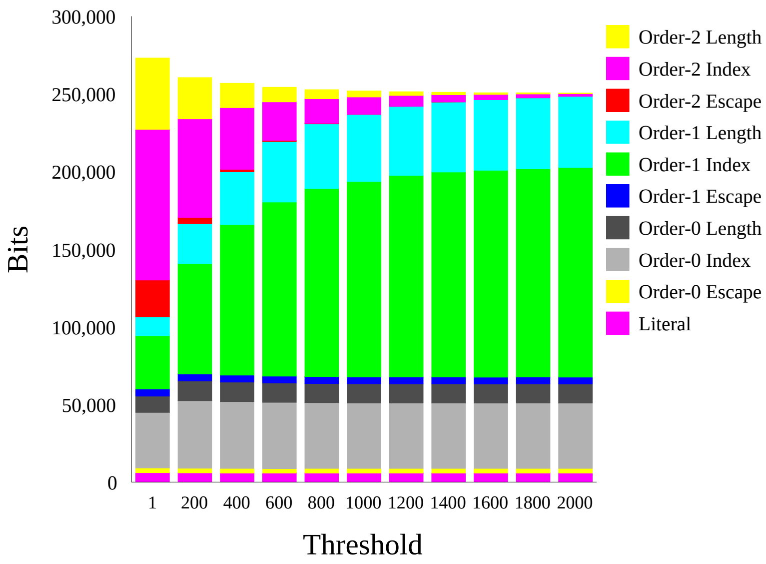
Threshold	Bits for Literal	Bits for Order 0	Bits for Order 1	Bits for Order 2	Total Bits	Total Bits/Symbol
1	6040	49279	51009	166785	273113	3.323
200	5920	59124	101287	94204	260535	3.170
400	5864	58500	135435	57034	256833	3.125
600	5872	57882	155486	35040	254280	3.093
800	5864	57613	167182	22083	252742	3.075
1000	5872	57399	173406	15271	251948	3.065
1200	5872	57342	178608	9598	251420	3.059
1400	5864	57382	181428	6391	251065	3.054
1600	5856	57277	183037	4588	250758	3.051
1800	5864	57339	184155	3384	250742	3.050
2000	5864	57299	185195	2231	250589	3.049

(b) Order-2 LZ77-PM

Table 2.1: Distribution of bits for coding `paper2` with various threshold values. Results are shown in (a) for an order-1 LZ77-PM coder. The row marked with (∗) gives results for the baseline non-PM LZ77 coder. In (b), results are shown for an order-2 LZ77-PM coder with fixed order-1 threshold of 300.



(c) Order-1 LZ77-PM



(d) Order-2 LZ77-PM

Figure 2.2: Distribution of bits versus context threshold T_σ. The results are shown for coding file `paper2` of the Calgary corpus. In (a), except for the first column, the results are for varying the order-1 context threshold for LZ77-PM with maximum context order of 1. The first column of (a), indicated by ($*$), shows the coding results for the non-PM baseline coder. In (b), the results are for varying order-2 context threshold with an order-1 threshold of 300 for LZ77-PM with maximum context order of 2.

LZW uses the same dictionary update rule as LZ78. After each match, a single string consisting of the matched string concatenated with the first unmatched character is added to the dictionary. This is one case where the decoder's dictionary differs from the encoder's dictionary since the decoder does not yet know the identity of the unmatched character. This is not as serious a problem as it may first seem. If the encoder happens to send the newly added string as the next match, the decoder can recover the unmatched character as the first character in the next match, which is already known to the decoder.

2.4.1 Baseline Coder

In the UNIX `compress` implementation of LZW, the size of the dictionary is restricted. In defining our baseline coder, this restriction is removed. Also, instead of initializing the dictionary to contain the characters of the alphabet, the dictionary initially contains only the empty string. When the longest match is the empty string, the next character in the file is coded literally. This can be viewed as an escape mechanism similar to coding the escape symbol as a match index in the LZ77-PM coder described above. In [101], this is referred to as *on-the-fly dictionary guaranteed progress*. This modification has a negligible effect on the compression and is done so that we can view the baseline coder as a special case of the PM coder where the maximum context is of order 0.

In LZW, the dictionary is updated in the same manner as in LZ78. The longest matching string concatenated with the first unmatched character is added to the dictionary. The result is that one entry is added for each dictionary match coded. In contrast, an LZ77 coder grows its effective dictionary at a faster rate.

2.4.2 PM Modification

In the LZW-PM coder evaluated in our experiments, we maintain an LZW dictionary for each previously seen context up to a maximum length, say o. As in the baseline coder, an important difference from LZW is that the dictionary initially contains the empty string in order to make use of the escape mechanism.

At each coding step, the longest matching context, say σ, is used for coding. The dictionary corresponding to σ is searched for the longest match. Regardless of the length of the match, which may be zero, the LZ78 dictionary update rule is used. If the longest match is the empty string, an escape symbol is coded[1] and the next longest matching context is used for coding. This process is repeated until a non-empty match is found or until an escape is coded from the order-0 context. A character is coded literally after an escape from the order-0 context. The identity of a dictionary entry is coded using a phased-in binary code using either $\lfloor \log_2 D \rfloor$ or $\lceil \log_2 D \rceil$ bits, where D is the size of the dictionary.

Coding the escape symbol as a dictionary entry approximately corresponds to the method used in PPMA [2], since it (approximately) involves the implicit assumption that the probability of seeing a previously unseen character in a given context is the inverse of the number of times the

[1]The empty string occupies one entry in the dictionary and an escape is coded by coding the index of the empty string.

context has been seen. While other probability models for escapes performed better as part of PPM (see [2, 41]) and are therefore likely to yield better compression here, we chose the method above for simplicity and computational efficiency.

We found that, unlike the case of LZ77-PM, delaying the use of a context does not yield better compression for LZW-PM in general. This is most likely due to the different rates at which the dictionaries are updated. Since LZ77-PM bases its dictionaries on the contents of a buffer of past input, each time a match is coded, multiple entries are added to the effective dictionaries of several contexts corresponding to the matched string. In contrast, only one entry is added to each context matched in LZW-PM. Intuitively, in the early growing phase, the LZ77-PM dictionaries quickly become "diluted", increasing the number of bits required to code a match without a corresponding increase in the reliable of prediction. Only after the initial growing phase does the cost of coding a match balance the reliability of prediction.

2.4.3 Experimental Results

Experiments on the Calgary corpus show a significant improvement in the compression rate when DPM is applied to the baseline LZW coder. The best results for the corpus as a whole is obtained for a maximum context order of 2. However, the best maximum context order for each file varies. For some of the binary files, notably `geo`, the compression degrades when using even an order-1 context dictionary. In contrast, the text files are better compressed by using context dictionaries. On average, introducing Partial Matching improves the compression rate by about 11%. For text files, the improvement is about 15%.

2.5 LZFG-PM

In the LZ77 coder, for a given match, several positions in the buffer may contain the same match. As a result, LZ77 effectively reserves codespace for pointers that will never be coded, which is wasteful. The LZFG coder of Fiala and Green [27] eliminates pointer redundancy by coding a string as a pointer into a trie, a digital search tree which stores parsed substrings that have occurred previously in the text. Each substring stored in the trie has a unique identifier. LZFG's efficient coding of pointers makes it an ideal candidate as a base dictionary coder for DPM. In addition, LZFG is a symmetric coder since the decoder must maintain the same trie. In a sense, DPM is also a symmetric coding scheme since the encoder and decoder must make the same context selection. Integrating DPM into LZFG, then, would have minimal impact on both the encoder and decoder. For these reasons, we chose to implement LZFG-PM as a fully functional encoder/decoder program.

2.5.1 Baseline Coder

We use an existing LZFG implementation [99] as the baseline coder. Some details of this implementation differ from the coder originally described by Fiala and Greene. In the base implementation, a sliding-window buffer of size 44K is used with the sliding occurring in 4K chunks. A limited number of internal nodes and leaves is globally allocated. Nodes and leaves are stored in self-organizing lists that use the least-recently-used (LRU) update heuristic. New nodes and leaves replace old ones when their allotted number has been reached. The interested reader is referred to [99] for further details of the implementation.

2.5.2 PM Modification

Following the outline of Section 2.2, LZFG-PM constructs separate tries for each context, including one for the order-0 context. Although the tries are separate, nodes and leaves are allocated from a common pool. When all nodes (respectively, leaves) have been allocated and there is a request for a new node (leaf), the oldest node (leaf) in a globally managed LRU list is deleted. In this way, fixed memory usage is assured. Furthermore, since the different contexts are competing for the same memory resources in a LRU list, context dictionaries that are used more frequently will suffer fewer deletions.

Coding of literal characters is handled differently in LZFG-PM than in LZFG. Whereas LZFG groups consecutive literals together and sends them in one chunk, LZFG-PM codes each literal individually. The reasoning is that since LZFG-PM allows pointers to code phrases of length 1 efficiently, literals are less likely to occur consecutively than in LZFG.

We have implemented two methods for coding escapes. In the first, we code the escape symbol as a special node, namely the root. In the second, we code the escape symbol as a special entry in the LRU list of leaves, using the same self-organizing-list heuristic. We find that coding an escape as a node gives slightly better compression than coding it as a leaf.

We use a heuristic to determine the appropriate context order to begin parsing the next phrase. If the previous context order is C and the length of the previous phrase is L, we do not use contexts with order higher than $L + C - 1$ for coding. Starting at an order higher than $L + C - 1$ would not likely be fruitful since that would mean repeating a search that has failed in the past. This heuristic saves us in the cost of coding escapes.

A new dictionary entry is added to the context used for coding as well as to higher-order contexts that were escaped from. Contexts of order less than the coding context could also be updated. In our experiments we find that updating lower-order contexts results in marginal improvement (about 0.01 bits/symbol) in compression at the expense of more overhead. Therefore we adopt the strategy of updating only the coding context and the contexts that are escaped from.

2.5.3 Experimental Results

Experimental results for the LZFG and LZFG-PM coders are given in Table 2.2. For both coders, the buffer size is 44K. Both coders limit the total number of internal nodes to 16K and the number of leaves to 32K. We experimentally determined the best settings for the start-step-stop codes used to code the nodes and leaves based upon a subset of the test files. We also varied the maximum context order and obtained the best results for a maximum context order of 1. The results, given in Table 2.2, are for a maximum context order of 1. The improvement in the average compression rate with DPM is about 4.3%. For text files, the improvement is 6.4%.

file	LZ77	LZ77-PM	LZFG	LZFG-PM	LZW	LZW-PM	CSD
bib	2.85	2.68	2.69	2.44	3.22	2.61	3.22
book1	3.44	3.32	3.57	3.28	3.17	2.74	3.59
book2	2.98	2.84	3.05	2.78	3.06	2.55	3.37
geo	6.71	6.96	5.65	5.63	5.79	6.51	6.74
news	3.51	3.42	3.42	3.25	3.62	3.18	4.13
obj1	5.11	5.24	4.06	4.32	5.05	5.14	5.12
obj2	3.51	3.40	3.10	3.04	4.08	3.42	3.80
paper1	3.24	3.14	2.94	2.74	3.62	3.08	3.43
paper2	3.20	3.05	3.02	2.80	3.38	2.94	3.28
paper3	3.52	3.43	3.27	3.05	3.65	3.30	N/A
paper4	3.94	3.89	3.55	3.41	3.98	3.69	N/A
paper5	4.02	3.98	3.61	3.49	4.20	3.86	N/A
paper6	3.34	3.25	3.00	2.80	3.77	3.19	N/A
pic	2.01	2.01	0.92	0.93	0.94	0.98	1.00
progc	3.28	3.22	2.89	2.74	3.72	3.18	3.42
progl	2.44	2.31	1.95	1.80	2.95	2.37	2.70
progp	2.42	2.34	1.91	1.81	3.00	2.40	2.66
trans	2.31	2.12	1.79	1.67	3.16	2.31	2.91
Average	3.44	3.37	3.02	2.89	3.58	3.19	N/A

Table 2.2: Compression results for files in the Calgary corpus. The results are expressed in bits/symbol. The LZ77 coder is almost identical to the A1 coder from [27]. LZ77-PM used a maximum context order of 1. The LZFG implementation used is from [99]. LZFG-PM used a maximum context order of 1. LZW-PM used a maximum context order of 2. CSD results are from [75]. CSD used a limited dictionary size (16K), while LZW and LZW-PM had unlimited dictionaries; the effects of this is most notable on the longer text files: `book1` and `book2`.

Our implementations of LZFG and LZFG-PM are experimental prototypes that were designed for flexibility and not efficiency. As such, the running times are about an order of magnitude more than "commercial-grade" compressors such as `gzip`. For the file `book1`, LZFG-PM requires 22% more time to compress than LZFG. This gives us an indication of the overhead of DPM over conventional dictionary coders.

2.6 Discussion

In this chapter, we have described a new approach to dictionary-based text compression, and we have demonstrated that it results in improved compression. The improvement is particularly significant when used with LZW, which is the among the fastest of the dictionary methods. We expect that the use of the dictionary PM approach also yields practically fast coders. The fact that this approach succeeded in conjunction with the three coders studied in this proposal suggests that the high-level specification of this approach can be combined with many dictionary methods, including those that might be developed in the future.

Why did the use of contexts improve compression significantly more in conjunction with some dictionary methods than with others? One plausible explanation is as follows. The LZ77 coder is the most liberal about adding entries to its dictionary, followed by LZFG then LZW. This parallels the level of improvement exhibited by introducing DPM, with LZ77 benefiting the least and LZW the most.

Past studies with PPM coders [2] have shown that, as the context order increases, the compression improves and then worsens, with the best compression when the maximum context order is about 4 or 5. The correspondence between (non-PM) dictionary methods and statistical methods suggests that, as a match continues, the probability estimates in the corresponding symbolwise statistical coder tend to get better as more contextual information is made available and then then get worse as the estimates at high context orders become more unreliable. The effect of a context dictionary is, loosely speaking, to bypass the lower order contexts with relatively poor probability estimates, thereby increasing the average quality of the remaining probability estimates.

A method that liberally updates its dictionary, all else being equal, makes longer matches at the expense of requiring more bits to identify the match. By tending to have longer matches, such a method has a relatively greater share of its coding inefficiency due to the later characters in its matches. For such a method, bypassing low-order contexts will not yield much of an improvement. Further, even if, for example, using an order-2 context to encode a single character tends to be worse than using an order-3 context, it is possible that the order-2 dictionaries tend to be better than the order-3 dictionaries. To see why this might be true, consider what happens in the corresponding statistical method when we use an order-2 context rather than an order-3 context. Loosely speaking, instead of cycling from contexts of length 3 up to some large number, we cycle from 2 through some large number. Adding the order 2 to the cycle will result in improved compression if, loosely speaking, using order-2 contexts is better than the average of the use of contexts of order 3 and higher.

A method that is "stingy" about adding entries to its dictionary, like LZW, has a greater share of its inefficiency due to the early characters in a match, and therefore we would expect it to be helped more by the addition of contexts, as is consistent with our experiments.

Part II

Video Compression

Chapter 3

Introduction to Video Compression

In this chapter, we present an introduction to aspects of video compression that will be useful for understanding the later chapters. We begin by describing the generation and representation of digital video. With standard representations defined, we then motivate video compression with several illustrative examples that underscore the need for lossy compression. To better understand the tradeoffs inherent in lossy coding systems, an introduction to rate-distortion theory and practice is presented. Next, we describe existing international standards for video coding and present an overview of the fundamentals of these standards. This chapter is by no means intended to be comprehensive; for an in-depth introduction to video coding fundamentals, the reader is referred to [4, 34, 71, 76].

3.1 Digital Video Representation

For compression to be meaningful, a standard representation should be defined for the data to be compressed. In this section, we give an overview of some of the more popular standard representations for digital video that are in use today.

3.1.1 Color Representation

Excluding synthetic (computer-generated) sources, video originates in the physical world. In a general sense, video can be characterized as a time-varying, two-dimensional mix of electromagnetic signals. Being too general, this characterization is not practical for representing visual information relevant to human observers. Although visible light consists of a continuum of wavelengths, it has been known for several centuries that a small set of primary colors, mixed in the right proportions, can simulate any perceived color. In painting, for example, one system of primary colors is cyan, magenta, and yellow; this is a *subtractive* system since the absence of all primary colors yields the color white. Red, green, and blue light sources form another set of primary colors; this is an

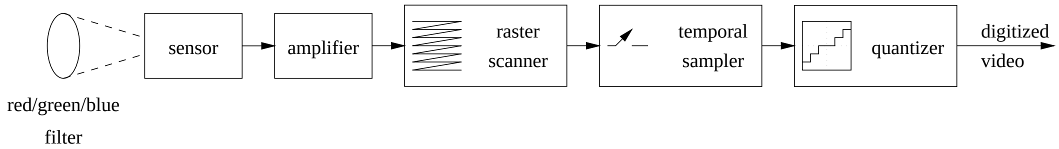


Figure 3.1: Block diagram of a video digitizer.

additive system since the presence of all the primary colors at their maximum intensities results in the perception of the color white. This phenomenon of color perception is caused by the way that the human eye detects and processes light, which makes it possible to represent a visual image as a set of three intensity signals in two spatial dimensions.

3.1.2 Digitization

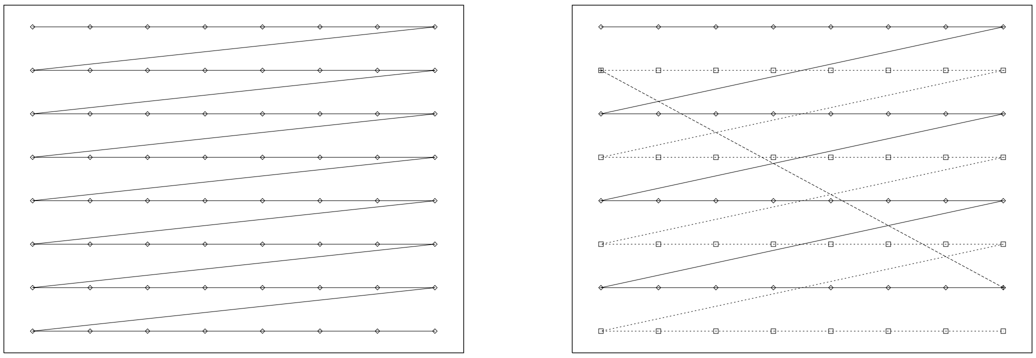
In order to be processed by computers, analog video that is captured by a light sensor must first be digitized. Digitization of video consists of three steps: 1) spatial sampling, 2) temporal sampling, and 3) quantization. A block diagram of the digitization process is depicted in Figure 3.1 for one color component. The steps need not be performed in the order indicated and some steps can even be combined into one operation.

3.1.2a Spatial Sampling

Spatial sampling consists of taking measurements of the underlying analog signal at a finite set of sampling points in a finite viewing area (or *frame*). To simply the process, the sampling points are restricted to lie on a lattice, usually a rectangular grid, say of size $N \times M$. The two dimensional set of sampling points are transformed into a one-dimensional set through a process called *raster scanning*. The two main ways to perform raster scanning are shown in Figure 3.2: *progressive* and *interlaced*. In a progressive (or non-interlaced) scan, the sampling points are scanned from left to right and top to bottom. In an interlaced scan, the points are divided into odd and even scan lines. The odd lines are scanned first from left to right and top to bottom. Then the even lines are scanned. The odd (respectively, even) scan lines make up a field. In an interlaced scan, two fields make up a frame. It is important to note that the odd and even fields are sampled and displayed at different time instances. Therefore the time interval between fields in an interlaced scan is half of that between frames. Interlaced scanning is commonly used for television signals and progressive scanning is typically used for film and computer displays.

3.1.2b Temporal Sampling

The human visual system is relatively slow in responding to temporal changes. By taking at least 16 samples per second at each grid point, an illusion of motion is maintained. This observation is the basis for motion picture technology, which typically performs temporal sampling at a rate of 24 frames/sec. For television, sampling rates of 25 and 30 frames/sec are commonly used. (With interlaced scanning the number of fields per second is twice the number of frames per second.)



(a) Progressive Scan (b) Interlaced Scan

Figure 3.2: Scanning techniques for spatial sampling of a video image.

3.1.2c Quantization

After spatial and temporal sampling, the video signal consists of a sequence of continuous intensity values. The continuous intensity values are incompatible with digital processing, and one more step is needed before this information can be processed by a digital computer. The continuous intensity values are converted to a discrete set of values in a process called *quantization* (or *discretization*.

Quantization can be viewed as a mapping from a continuous domain to a discrete range.[1] A particular quantization mapping is called a *quantizer*. An example is shown in Figure 3.3. In the figure, there are eleven discrete quantization levels, also called *bins*. Each bin has an associated size, which is the extent of the continuous values that map to that bin. In the example, each bin, except for the bins for -5, 0, and 5, has the same size, which is sometimes referred to as the *quantizer step size*. This type of quantizer is called a *uniform* quantizer.

A binary encoding can be assigned to each of the bins. Typically the initial quantization of a continuous source is done using a number of quantization levels that is a power of 2, so that a fixed number of bits can be used to represent the quantized value.[2] This process of representing a continuous value by a finite number of levels using a binary code is often referred to as *pulse code modulation* (PCM). Thus, after spatial sampling, temporal sampling, and quantization, we have $N \times M$ data points, commonly called *pixels* or *pels*, represented using a fixed number of bits.

3.1.3 Standard Video Data Formats

To promote the interchange of digital video data, several formats for representing video data have been standardized. We now review some of the more popular standard representations.

[1] This definition is intended also to encompass mappings from a discrete domain to a discrete range.

[2] Further quantization of digitized data may use a number of quantization levels that is not a power of 2 and employ variable-length entropy coding.

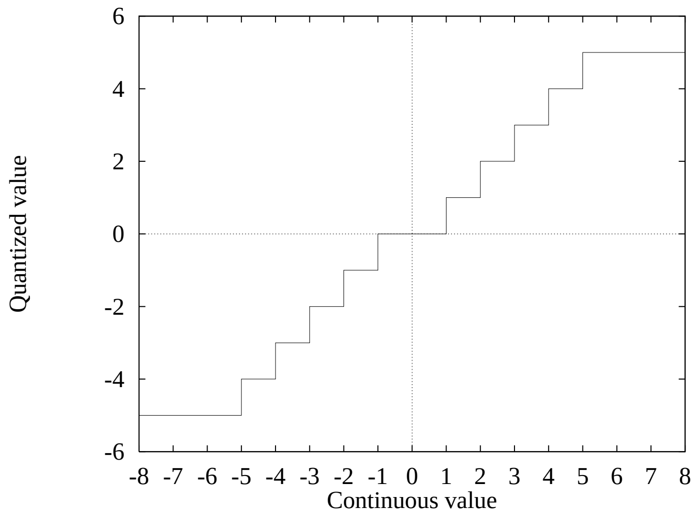


Figure 3.3: Example of uniform quantization.

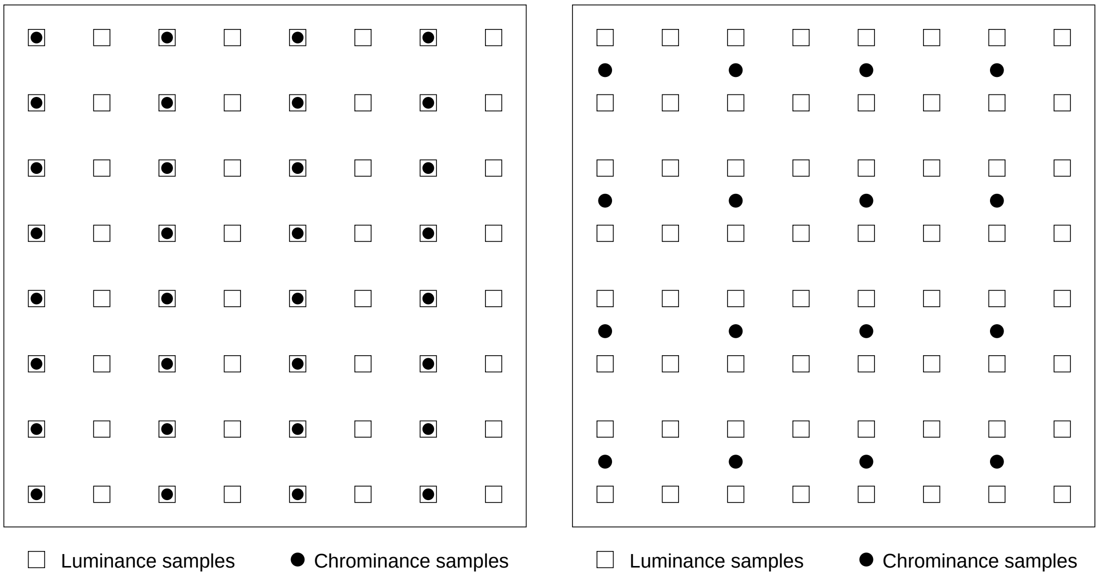
The CCIR-601 [6] format for video frames specifies spatial sampling of 720×480 and temporal sampling at 30 frames/sec for NTSC (U.S. and Japan) television systems and 720×576 at 25 frames/sec for PAL (Europe) television systems. Color is represented using three components: a luminance (Y) component and two chrominance components (Cb and Cr). The luminance component encodes the brightness or intensity of each pixel and the chrominance components encode the color values.[3] Each component is quantized linearly using eight bits. For NTSC (respectively, PAL), there are 720×480 (720×576) luminance values, one for each pixel, and 360×480 (360×576) values for each chrominance component. The chrominance components are subsampled horizontally with respect to the luminance component to take advantage of reduced human sensitivity to color. This subsampling process is referred to as the 4:2:2 format and is depicted in Figure 3.4(a).

The Source Input Format (SIF) specifies spatial sampling of 360×240 (respectively, 360×288) and temporal sampling at 30 (25) frames/sec for NTSC (PAL) television systems. As with CCIR-601, color is represented using three components: Y, Cb, and Cr. Each component is quantized linearly using eight bits. For NTSC (respectively, PAL), there are 360×240 (360×288) luminance values, one for each pixel, and 180×120 (180×144) values for each chrominance component. This subsampling format is referred to as the 4:2:0 format[4] and is depicted in Figure 3.4(b).

One drawback with the CCIR-601 and SIF formats is that they specify different spatial and temporal sampling parameters for NTSC and PAL systems. As its name suggests, the Common Intermediate Format (CIF) was proposed as a bridge between NTSC and PAL. As with CCIR-601, color is represented using three components, each quantized linearly using eight bits. The CIF format uses 4:2:0 color subsampling with an image size of 352×288. Temporal sampling is set at 30 frames/sec. For use with PAL systems, the CIF format requires conversion of the frame rate to 25 frames/sec. For NTSC systems, a spatial resampling may be necessary.

[3]The Y-Cb-Cr color space is related to the red-green-blue (RGB) color space by a matrix multiplication.

[4]This should not be confused with the older 4:1:1 format in which the chrominance components are subsampled by a factor of 4 only in the horizontal direction.



(a) 4:2:2 subsampling (b) 4:2:0 subsampling

Figure 3.4: Color subsampling formats, as specified in the MPEG-2 standard.

For videoconferencing and other low-bit-rate, low-resolution applications, a scaled-down version of CIF called Quarter-CIF (or QCIF) is commonly used. QCIF specifies an image with half the resolution of CIF in each spatial dimension: 176×144. For many low-bit-rate applications, the frame rate is reduced from 30 frames/sec to as low as five frames/sec.

3.2 A Case for Video Compression

Now that we have standard representations for digital video, we can estimate the compression ratio required for typical applications.

For a two-hour movie encoded in NTSC CCIR-601 format, the uncompressed video representation would require about 149 gigabytes to store:

$$\#\text{ bytes} = (720 \cdot 480 + 2 \cdot 360 \cdot 480)\frac{\text{bytes}}{\text{frame}} \cdot 30\frac{\text{frames}}{\text{second}} \cdot 3600\frac{\text{seconds}}{\text{hour}} \cdot 2\text{ hours} = 1.493 \cdot 10^{11}\text{ bytes.}$$

In order to store the movie on one single-sided digital video disk (DVD), which has a capacity of 4.7 gigabytes, we need to compress the video by a factor of about 32:1. To allow room for audio and other auxiliary data (such as text captioning), an even higher compression ratio is needed.

As another example, consider low-bit-rate videoconferencing over a 28.8 kbits/sec modem. Assuming that the uncompressed video is encoded in QCIF format at 10 frames/sec, the uncompressed rate is computed to be:

$$\#\frac{\text{bits}}{\text{second}} = (176 \cdot 144 + 2 \cdot 88 \cdot 144)\frac{\text{bytes}}{\text{frame}} \cdot 8\frac{\text{bits}}{\text{byte}} \cdot 10\frac{\text{frames}}{\text{second}} = 4.055 \cdot 10^{6}\frac{\text{bits}}{\text{second}}.$$

To transmit video in this format over a 28.8 kbits/sec modem would require a compression ratio of 141:1. At such a high compression ratio, depending upon the complexity of the video sequence, the quality of the compressed video may have to be sacrificed. Alternatively, the frame rate could be reduced to increase the image quality, at the expense of increased jerkiness in the motion.

The above examples show why compression is a must for some important digital video applications. For example, without compression, a single-sided DVD can hold less than four minutes of CCIR-601 digital video!

3.3 Lossy Coding and Rate-Distortion

The examples in Section 3.2 show that existing video applications require high compression ratios, over an order of magnitude higher than what is typically possible for the lossless compression methods discussed in Chapters 1 and 2. These high levels of compression can be realized only if we accept some loss in fidelity between the uncompressed and compressed representations. There is a natural tradeoff between the size of the compressed representation and the fidelity of the reproduced images. This tradeoff between rate and distortion is quantified in rate-distortion theory.

3.3.1 Classical Rate-Distortion Theory

Let D be a measure of distortion according to some fidelity criterion and R be the number of bits in a compressed representation of an information source. In classical rate-distortion theory, as pioneered by Claude Shannon [96], a rate-distortion function, $R(D)$, is defined to be the theoretical lower bound on the best compression achievable as a function of the desired distortion D for a given information source, by *any* compressor. In general, the fidelity criterion can be any valid metric; in practice, a squared-error distortion is often used; that is, $D(x, \hat{x}) = (x - \hat{x})^2$.

For a discrete source, $R(0)$ is simply the entropy of the source and corresponds to lossless coding ($D = 0$). In cases where the distortion is bounded above by $D_{\max}$, then $R(D_{\max}) = 0$. Furthermore, it can be shown that $R(D)$ is a non-increasing convex function of D (see, e.g., [21]).

For some specific information sources and distortion measures, closed form expressions for the rate-distortion function have been determined. As an example, for a zero-mean Gaussian source with variance σ^2 and a squared-error distortion measure,

$$R(D) = \begin{cases} \frac{1}{2}\log_2 \frac{\sigma^2}{D}, & 0 \le D \le \sigma^2, \\ 0, & D > \sigma^2. \end{cases}$$

This is plotted for $\sigma = 1$ in Figure 3.5.

3.3.2 Operational Rate-Distortion

In practice, classical rate-distortion theory is not directly applicable to complex encoding and decoding systems since sources are typically not well-characterized and $R(D)$ is difficult, if not impossible, to determine. Even though not directly computable, the existence of a hypothetical rate-distortion function for a given type of information source allows a comparison to be made between competing encoding systems and algorithms.

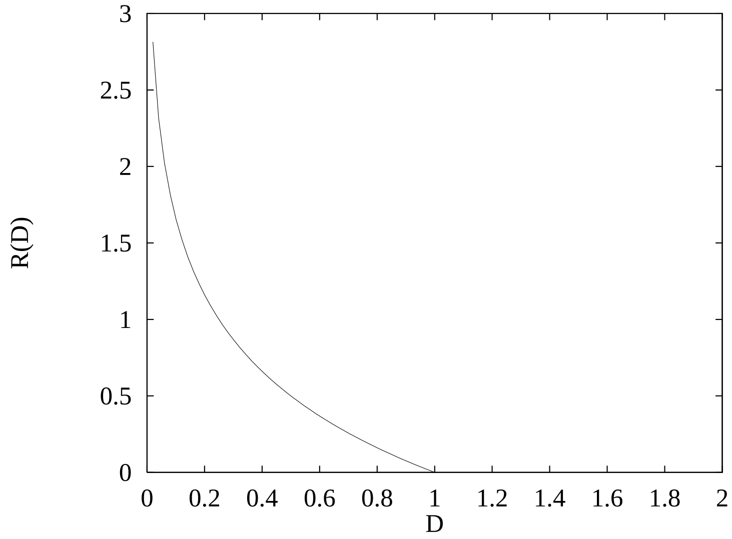


Figure 3.5: Rate-distortion function for a Gaussian source with $\sigma = 1$.

A more practical approach is taken in [14, 98]. By measuring actual rates and distortion achieved by the coder under study, an *operational rate-distortion* plot similar to Figure 3.6 can be constructed. It is sometimes useful to show the convex hull of the data points to fill in the gap between points. Data points are typically generated by varying the level of quantization or other coding parameters under study.

By plotting operational rate-distortion curves for various competing coders, a comparison can be made of their effectiveness. A basic idea is that the more effective and capable a coder is, the closer is its operational rate-distortion curve to the hypothetical rate-distortion function. In Figure 3.7, Coder 1 performs better than Coder 2 for rates greater than about 600 bits per *coding unit*, where a coding unit is a generic term for a block of data. At rates less than 600 bits/unit, Coder 2 performs better.

The mean square error (MSE) distortion measure is commonly used in the literature since it is a convenient measure that lends itself to mathematical analysis using least-mean-square theory. For images and video, however, MSE is not an ideal measure since it is not a good model of human visual perception. For example, in many cases, two encodings with the same MSE can have remarkably different perceptual quality. In keeping with convention, we will assume the use of MSE as the distortion measure, unless otherwise stated.

3.3.3 Budget-Constrained Bit Allocation

A common problem that serves well to illustrate the operational rate-distortion framework is the budget-constrained bit allocation problem. The problem is stated below. Without loss of generality, quantization is the coding parameter to be adjusted.

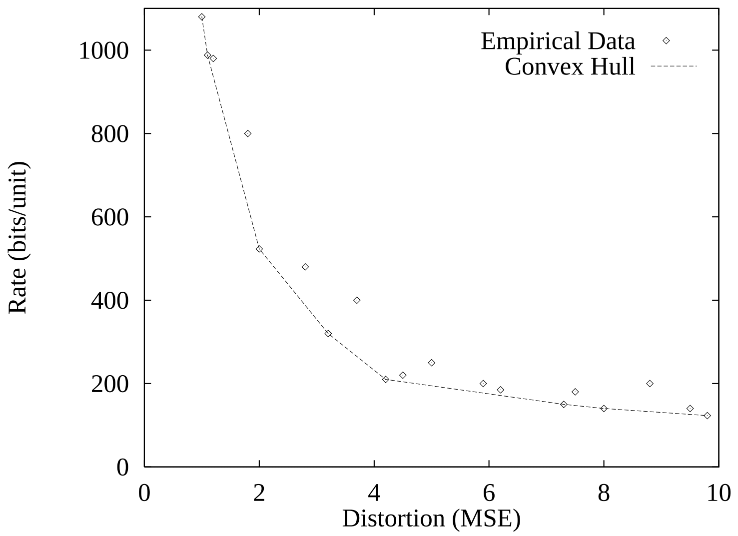


Figure 3.6: Sample operational rate-distortion plot. Plotted is rate versus average distortion.

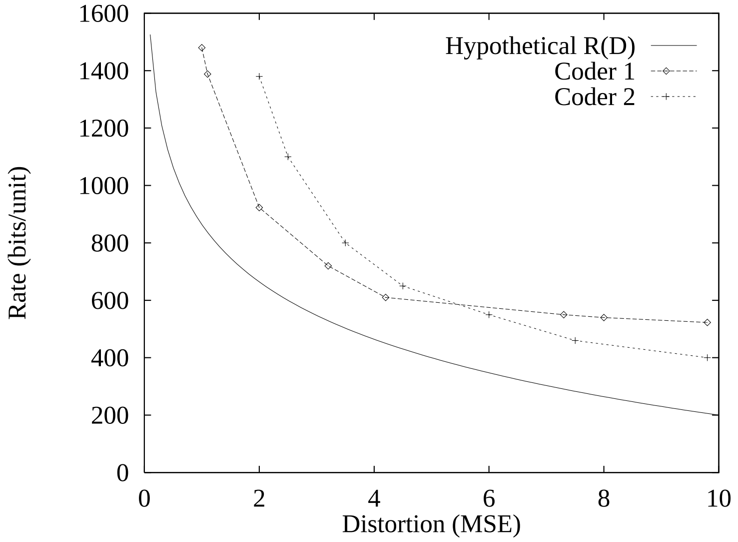


Figure 3.7: Comparison of coders in a rate-distortion framework.

Problem 3.1 *Given a set of quantizers* $\{q_1, q_2, \ldots, q_M\}$, *a sequence of blocks* $\langle x_1, x_2, \ldots x_N \rangle$, *and a target bit budget* B, *determine an assignment of quantizers* $\mathbf{Q} = \langle Q_1, Q_2, \ldots, Q_N \rangle$ *to each block that minimizes a distortion measure* $D(\mathbf{Q})$ *and uses* $R(\mathbf{Q}) \leq B$ *bits.*

3.3.3a Viterbi Algorithm

Problem 3.1 can be solved using a dynamic programming algorithm commonly referred to as the Viterbi algorithm (VA) [28, 105]. Assuming that quantization always produces an integral number of bits, the Viterbi algorithm works by first constructing a trellis of nodes and then finding a shortest path through the trellis. Each node represents a state and each edge a transition between states. For the bit allocation problem, we identify each state with a tuple (b, t, d, p), where t is a time index, b is the total number of bits used in an allocation for the sequence of blocks $\langle x_1, x_2, \ldots, x_t \rangle$, d is the minimum sum distortion for any allocation to those blocks using exactly b bits, and p is a pointer back to a previous state. There is a single start state labeled $(0, 0, 0, 0)$.

Starting with the start state, we construct the trellis by adding an edge for each choice of quantizer and creating a corresponding set of new states. The new states record the number of bits and minimum distortion for all choices of quantizer for coding the first block. There may be more than one edge entering a new state if more than one quantizer results in the same number of bits. However, only the minimum distortion is recorded as d, and p is made to point to a source state that results in the minimum distortion. In case more than one incoming edge produces the minimum distortion, the pointer can point to any of the edges with the minimum distortion. This process is repeated so that new states for time index $k + 1$ are constructed by adding edges corresponding to the quantization of block x_{k+1} to the states with time index k. In the trellis construction, we prune out those states whose bit consumption exceeds the bit budget B. After all the states with time index N have been constructed, we pick a state with time index N that has the minimum distortion. A bit allocation can then be constructed by following the pointers p back from the end state to the start state.

A simple example with $M = 2$ and $N = 3$ is shown in Figure 3.8 to illustrate the Viterbi algorithm. In the example, the shaded node marks a state that exceeds the bit budget B and can be pruned. An optimal path is shown with thick edges. As in this example, there may be more than one path with the minimum distortion.

3.3.3b Lagrange Optimization

Although the Viterbi algorithm finds an optimal solution to Problem 3.1, it is computationally expensive. There could potentially be an exponential number of states generated, on the order of M^N.

In [98], Shoham and Gersho give an efficient bit allocation algorithm based on the Lagrange-multiplier method [25]. In this method, Problem 3.1, a constrained optimization problem, is transformed to the following unconstrained optimization problem.

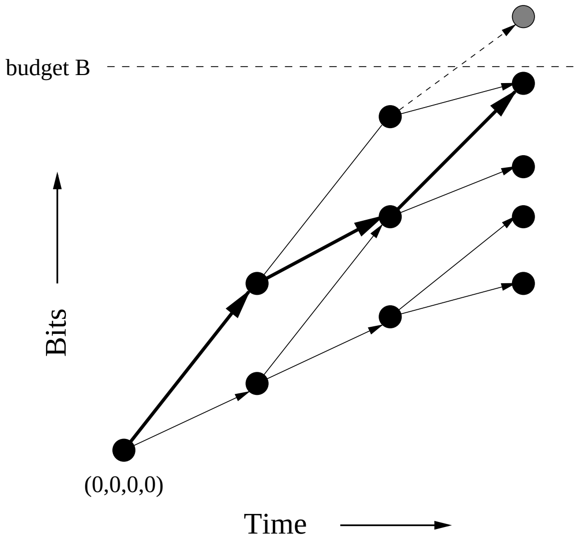


Figure 3.8: Example of a trellis constructed with the Viterbi algorithm. The shaded node marks a state that exceeds the bit budget B and can be pruned. An optimal path is shown with thick edges. Note that there may be more than one path with the minimum distortion.

Problem 3.2 *Given a set of quantizers* $\{q_1, q_2, \ldots, q_M\}$*, a sequence of blocks* $\langle x_1, x_2, \ldots x_N\rangle$*, and a parameter* λ*, determine an assignment of quantizers* $\mathbf{Q} = \langle Q_1, Q_2, \ldots, Q_M\rangle$ *to each block that minimizes the cost function* $C_\lambda(\mathbf{Q}) = D(\mathbf{Q}) + \lambda R(\mathbf{Q})$.

Here, the parameter λ is called the Lagrange multiplier. Let $\mathbf{Q}^*(\lambda)$ denote an optimal solution given λ and $R^*(\lambda) \equiv R(\mathbf{Q}^*(\lambda))$ denote the resulting total number of bits allocated. Note that there may be more than one solution with a given λ. It can be shown that a solution to Problem 3.2 is also a solution to Problem 3.1 when $R^*(\lambda) = B$. This is proved in [25], and we reproduce the theorem and proof as presented in [98].

Theorem 3.1 *For any* $\lambda \geq 0$*, a solution* $\mathbf{Q}^*(\lambda)$ *to Problem 3.2 is also a solution to Problem 3.1 with the constraint* $R(\mathbf{Q}) \leq B$*, where* $B = R^*(\lambda)$.

Proof: For the solution $\mathbf{Q}^*$, we have

$$D(\mathbf{Q}^*) + \lambda R(\mathbf{Q}^*) \leq D(\mathbf{Q}) + \lambda R(\mathbf{Q})$$

for all quantizer allocations $\mathbf{Q}$. Equivalently, we have

$$D(\mathbf{Q}^*) - D(\mathbf{Q}) \leq \lambda(R(\mathbf{Q}) - R(\mathbf{Q}^*))$$

for all quantizer allocations $\mathbf{Q}$. In particular, this result applies for all quantizer allocations $\mathbf{Q}$ belonging to the set

$$S^* = \{\mathbf{Q} : R(\mathbf{Q}) \leq R(\mathbf{Q}^*)\}\,.$$

Since $\lambda \geq 0$ and $R(\mathbf{Q}) - R(\mathbf{Q}^*) \leq 0$ for $Q \in S^*$, we have

$$D(\mathbf{Q}^*) - D(\mathbf{Q}) \leq 0, \qquad \text{for } \mathbf{Q} \in S^*.$$

Therefore $\mathbf{Q}^*$ is a solution to the constrained problem, and the theorem is proved. $\square$

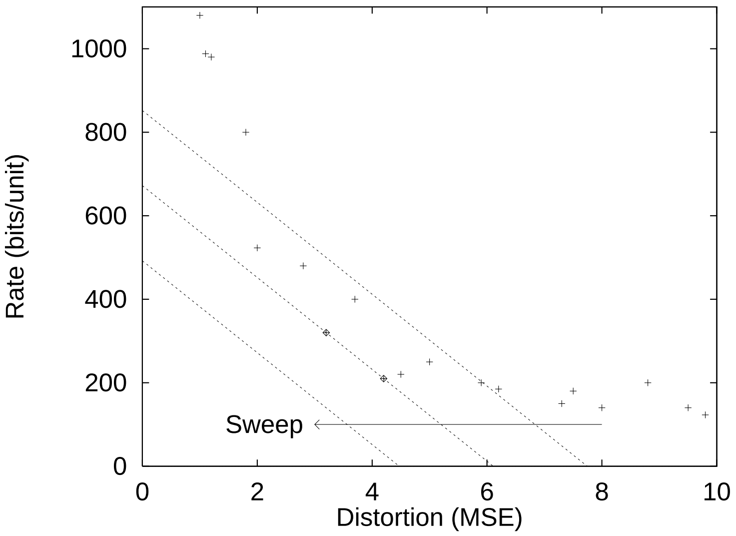


Figure 3.9: Graphical interpretation of Lagrange-multiplier method. Lagrange minimization can be viewed as finding the last point(s) intersected by a right-to-left sweep of a line with slope $-\lambda$. In this example, the two data points circled are found in the minimization.

It should be noted that Theorem 3.1 does not guarantee that, in general, a solution for the constrained Problem 3.1 can be found by solving the unconstrained Problem 3.2. Theorem 3.1 only applies for cases where there is a value for λ such that the number bits used in a solution (there may be more than one for a given λ) to Problem 3.2 is the same as the bit budget B in Problem 3.1.

The Lagrange multiplier λ can be viewed as determining a tradeoff between rate and distortion. A low value for λ favors minimizing distortion over rate, and a high value favors minimizing rate over distortion. In the limit, when $\lambda = 0$, we are minimizing distortion; as $\lambda \to \infty$, we minimize rate. Lagrange optimization can be interpreted graphically as shown in Figure 3.9. The minimization of the Lagrange cost function C_λ can be viewed as finding the last point or points intersected in the rate-distortion plane as a line with slope $-\lambda$ is swept from right to left. In the example shown, there are two such points. From this graphical view, we can easily see that the only points that can be selected with Lagrange optimization are those that lie on the convex hull of the set of all points.

For a given bit budget B, in order to apply the Lagrange-multiplier method, we need to know what value of λ to use. In practice, an iterative search procedure can be used to determine the proper value. The search procedure takes advantage of a useful property of Lagrange optimization: the solution rate $R(\mathbf{Q}^*(\lambda))$ is a non-increasing function of λ. With appropriate initial upper and lower bounds for λ, a bisection search can be performed to find the proper value for λ. Details of the search procedure can be found in [98].

For an additive distortion measure, the distortion $D(\mathbf{Q})$ can be expressed as

$$D(\mathbf{Q}) = \sum_{i=1}^{N} D_i(Q_i),$$

where $D_i(Q_i)$ is the distortion for block i when using the quantizer specified by Q_i. If we assume that the coding of each block is independent of the quantization choices of other blocks, the rate $R(\mathbf{Q})$ can be expressed as

$$R(\mathbf{Q}) = \sum_{i=1}^{N} R_i(Q_i),$$

where $R_i(Q_i)$ is the rate for block i when using the quantizer specified by Q_i. The minimization of C_λ in Problem 3.2 can then be expressed as

$$\begin{aligned} \min_{\mathbf{Q}} C_\lambda(\mathbf{Q}) &= \min_{\mathbf{Q}} \{R(\mathbf{Q}) + \lambda D(\mathbf{Q})\} \\ &= \min_{\mathbf{Q}} \left\{ \sum_{i=1}^{N} R_i(Q_i) + \lambda \sum_{i=1}^{N} D_i(Q_i) \right\} \\ &= \sum_{i=1}^{N} \left(\min_{Q_i} \{R_i(Q_i) + \lambda D_i(Q_i)\} \right). \end{aligned}$$

That is to say, the cost function C_λ can be minimized by minimizing $R_i(Q_i) + \lambda D_i(Q_i)$ separately for each block.

3.4 Spatial Redundancy

Redundancy exists in a video sequence in two forms: spatial and temporal. The former, also called *intraframe* redundancy, refers to the redundancy that exists within a single frame of video, while the latter, also called *interframe* redundancy, refers to the redundancy that exists between consecutive frames within a video sequence.

Reducing spatial redundancy has been the focus of many image compression algorithms. Since video is just a sequence of images, image compression techniques are directly applicable to video frames. Here, we outline some popular image coding techniques applicable to lossy video coding.

3.4.1 Vector Quantization

In vector quantization (VQ) [30], an image is segmented into same-sized blocks of pixel values. The blocks are represented by a fixed number of vectors called *codewords*. The codewords are chosen from a finite set called a *codebook*. This is analogous to the quantization described in Section 3.1.2 except that now quantization is performed on vectors instead of scalar values. The size of the codebook affects the rate (number of bits needed to encode each vector) as well as the distortion; a bigger codebook increases the rate and decreases the average distortion while a smaller codebook has the opposite effects.

With vector quantization, encoding is more computationally intensive than decoding. Encoding requires searching the codebook for a representative codeword for each input vector, while decoding requires only a table lookup. Usually, the same codebook is used by the encoder and the decoder. The codebook generation process is itself computationally demanding. As with lossless dictionary coding, a VQ codebook can be constructed statically, semi-adaptively, or adaptively. Some applications of VQ in video compression can be found in [29, 106].

3.4.2 Block Transform

In block-transform coding, an image is divided into blocks, as with vector quantization. Each block is mathematically transformed into a different representation, which is then quantized and coded. The mathematical transform is chosen so as to "pack" most of the useful information into a small set of coefficients. The coefficients are then selectively quantized so that after quantization most of the "unimportant" coefficients are 0 and can be ignored, while the "important" coefficients are retained. In the decoder, a dequantization process is followed by an inverse transformation.

Block-transform coding can be viewed as an instance of vector quantization where the codebook is determined by the transform and quantization performed. Viewed in this way, for any source, a vector quantizer can be designed that will be at least as good (in a rate-distortion sense) as a particular block transform. A motivation for using block transforms is that for certain block transforms with fast algorithms, encoding can be done faster than full-blown vector quantization. However, with block transforms, decoding has approximately the same complexity as encoding, which is more complex than decoding with vector quantization.

3.4.3 Discrete Cosine Transform

For images, the two-dimensional discrete cosine transform (2D-DCT) is a popular block transform that forms the basis of the lossy JPEG standard [83] developed by the Joint Photographic Experts Group. Because of its success within JPEG, the 2D-DCT has been adopted by many video coding standards as well. We now describe the mathematical basis of the DCT and show how it is applied to code an image.

3.4.3a Forward Transform

The JPEG standard specifies a block size of 8×8 for performing the 2D-DCT. This block size is small enough for the transform to be quickly computed but big enough for significant compression. For an 8×8 block of pixel values $f(i,j)$, the 2D-DCT is defined as

$$F(u,v) = \frac{1}{4} C(u) C(v) \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos \frac{\pi u(2i+1)}{16} \cos \frac{\pi v(2j+1)}{16}, \tag{3.1}$$

where $F(u,v)$ are the transform coefficients and

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & x = 0, \\ 1 & \text{otherwise.} \end{cases}$$

3.4.3b Inverse Transform

To be useful for coding, a block transform needs an inverse transform for purposes of decoding. The two-dimensional inverse discrete cosine transform (2D-IDCT) for an 8×8 block is defined as

$$f(i,j) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} F(u,v) C(u) C(v) \cos \frac{\pi u(2i+1)}{16} \cos \frac{\pi v(2j+1)}{16}. \tag{3.2}$$

$$\begin{bmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{bmatrix}$$

Figure 3.10: Typical quantization matrix applied to 2D-DCT coefficients.

3.4.3c Quantization

Since the DCT and IDCT are transform pairs, they do not result in any compression by themselves. Compression is achieved by subsequent quantization of the transform coefficients.

Quantization as applied to transform coefficients can be viewed as division followed by integer truncation. Specifically, the transform coefficients are first divided by a (prespecified) matrix of integers that is weighted by a *quantization scale*. After division, the results are truncated to integer values. In the dequantization, the quantized values are multiplied by the quantization matrix and adjusted according to the quantization scale. Typically 8 to 12 bits of precision are used.

An example of a quantization matrix is shown in Figure 3.10. The coefficients can be specified to exploit properties of the human visual system. Since the human eye is more sensitive to low spatial frequencies and less sensitive to high spatial frequencies, the transform coefficients corresponding to high spatial frequencies can be quantized more coarsely than those for low spatial frequencies. This selective quantization is shown in Figure 3.10.

3.4.3d Zig-Zag Scan

Because of the coarse quantization of coefficients corresponding to high spatial frequencies, those coefficients are often quantized to 0. An effective way to code the resulting set of quantized coefficients is with a combination of a zig-zag scan of the coefficients as shown in Figure 3.11 and run-length encoding of consecutive zeros. Typically, the DC coefficient, $F(0,0)$, is coded separately from the other coefficients and is not included in the zig-zag scan.

3.5 Temporal Redundancy

Successive frames in a video sequence are typically highly correlated, especially for scenes where there is little or no motion. The spatial decorrelation techniques described in the previous section only operate within a single frame and do not exploit the redundancy that exists between frames. We now review some basic techniques for reducing temporal redundancy.

3.5.1 Frame Differencing

A very simple technique for exploiting temporal redundancy in a video sequence is to code the difference between one frame and the next. This technique is called *frame differencing* and is an

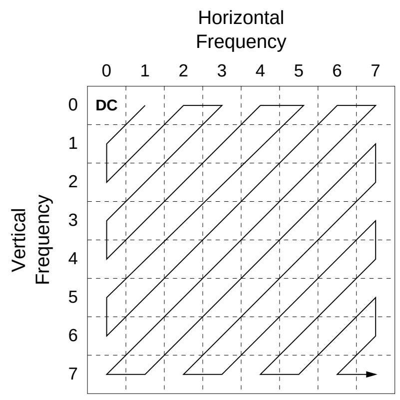


Figure 3.11: Zig-zag scan for coding quantized transform coefficients as an one-dimensional sequence. Run-length encoding of zero values results in efficient coding.

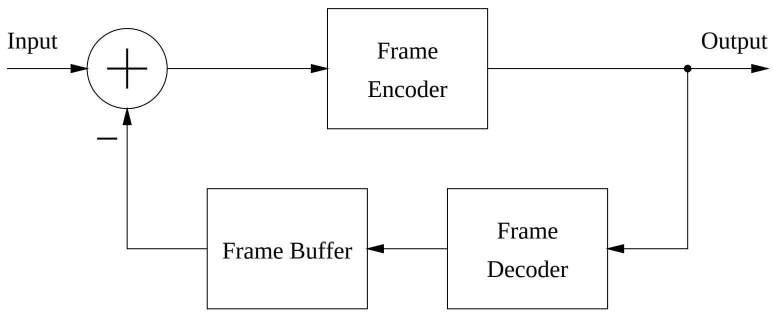


Figure 3.12: Block diagram of a simple frame-differencing coder. The frame buffer stores the previously decoded frame which is used to compute a difference frame.

extension of the basic differential pulse code modulation (DPCM) coding techniques (see, e.g. [76]). A block diagram of an encoder that uses frame differencing is shown in Figure 3.12.

If there is little motion between successive frames, frame differencing yields a difference image that is mostly uniform and can be coded efficiently. However, frame differencing fails when there is appreciable motion between frames or when a scene change occurs.

3.5.2 Motion Compensation

Frame differencing can be viewed as a predictive coding technique where the prediction is simply the previous decoded frame. By improving the prediction, we can potentially obtain better compression. *Motion compensation* is one such technique that uses a model of the motion of objects between frames to form a prediction. Using the model, the encoder performs *motion estimation* to determine the motion that exists between a reference frame and the current frame. The reference frame can occur temporally before the current frame (forward prediction) or after the current frame (backward prediction). An advanced technique, called bidirectional prediction, uses two reference frames, one each for forward and backward prediction, and interpolates the results. This usually gives better prediction and handles the case where an object is temporarily occluded.

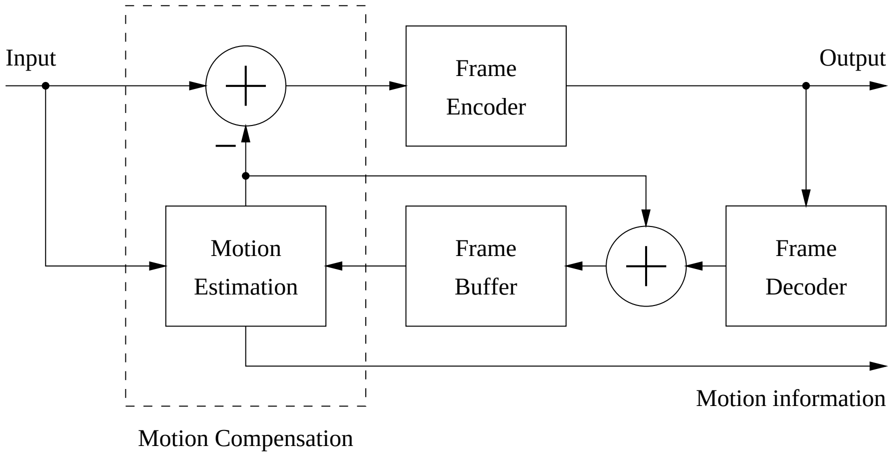


Figure 3.13: Block diagram of a generic motion-compensated video encoder.

The encoding process is illustrated in Figure 3.13. After motion estimation and compensation, the motion information and prediction error are transmitted to the decoder, which reconstructs the predicted frame from the motion information and the decoded reference frame. Note that the reference frame must have already been decoded for the decoder to be able to reconstruct the current frame. At some initial point, a frame must be coded without motion compensation using only spatial coding techniques. Such a frame is commonly referred to as an *intra-coded* frame, or I-frame for short. Because they do not take advantage of interframe redundancy, I-frames consume more bits than predictive frames of comparable quality. To prevent degradation in image quality from the accumulation of prediction error and to allow for easy random access to frames in a video, frames are periodically coded as I-frames.

Frames that are coded using forward prediction are called P-frames, short for *predictive* frames. A P-frame uses as a reference a past I-frame or P-frame.

Backward prediction is typically not used exclusively, but as an option for B-frames, short for *bidirectionally predicted* frames. A B-frame is coded from a past reference frame and a future reference frame, as shown in Figure 3.14. At first, this might seem to present a causality problem since there is a dependence upon a future frame. To avert any such problem, the frames are reordered so that all reference frames that are required by a B-frame or P-frame come before that frame in the reordered sequence. An example is shown in Figure 3.15. In practice, this reordering introduces some encoding and decoding delays and requires two frame buffers to hold the reference frames. For non-real-time applications, such as stored video, the additional delay is not a serious issue. For real-time applications, such as videoconferencing, the distance between successive reference frames are kept small to reduce the delay. B-frames may be omitted altogether to further reduce the delay.

For interframe coding, perceptual weighting as per Figure 3.10 is not usually applied since the block to be coded is the block of *prediction errors*, which does not share the perceptual properties of the original spatial block of pixel values. A quantization matrix with uniform values is typically used for inter-coded blocks.

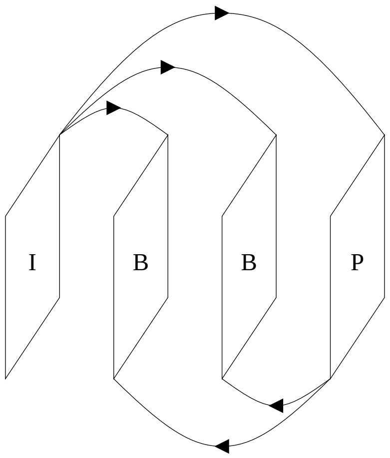


Figure 3.14: Illustration of frames types and dependencies in motion compensation.

Frame Type:	**I**	**B**	**B**	**P**	**B**	**B**	**P**	**B**	**I**
Temporal Index:	1	2	3	4	5	6	7	8	9

(a) Original Sequence (Temporal Order)

Frame Type:	**I**	**P**	**B**	**B**	**P**	**B**	**B**	**I**	**B**
Temporal Index:	1	4	2	3	7	5	6	9	8

(b) Reordered Sequence (Encoding Order)

Figure 3.15: Reordering of frames to allow for causal interpolative coding.

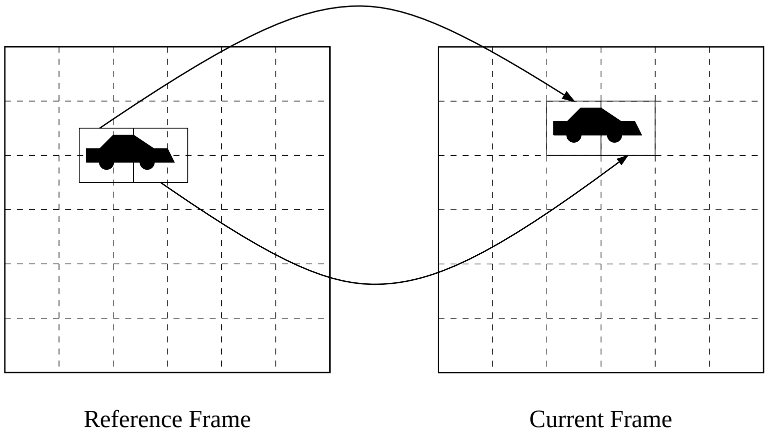


Figure 3.16: Illustration of the block-translation model.

In the final step, indicated in Figure 3.13, the prediction error that results from motion compensation is coded with an intraframe coder, for instance, one of the techniques mentioned in Section 3.4.

3.5.3 Block-Matching

A motion model that is commonly used is the *block-translation* model developed by Jain and Jain [50]. In this model, an image is divided into non-overlapping rectangular blocks. Each block in the predicted image is formed by a translation of a similarly shaped source region from the reference frame. The source region needs not coincide with the block boundaries. This model does not consider any rotation or scaling of the blocks, simplifing the motion estimation procedure at the expense of decreased accuracy. A motion vector may be specified in integer or fractional pixel (pel) increments. Fractional-pel motion compensation involves interpolation of the pixel values in the source block. The block-translation model is illustrated in Figure 3.16. For each block, the encoder transmits a motion vector that specifies the displacement in the translation model.

Motion estimation algorithms using the block-translation model are commonly called block-matching algorithms since the procedure involves matching (regularly-positioned) blocks in the current frame with (arbitrarily-positioned) blocks in the reference frame. Because of its simplicity, block-matching is commonly used with current video coding standards.

3.6 H.261 Standard

In 1990, the International Telegraph and Telephone Consultative Committee (CCITT)[5] approved an international standard for video coding at bit rates of $p \times 64$ kbits/sec, where p is an integer between 1 and 30, inclusive [8, 65]. Officially known as CCITT Recommendation H.261, it is informally called the $p \times 64$ standard and is intended for low-bit-rate applications such as videophone and videoconferencing. We now provide a summary of some key aspects of the standard.

[5] The CCITT has since changed its name to the International Telecommunication Union (ITU-T).

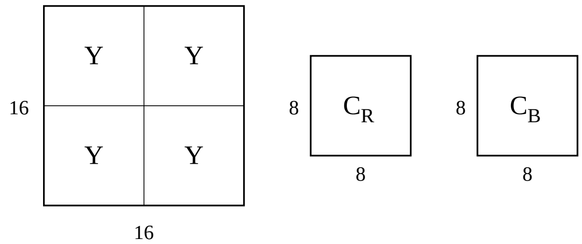


Figure 3.17: Structure of a macroblock.

3.6.1 Features

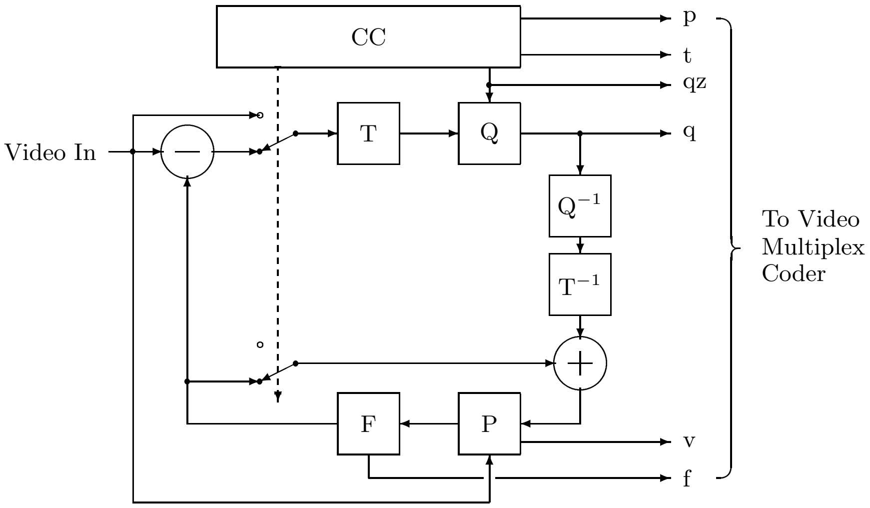
The $p \times 64$ standard uses a combination of block-matching motion compensation (BMMC) and 2D-DCT coding, as described in Sections 3.4.2 and 3.5.3. Since $p \times 64$ is intended for real-time videoconferencing applications, there is a requirement for low encoding delay. This precludes the use of bidirectional predictive motion compensation. Therefore only intraframe coding and forward predictive coding are used, with a predicted block depending only upon the previous frame. The real-time requirement also restricts the complexity of higher-level algorithms, such as motion estimation and rate control.

The Common Intermediate Format (CIF) and Quarter-CIF (QCIF), described in Section 3.1.3, are specified for video frames. A video frame is divided into Groups of Blocks (GOB) made up of a number of macroblocks (MB). As depicted in Figure 3.17, each macroblock is composed of four 8×8 luminance blocks and two 8×8 chrominance blocks, one each for the Cb and Cr color components. Integer-pel motion compensation is performed at the macroblock level; that is, there is one motion vector per macroblock.

3.6.2 Encoder Block Diagram

A block diagram of a basic $p \times 64$ coder is shown in Figure 3.18. At a high level, the basic encoding process works as follows: The encoder first decides whether to code a macroblock M using intraframe or interframe coding. For intraframe coding, the techniques outlined in Section 3.4.3 are used. If interframe coding is selected, the encoder performs motion estimation to choose a motion vector $\vec{v}$ (how this is done is left unspecified in the standard). If the previous macroblock is intra-coded, $\vec{v}$ is transmitted using a static Huffman code, otherwise the difference between $\vec{v}$ and the motion vector for the previous macroblock is sent using a static Huffman code. For each 8×8 block B contained in M, a lossy version of the block of prediction errors obtained by using $\vec{v}$ to predict B is then transmitted. This is done by applying the 2D-DCT to the block of prediction errors, quantizing and scanning the transform coefficients, and encoding the results using a run-length/Huffman coder, as prescribed in Section 3.4.3.

The encoder has the option of changing certain aspects of the above process. First, the encoder may simply not transmit the current macroblock; the decoder is then assumed to use the corresponding macroblock in the previous frame in its place. If motion compensation is used, there is an option to apply a linear filter to the previous decoded frame before using it for prediction.



T: Transform
Q: Quantizer
P: Picture Memory with motion-compensated variable delay
F: Loop Filter
CC: Coding Control

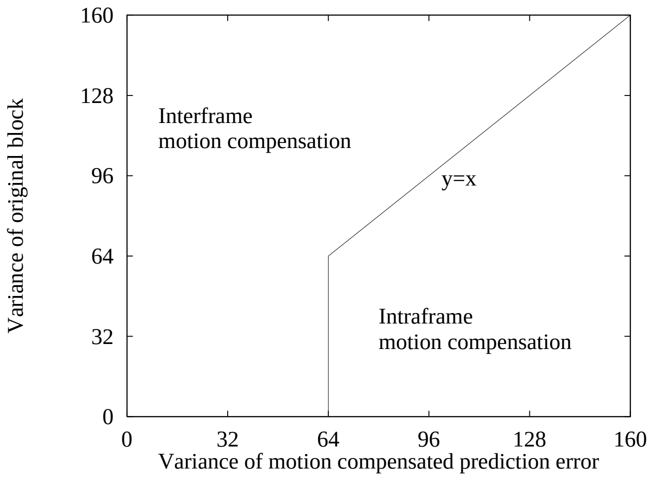
p: Flag for INTRA/INTER
t: Flag for transmitted or not
qz: Quantizer indication
q: Quantizing index for transform coefficients
v: Motion vector
f: Switching on/off of the loop filter

Figure 3.18: Block diagram of a typical $p \times 64$ source coder [8].

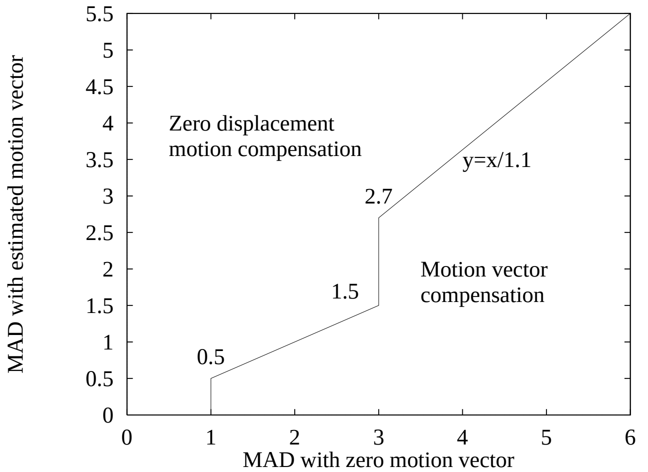
3.6.3 Heuristics for Coding Control

The $p \times 64$ standard does not specify how to make coding decisions. However, to aid in the evaluation of different coding techniques, the CCITT provides an encoder simulation model called Reference Model 8 (RM8) [7]. Motion estimation is performed to minimize the mean absolute difference (MAD) of the prediction errors. A fast three-step search, instead of an exhaustive full-search, is used for motion estimation. RM8 specifies several heuristics used to make the coding decisions.

The variance V_{P} of the prediction errors for the luminance blocks in M after motion compensation using $\vec{v}$ is compared against the variance V_{Y} of the original luminance blocks in M to determine whether to perform intraframe or interframe coding. The intra/inter decision diagram, as specified in RM8, is plotted in Figure 3.19(a). If interframe motion compensation mode is selected, the decision of whether to use motion compensation with a zero motion vector or with the estimated motion vector is made by comparing the MAD of motion compensation with zero motion against that with the estimated motion vector. If the zero motion vector is chosen, this is indicated by a special coding mode and no motion vector is sent. The decision diagram, as recommended in [7], are shown in Figure 3.19(b). The loop filter is enabled if a non-zero motion vector is used. The decision of whether to transmit the block-transform coefficients is made individually for each block in a macroblock by considering the values of the quantized transform coefficients. If all the coefficients are zero for a block, they are not transmitted for that block.



(a) Intraframe/interframe decision



(b) Motion vector decision

Figure 3.19: Heuristic decision diagrams for coding control from Reference Model 8 [7].

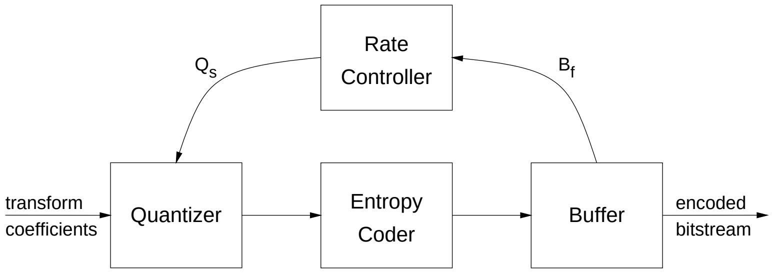


Figure 3.20: Block diagram of rate control in a typical video coding system.

3.6.4 Rate Control

Video coders often have to operate within fixed bandwidth limitations. Since the $p \times 64$ standard uses variable-length entropy coding of quantized transform coefficients and side information, resulting in a variable bit rate, some form of *rate control* is required for operation on bandwidth-limited channels. For example, if the coder's output exceeds the channel capacity, then frames could be dropped or the quality decreased in order to meet the bandwidth constraints. On the other hand, if the coder's output is well below the channel's capacity, the quality and/or frame-rate can be increased to better utilize the channel.

A simple technique for rate control that is specified in RM8 uses a buffered encoding model as shown in Figure 3.20. In this model, the output of the encoder is connected to a buffer whose purpose is to even out the fluctuations in bit rate. By monitoring the fullness of the buffer, the rate controller can adjust the quantization scale Q_s, which affects the encoder's bit rate, to prevent the buffer from underflowing or overflowing. In the model, the buffer is defined for the purpose of regulating the output bit rate and may or may not correspond to an actual encoder buffer.

RM8 gives some parameters and prescriptions for the rate control process. The size of the buffer is specified to be $p \times 6.4$ kbits, which translates to a maximum buffering delay of 100 ms. For purposes of rate control, the first frame is coded using a fixed quantization scale that is computed from the target bit rate. After the first frame is coded, the buffer is reset to be half full. The quantization scale Q_s is determined from the buffer fullness B_f using the formula:

$$Q_\mathrm{s} = \min(\lfloor 32 B_\mathrm{f} \rfloor + 1, 31),$$

where Q_s has a integral range of $[1, 31]$, and B_f is normalized to have a real-valued range of $[0, 1]$. This feedback function is plotted in Figure 3.21. The quantization scale is adjusted once for each GOB (11 macroblocks in RM8).

3.7 MPEG Standards

In 1988, the International Standards Organization (ISO) formed the Moving Pictures Expert Group (MPEG), with the formal designation ISO-IEC/JTC1 SC29/WG11, to develop standards for the digital encoding of moving pictures (video) and associated audio. In 1991, the MPEG committee completed its first international standard, MPEG-1 [47, 57], formally ISO 11172.

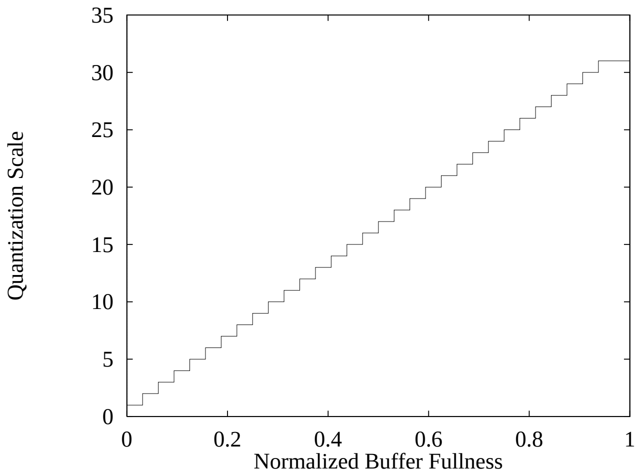


Figure 3.21: Feedback function controlling quantization scale based on buffer fullness.

As a generic video coding specification, MPEG-1 supports multiple image formats, including, CIF, SIF, and QCIF. Image sizes up to $4,095 \times 4,095$ are supported. However, only progressive scan and 4:2:0 color subsampling are supported. While MPEG-1 proved successful for the computer entertainment industry, its lack of support for interlaced scan prevented its use in digital television.

In 1990, the MPEG committee started work on MPEG-2 [49], formally ISO 13818. MPEG-2 is an extension of MPEG-1 that remedies several major shortcomings of MPEG-1 by adding support for interlaced video, more color subsampling formats, and other advanced coding features. To leverage existing MPEG-1 titles and to promote its adoption, MPEG-2 retains backward compatibility with MPEG-1.

As an international standard, MPEG-2 is gaining success. For example, it is being considered as a component in several proposals for high definition television (HDTV); it is currently used in direct digital satellite broadcast and is part of standards for digital video broadcast (DVB); and it is specified as the video compression technology for use with the upcoming digital video disk (DVD).

3.7.1 Features

As with the H.261 standard, the MPEG standards specify a syntax for the coded bitstream and a mechanism for decoding the bitstream. Not covered by the standard are details about the encoding process, thus allowing for flexibility and innovation in encoder design and implementation.

Like H.261, the MPEG standards employ a hybrid video coding scheme that combines BMMC with 2D-DCT coding. Unlike H.261, the MPEG standards allow for bidirectional prediction in addition to intraframe coding and forward prediction, all of which are described in Section 3.5. Additionally, the MPEG standards support motion compensation at half-pel accuracy to allow for better image quality at the expense of additional computation. By supporting advanced coding techniques, the MPEG standards allow an encoding system to trade off between computation and image quality. This flexibility can be a great advantage for non-real-time encoding systems that

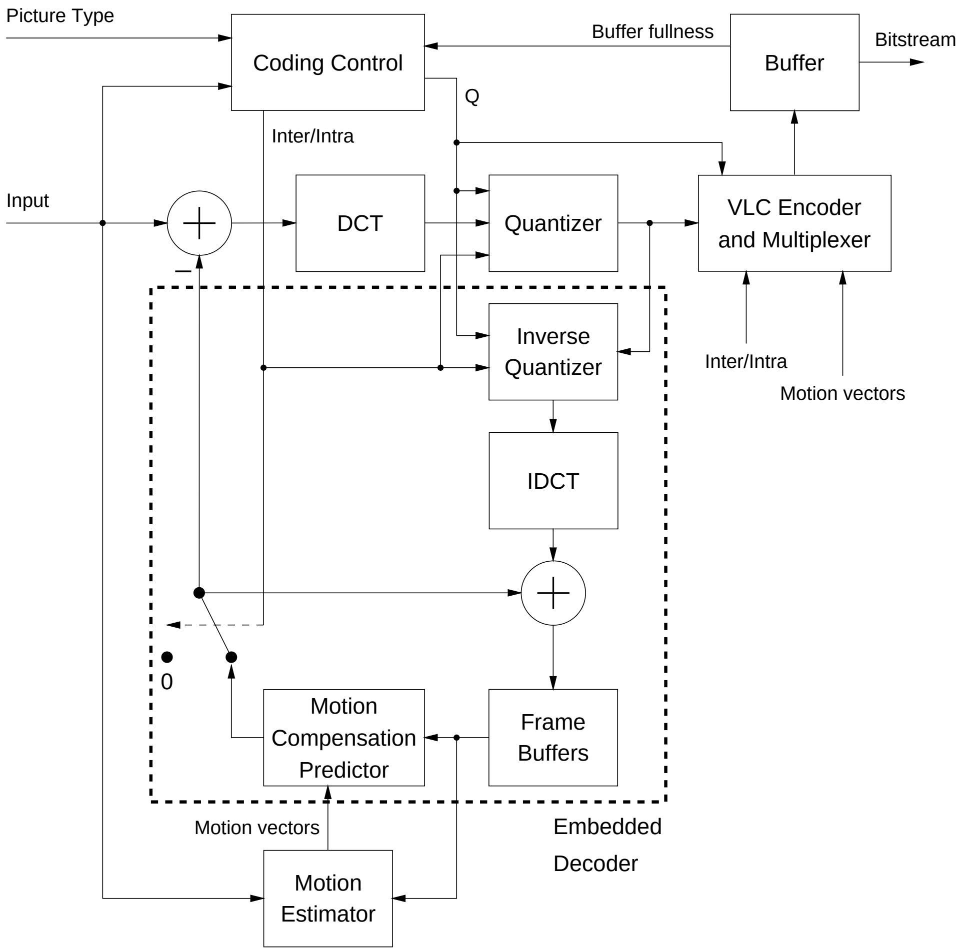


Figure 3.22: Block diagram of a typical MPEG encoder.

can be afforded time to code a video sequence well, especially for applications such as movies and commercials, where the quality of the coded video is of utmost importance since the video will be played many times. However, by using a subset of the coding features, the MPEG standards can also be used for real-time applications such as videoconferencing, news, and other live broadcasts.

3.7.2 Encoder Block Diagram

A basic encoder block diagram is shown in Figure 3.22, with the embedded decoder highlighted. The structure of the encoder is very similar to that in Figure 3.18. The main differences, as outlined above, are hidden in the Coding Control, Motion Estimation, and Motion Compensation blocks.

3.7.3 Layers

In the syntax of the MPEG standards, a video sequence is partitioned into a hierarchy of *layers*. The presence of a layer in the bitstream is indicated by a *start code* indicating the layer type followed by a *header* that specifies parameters for that layer. At the top of the hierarchy is the

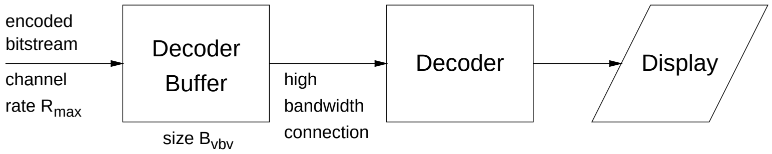


Figure 3.23: Block diagram of the MPEG Video Buffering Verifier.

sequence layer. The sequence header specifies information for the entire video sequence, such as the frame size, frame rate, bit rate, and quantization matrix . Below the sequence layer is the *group of pictures* (GOP) layer. A GOP is structured as a set of contiguous frames that contains at least one I-frame. A GOP can start with either a B-frame or an I-frame and end with either an I-frame or a P-frame. The GOP structure is designed to support random access, indexing, and editing. An example of a GOP unit can be found in Figure 3.14(a). If a GOP begins with a frame that does not depend upon a preceding frame, it can be decoded and displayed independently of the previous GOP and is called a *closed* GOP. GOP's that are not closed are referred to as *open*.

Below the GOP layer is the *picture* layer. The picture header contains information about each picture[6] coded, such as picture type (I, P, or B) and temporal reference. A picture is divided into *slices*, which consists of a segment of consecutive *macroblocks*. Dividing a picture into slices limits the effects of transmission errors and allows the decoder to recover from these errors.

A macroblock consists of a number of 8×8 *blocks* of intensity and chrominance values. The number of blocks in a macroblock depends upon the color subsampling scheme used (see Figure 3.4). For 4:2:0 subsampling, the structure of a macroblock is shown in Figure 3.17. For 4:2:2 subsampling, a macroblock contains four Y blocks, two Cb blocks, and two Cr blocks. MPEG-2 supports an additional color subsampling mode, 4:4:4, in which the Cb and Cr color components have the same spatial resolution as the luminance component Y. Thus for 4:4:4 color subsampling, a macroblock consists of a total of twelve blocks.

As with JPEG and H.261, the 8×8 block is the basic unit for DCT coding and quantization.

3.7.4 Video Buffering Verifier

In addition to defining a bitstream syntax and decoding process, the MPEG video standards define an idealized decoder model called the *Video Buffering Verifier* (VBV). The purpose of the VBV is to put quantifiable limits on the variability in the coding rate such that an encoded bitstream can be decoded with reasonable buffering requirements. As diagrammed in Figure 3.23, the VBV consists of a decoder buffer, a decoder, and a display unit. The decoder buffer stores the incoming bits for processing by the decoder. At regular display intervals, the decoder *instantaneously* removes, decodes, and displays the earliest picture in the buffer. It should be stressed that the VBV is only a idealized decoder model and not a prescription of how to build a decoder or how an actual decoder would function. The VBV model, however, is useful in establishing rate constraints on encoded video such that the encoding would be decodable with specified buffering requirements.

[6]With progressive scan, a *picture* in MPEG's terminology is equivalent to what we have been calling a *frame*. With interlaced scan, a picture may refer to a single field.

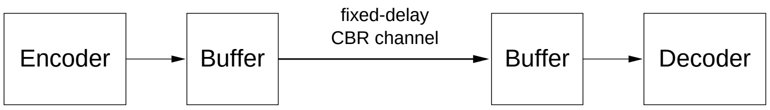


Figure 3.24: Block diagram of a fixed-delay CBR video transmission system.

The VBV has three prescribed modes of operation: a *constant bit rate* (CBR) mode and two *variable bit rate* (VBR) modes. MPEG-1 supports only the CBR mode while MPEG-2 supports all three modes. In CBR mode, bits enter the decoder buffer at a constant rate $R_{\max}$ as specified in the sequence header. Initially, the buffer is empty and fills for a prespecified amount of time before bits for the first picture are removed and decoded. Afterwards, the buffer continues to fill at the channel rate $R_{\max}$ while the decoder removes bits for coded pictures at regular display intervals. The CBR mode models operation of a decoder connected to a constant-bit-rate channel with a fixed channel delay, as shown in Figure 3.24.

The amount of time that the start code for a given picture is to reside in the VBV buffer before that picture is decoded is specified in the picture header with a parameter called **vbv_delay**.[7] In CBR mode, **vbv_delay** is related to the VBV buffer fullness B_{f} in the following manner:

$$\textbf{vbv\_delay} = \frac{90000 \cdot B_{\text{f}}}{R_{\max}}.$$

In the first VBR mode, the compressed bits for picture n enter the buffer at a constant rate $R(n)$ that may vary from picture to picture, up to the maximum rate $R_{\max}$ specified in the sequence header. The relationship between $R(n)$ and **vbv_delay** is as follows:

$$R(n) = \frac{s_n}{\tau(n) - \tau(n+1) + t(n+1) - t(n)},$$

where

$$\begin{aligned}
R(n) &= \text{ the rate (in bits/sec) at which bits for picture } n \text{ enter the VBV buffer,} \\
s_n &= \text{ the number of bits for picture } n, \\
\tau(n) &= \text{ the decoding delay (in seconds) coded in } \textbf{vbv_delay} \text{ for picture } n, \text{ and} \\
t(n) &= \text{ the time (in seconds) when the } n\text{th picture is removed from the VBV buffer.}
\end{aligned}$$

In the second VBR mode, **vbv_delay** is set to 65535 for all pictures. The VBV buffer is initially filled to capacity at the peak rate $R_{\max}$ before the first picture is removed. Thereafter, in each display interval, bits enter the buffer at the peak rate until the buffer is full, at which point bits stop entering the buffer until the next picture has been decoded. When the buffer is full, bits are not discarded, however. The channel is assumed to be able to hold the bits until needed by the VBV. For stored-video applications, this requirement can be met using the double-buffering scheme shown in Figure 3.25, for example. With a maximum disk latency of T_L, a buffer size of $T_L R_{\max}$ is sufficient to guarantee timely delivery of bits to the VBV.

Since the decoder buffer stops receiving bits when it is full, a potentially variable number of bits can enter the buffer in each display period. The second VBR mode can be thought of as modeling

[7]The parameter **vbv_delay** is coded as an integer in the range $[0, 65534]$ and is expressed in units of 1/90000 sec.

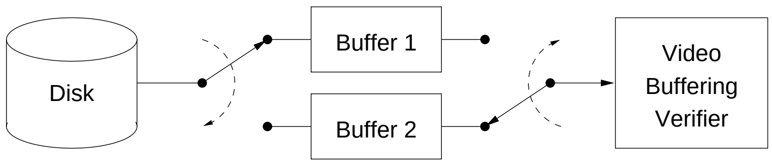


Figure 3.25: Block diagram of a stored-video system using double buffering.

the operation of a decoder connected to a channel or device (a disk drive for example) that can transfer data at a variable rate up to a peak rate R_{max}.

For proper operation in any mode, the decoder buffer should not exceed its capacity B_{VBV} as specified in the sequence header.[8] Also, the buffer should contain at least the number of bits needed to decode the next picture at the time it is to be decoded. As will be shown in Chapter 6, these requirements impose constraints on the number of bits that the encoder can produce for each picture.

A compliant encoder must produce a bitstream that results in proper operation of the VBV. The VBV is not intended to be a prescription for an actual decoder implementation. However, a compliant decoder implementation should be able to decode successfully (within resource limits) any bitstream that meets the VBV buffering constraints.

3.7.5 Rate Control

As with H.261, the MPEG standards do not specify how to perform rate control. To allow for testing and experimentation using a common set of encoder routines, MPEG created a series of test models. Here, we describe the rate control strategy outlined in the MPEG-2 Test Model 5 (TM5) [48]. In TM5, rate control is broken down into three steps:

1. **Target bit allocation**. In this step, the complexity of the current picture is estimated based on the encoding of previous pictures to allocate a number of bits to code the picture.

2. **Rate control**. A reference quantization scale is determined using a virtual buffer in a feedback loop to regulate the coding rate so as to achieve the target bit allocation.

3. **Adaptive quantization**. The reference quantization scale is modulated according to the spatial activity in each macroblock to determine the actual quantization scale with which to code the macroblock.

Target Bit Allocation. The number of bits to be allocated to a picture depends upon its type: I, P, or B. For each picture type, there is a complexity model that attempts to relate the number of bits that would result from coding a picture of a given type to the quantization scale used. The complexity models are of the form

$$S_{\mathrm{i}} = \frac{X_{\mathrm{i}}}{Q_{\mathrm{i}}}, \qquad S_{\mathrm{p}} = \frac{X_{\mathrm{p}}}{Q_{\mathrm{p}}}, \qquad S_{\mathrm{b}} = \frac{X_{\mathrm{b}}}{Q_{\mathrm{b}}},$$

[8] By definition, this requirement is always met in second VBR mode.

where S, X, and Q denote number of bits, complexity, and quantization scale, respectively; and the subscript indicate the picture type.

Initially the complexity values are set to:

$$X_{\mathrm{i}} = \frac{160 \cdot \textbf{bit\_rate}}{115}, \qquad X_{\mathrm{p}} = \frac{60 \cdot \textbf{bit\_rate}}{115}, \qquad X_{\mathrm{b}} = \frac{42 \cdot \textbf{bit\_rate}}{115},$$

where **bit_rate** is measured in bits/sec.

After a picture of a given type is coded, its associated complexity model is updated based on the average quantization scale used and number of bits produced:

$$X_{\mathrm{i}} = S_{\mathrm{i}} Q_{\mathrm{i}}, \qquad X_{\mathrm{p}} = S_{\mathrm{p}} Q_{\mathrm{p}}, \qquad X_{\mathrm{b}} = S_{\mathrm{b}} Q_{\mathrm{b}}.$$

Bit allocation is performed with the goal that the average bit rate is achieved at the GOP layer. A corresponding number of bits is assigned to code each GOP. Bits are allocated to each picture in a GOP based on the complexity models, the number of bits available to code the remaining pictures in the GOP, and the number of remaining I, P, and B pictures in the GOP. Let N be the total number of pictures in a GOP. Let N_{i}, N_{p}, and N_{b} be the number of remaining I, P, and B pictures, respectively. The bit target for each type of picture is calculated according to

$$\begin{aligned}
T_{\mathrm{i}} &= \max\left\{\frac{R}{1 + \frac{N_{\mathrm{p}} X_{\mathrm{p}}}{K_{\mathrm{p}} X_{\mathrm{i}}} + \frac{N_{\mathrm{b}} X_{\mathrm{b}}}{K_{\mathrm{b}} X_{\mathrm{i}}}}, \frac{\textbf{bit_rate}}{8 \cdot \textbf{picture_rate}}\right\} \\
T_{\mathrm{p}} &= \max\left\{\frac{R}{N_{\mathrm{p}} + \frac{N_{\mathrm{b}} K_{\mathrm{p}} X_{\mathrm{b}}}{K_{\mathrm{b}} X_{\mathrm{p}}}}, \frac{\textbf{bit_rate}}{8 \cdot \textbf{picture_rate}}\right\} \\
T_{\mathrm{b}} &= \max\left\{\frac{R}{N_{\mathrm{b}} + \frac{N_{\mathrm{p}} K_{\mathrm{b}} X_{\mathrm{p}}}{K_{\mathrm{p}} X_{\mathrm{b}}}}, \frac{\textbf{bit_rate}}{8 \cdot \textbf{picture_rate}}\right\}
\end{aligned}$$

where K_{p} and K_{b} are constants that depend upon the quantization matrices,[9] and R is the number of bits available to code the remaining pictures in the GOP.

After all the pictures in a GOP have been coded, any difference between the target and actual bit allocation is carried over to the next GOP.

Rate Control. Given a target bit allocation for a picture, a virtual encoding buffer is used to determine a reference quantization scale similar to the procedure in Section 3.6.4 for the $p \times 64$ standard. A separate virtual buffer is maintained for each picture type. The reference quantization scale is computed for each macroblock from the buffer fullness as:

$$Q_j = \frac{31 \cdot d_j}{r},$$

where Q_j is the reference quantization for macroblock j, d_j is the buffer fullness, and r is a *reaction parameter* given by

$$r = \frac{2 \cdot \textbf{bit\_rate}}{\textbf{picture\_rate}}.$$

[9]For the matrices specified in TM5, $K_{\mathrm{p}} = 1.0$ and $K_{\mathrm{b}} = 1.4$.

Adaptive Quantization. The rate control step provides a reference quantization scale to code each macroblock. The reference quantization scale is modulated with an *activity factor* that is determined from a measure of the spatial activity of the macroblock. The rationale is that a macroblock that has little spatial activity, such as a smooth region, should be quantized finer than a macroblock with high spatial activity, such as a textured region, since quantization error is typically more noticeable in smooth regions than in textured regions. This is an attempt at equalizing perceptual quality for a given quantization scale.

For macroblock j, an activity factor act_j is computed as one plus the minimum of the variances of the luminance blocks within the macroblock. A normalized activity factor is then computed based on the average of the activity factor of the last encoded picture:

$$N_{\text{act}} = \frac{2 \cdot \text{act}_j + \textbf{avg\_act}}{\text{act}_j + 2 \cdot \textbf{avg\_act}}.$$

For the first picture, $\textbf{avg\_act} = 400$. The actual quantization scale Q_{s} used to code the macroblock is computed as:

$$Q_{\text{s}} = \min(Q_j \cdot N_{\text{act}}, 31).$$

It should be noted that the TM5 rate control strategy does not take into account the VBV and therefore does not guarantee VBV compliance. Also, TM5 only performs rate control for CBR operation and not for VBR.

Chapter 4

Motion Estimation for Low-Bit-Rate Video Coding

4.1 Introduction

As described in the previous chapter, hybrid video coding that combines block-matching motion compensation (BMMC) with transform coding of the residual is a popular scheme for video compression, adopted by international standards such as H.261 [8, 65] and the MPEG standards [47, 49, 57]. Motion compensation is a technique that exploits the typically strong correlation between successive frames of a video sequence by coding *motion vectors* that tell the decoder where to look on the previous frame for predictions of the intensity of each pixel in the current frame. With BMMC, the current frame is divided into blocks (usually 8×8 or 16×16) whose pixels are assigned the same motion vector $\vec{v}$. The residual from motion compensation is then coded with a lossy transform coder, such as the 2D-DCT, followed by a variable-length entropy coder.

In previous work on BMMC, motion vectors are typically chosen to minimize prediction error, and much of the emphasis has been on speeding up the motion search [50, 55, 88, 100]. However, for low bit-rate applications, the coding of motion vectors takes up a significant portion of the bandwidth, as evidenced with a coding experiment summarized in Table 4.1 and Figure 4.1. This observation has previously been made in [58]. In this chapter, we investigate the use of cost measures that take into account the effects of the choice of motion vector on rate and distortion. We first develop and present computationally intensive coders that attempt explicitly to optimize for rate and distortion. Insights from these implementations lead to faster coders that minimize an efficiently computed heuristic function. Experiments show that using these measures yields substantially better rate-distortion performance than measures based solely upon prediction error.

We implemented and tested our motion estimation algorithms within the H.261 standard, also known informally as the $p \times 64$ standard. The $p \times 64$ standard is intended for applications like videophone and videoconferencing, where low bit rates are required, not much motion is present, and frames are to be transmitted essentially as they are generated. Our experimental results are for benchmark videos typical of the type for which the $p \times 64$ standard was intended: they consist

Quantizer Step Size	Bits/Frame	MV Bits	Side Bits	DCT Bits
31	1793.92	694.24	1002.86	96.82
30	1803.66	692.26	1004.92	106.48
29	1808.18	694.28	1004.88	109.02
28	1814.78	677.92	1008.04	128.82
27	1822.32	676.98	1009.22	136.12
26	1826.88	668.20	1008.62	150.06
25	1833.42	668.30	1010.08	155.04
24	1867.10	662.98	1015.84	188.28
23	1875.54	664.44	1013.26	197.84
22	1920.16	659.14	1018.24	242.78
21	1924.52	652.84	1019.76	251.92
20	1987.80	654.08	1025.48	308.24
19	1997.84	640.94	1027.58	329.32
18	2062.24	635.34	1033.54	393.36
17	2095.80	626.74	1037.14	431.92
16	2198.60	619.34	1045.66	533.60
15	2243.52	614.54	1050.12	578.86
14	2374.58	591.92	1061.48	721.18
13	2442.98	592.34	1065.44	785.20
12	2666.88	556.58	1083.72	1026.58

Table 4.1: Distribution of bits for intraframe coding of the Miss America sequence at various bit rates with a standard $p \times 64$ coder. All quantities are averaged over the entire sequence.

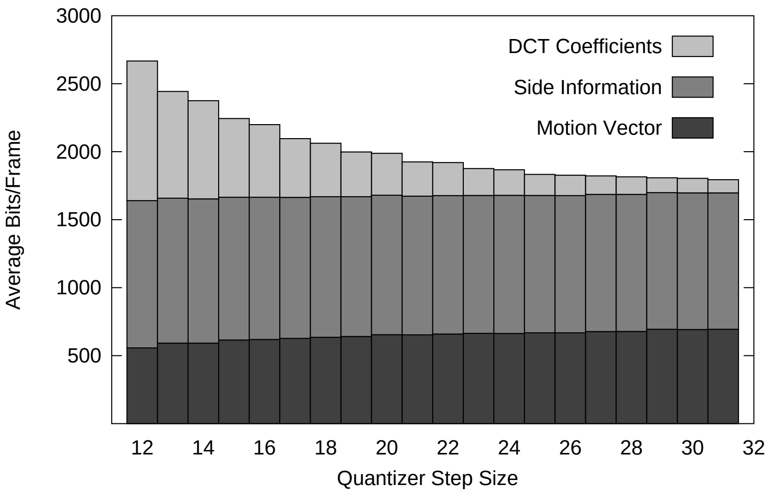


Figure 4.1: Distribution of bits for intraframe coding of the Miss America sequence at various bit rates with a standard $p \times 64$ coder.

of a "head-and-shoulders" view of a single speaker.

In the next section, we briefly describe an existing implementation of the $p \times 64$ standard that we use as a basis for comparison. We then show how to modify the base implementation, but remain within the $p \times 64$ standard, to choose motion vectors that more directly minimize rate and distortion. Experiments show that when transmitting two benchmark QCIF video sequences, `Miss America` and `Claire`, at 18 kbits/sec using rate control, choosing motion vectors explicitly to minimize rate improves average PSNR by 0.87 dB and 0.47 dB respectively. In the $p \times 64$ standard, two binary coding decisions must be made from time to time.[1] In the base implementation, heuristics based upon prediction error are used to make these decisions. When bit minimization is also applied to make the coding decisions, the improvement in PSNR becomes a significant 1.93 dB for `Miss America` and 1.35 dB for `Claire`. If instead of minimizing the bit rate, we minimize a combination of rate and distortion, we observe improvements of 2.09 dB and 1.45 dB respectively.

In Section 4.4 we describe coders that minimize a heuristic function of the prediction error and motion vector code-length. These heuristic coders give compression performance comparable to the explicit minimization coders while running much faster. Experimental results are presented in Sections 4.3.4 and 4.4.2.

Preliminary descriptions of this work can be found in [37, 38, 39, 40].

4.2 PVRG Implementation of H.261

As a basis for comparing the different motion estimation schemes proposed in this chapter, we use the $p \times 64$ coder supplied by the Portable Video Research Group (PVRG) [45]. The block diagram for a basic $p \times 64$ coder is shown in Figure 3.18.

In the base PVRG implementation, a motion vector $\vec{v}$ is determined for each macroblock M using standard full-search block-matching. Only the luminance blocks are compared to the determine the best match, with the mean absolute difference (MAD) being used as the measure of prediction error. Decisions on how to code individual blocks are made according to Reference Model 8 [7], as described in Section 3.6.3.

4.3 Explicit Minimization Algorithms

In the PVRG coder, motion estimation is performed to minimize the MAD of the prediction error. A rationale for this is that minimizing the mean square error (MSE) of the motion-compensated prediction, which is approximated with the MAD, is equivalent to minimizing the variance of the 2D-DCT coefficients of the prediction error, which tends to result in more coefficients being quantized to zero. However, minimizing the variance of the DCT coefficients does not necessarily lead to a minimum-length encoding of the quantized coefficients, especially since the quantized coefficients are then Huffman and run-length coded. Furthermore, since coding decisions are typically made independently of motion estimation, the effect of motion estimation on rate is further made indirect.

[1] These are 1) whether to use motion compensation and 2) whether to use the loop filter with motion compensation.

In this section, we describe two algorithms that perform motion estimation explicitly to minimize rate and a third algorithm that minimizes a combination of rate and distortion. We then present results of experiments that compare these algorithms with the standard motion estimation algorithm used by the PVRG coder.

4.3.1 Algorithm M1

In Algorithm M1, motion estimation is performed explicitly to minimize (locally) the code-length of each macroblock. The decisions of whether to use motion compensation and whether to use the loop filter are made in the same way as in the PVRG implementation. We invoke the appropriate encoding subroutines for each choice of motion vector within the search area, picking the motion vector that results in the minimum code-length for the entire macroblock. The computed code-length includes the coding of the transform coefficients for the luminance blocks,[2] the motion vector, and all other side information. When choosing the motion vector to minimize the coding of the current macroblock, we use the fact that the motion vectors for previous macroblocks (in scan order) have been determined in order to compute the code-length. However, since the choice of a motion vector for the current macroblock affects the code-length of future macroblocks, this is a greedy minimization procedure which may not result in a globally minimal code-length.

4.3.2 Algorithm M2

Algorithm M2 differs from Algorithm M1 in that the decisions of whether to use motion compensation and the loop filter are also made to minimize rate: all three combinations of the decisions are tried, and the one resulting in the minimum code-length is used. Since M2 is able to make decisions on how to code each macroblock, it is able to take into account the coding of side information in minimizing the rate. For low bit rates, where the percentage of side information is significant compared to the coding of motion vectors and transform coefficients, we would expect M2 to be effective in reducing the code-length of side information.

4.3.3 Algorithm RD

With Algorithms M1 and M2, we minimize rate without regard to distortion and then choose the quantization step size to achieve the desired distortion level. This is not always the best policy. There may be cases where the choice of motion vector and coding decisions that minimize rate results in a relatively high distortion, whereas another choice would have a slightly higher rate but substantially lower distortion. In terms of rate-distortion tradeoff, the second choice may be better. Since the ultimate goal is better rate-distortion performance, we expect further improvements if we minimize a combination of rate and distortion. M1 and M2 call encoder routines in the minimization steps. By adding calls to decoder routines, we can compute the resulting distortion. We incorporate this idea into Algorithm RD.

[2]The transform coding of the chrominance blocks could be included as well. However, we chose not to do so in order to make a fair comparison to the base PVRG coder. This is also the policy for the other coders described in this chapter.

Algorithm RD minimizes a linear combination of rate and distortion. Let $B(\vec{v}, \vec{c})$ denote the number of bits to code the current macroblock using motion vector $\vec{v}$ and coding decisions $\vec{c}$. Similarly, let $D(\vec{v}, \vec{c})$ be the resulting mean squared error. RD minimizes the objective function

$$C_{\mathrm{RD}}(\vec{v}, \vec{c}) = B(\vec{v}, \vec{c}) + \lambda D(\vec{v}, \vec{c}). \tag{4.1}$$

This objective function is very similar to the Lagrange cost function C_λ presented in Section 3.3.3.[3] However, there are notable differences. In Section 3.3.3, the Lagrange cost C_λ is used to solve a budget-constrained bit allocation problem. While we could recast motion estimation as such a problem, where the motion vectors are the parameters being controlled, there are several disadvantages of taking such an approach. First, this would require setting a bit budget for coding each frame and thus involves complex rate-control issues. Secondly, since in $p \times 64$ a motion vector is coded with reference to a previous motion vector, there is dependence at the macroblock level, further complicating the problem.

Instead, we view C_{RD} as simply being a weighted sum of distortion and rate. However, noting the connection between C_{RD} and C_λ, we can choose a value for λ based upon the operational rate-distortion curve for the input video. A good choice is to set λ to be equal to the negative of the slope of the line tangent to the operational distortion vs. rate curve at the operating point. This can be determined, for example, by preprocessing a portion of the input video to estimate the rate-distortion curve. An on-line iterative search method could also be used [98]. In our experiments, we code the test sequence several times with different quantizer step sizes to estimate the rate-distortion function, and fix λ based upon the slope of the function at the desired operating point. Our purpose is to explore the performance improvement offered by such an approach.

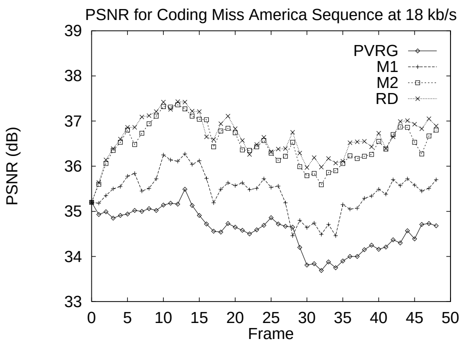
4.3.4 Experimental Results

For our experiments, we coded 49 frames of the `Miss America` sequence and 30 frames of the `Claire` sequence, both in QCIF format sampled at 10 frames/sec. These are “head and shoulders” sequences typical of the type found in videophone and videoconferencing applications. We present results here for coding at 18 kbits/sec using the rate controller outlined in Reference Model 8. The average PSNR for each coded frame is plotted for the `Miss America` and `Claire` sequences in Figure 4.2. The average PSNR for inter-coded frames are tabulated in Table 4.2. For each sequence, all the coders used the same quantization step size for the initial intra-coded frame.

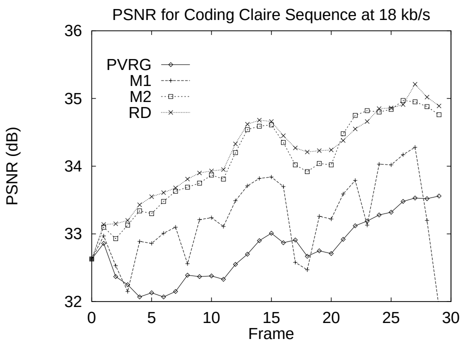
4.4 Heuristic Algorithms

While Algorithms M1, M2, and RD generally exhibit better rate-distortion performance than the base PVRG coder, they are computationally expensive. The additional computation is in the explicit evaluation of the rate (and distortion in the case of RD). To reduce the computational

[3] An obvious difference is the placement of the multiplier λ. In principle, the placement of λ does not matter since the objective function is being minimized. Multiplying the objective function by $\frac{1}{\lambda}$ would convert between the two forms. We choose this particular formulation for C_{RD} because it is convenient to think of the cost function as being in unit of bits with λ being a conversion factor between distortion and bits.



(a) Miss America



(b) Claire

Figure 4.2: Comparison of explicit-minimization motion estimation algorithms for coding the Miss America and Claire sequences at 18 kbits/sec.

complexity, we propose to minimize an efficiently computed model of rate and distortion. The idea is that the prediction error (MSE, MAD, or similar measure) can be used to estimate the rate and distortion for transform coding. This estimate is then combined with the motion vector code-length, which is readily available with a table lookup. We develop such a cost function below and use it in two heuristic coders H1 and H2 that are analogous to the explicit minimization coders M1 and M2. Both H1 and H2 choose motion vectors to minimize the cost function. However, H1 makes coding decisions using the same decision functions that the PVRG and M1 coders use, while H2 chooses the coding control that minimizes the coding rate given the estimated motion vectors. Since H2 has to try out three coding control choices, it will be about three times slower than H1. However, H2 gives us an indication of the performance that is achievable by improving the coding control. Also, H2 is easily parallelized, using duplicated hardware for example.

4.4.1 Heuristic Cost Function

Let $\vec{E}(\vec{v})$ denote a measure of the prediction error that results from using motion vector $\vec{v}$ to code the current macroblock. For example, the error measure could be defined as $\vec{E}(\vec{v}) = \langle \mathrm{MAD}(\vec{v}), \mathrm{DC}(\vec{v}) \rangle$, where $\mathrm{MAD}(\vec{v})$ is the mean absolute prediction error and $\mathrm{DC}(\vec{v})$ is the average prediction error. Suppose we have a model $H(\vec{E}(\vec{v}), Q)$ that gives us an estimate of the number of bits needed to code the motion compensation residual, where $\vec{E}(\vec{v})$ is defined above and Q is the quantization step size. We could then combine this estimate with $B(\vec{v})$, the number of bits to code the motion vector $\vec{v}$. The result is a cost function that we can use for motion estimation:

$$C_{\mathrm{H}}(\vec{v}, Q) = H(\vec{E}(\vec{v}), Q) + B(\vec{v}). \tag{4.2}$$

As defined above, the function H provides an estimate of the number of bits needed to code the motion compensation residual with quantizer step size Q. As we will discuss later, it can also be used to estimate a combination of rate and distortion.

The choice of error measure $\vec{E}$ and heuristic function H are parameters to the motion estimation algorithm. In our investigations, we used MAD as the error measure, for computational reasons. We also looked into using the MSE, but this did not give any clear advantages over the MAD. It is also possible to define $\vec{E}$ to be a function of several variables. However, we report only on the use of MAD for $\vec{E}$ and denote $\vec{E}(\vec{v})$ by ξ for convenience, where the dependence upon $\vec{v}$ is implicit. We examined several choices for H and describe them below.

As mentioned above, we can use H to estimate the number of bits used to transform-code the prediction error. To get an idea of what function to use, we gathered experimental data on the relationship between the MAD and DCT coded bits per macroblock for a range of motion vectors. Fixing the quantization step size Q at various values, the data was generated by running the RD coder on two frames of the Miss America sequence and outputting the MAD and DCT coded bits per macroblock for each choice of motion vector. The results are histogrammed and shown as density plots in Figure 4.3.

These plots suggest the following forms for H:

$$H(\xi) = c_1\xi + c_2 \ , \tag{4.3}$$

$$H(\xi) = c_1 \log(\xi + 1) + c_2, \tag{4.4}$$

$$H(\xi) = c_1 \log(\xi + 1) + c_2\xi + c_3. \tag{4.5}$$

The above forms assume a fixed Q. In general, H also depends upon Q; however, when using H to estimate the motion for a particular macroblock, Q is held constant to either a preset value or to a value determined by the rate control mechanism. We can treat the parameters c_i as functions of Q. Since there is a small number (31) of possible values for Q, we can perform curve fitting for each value of Q and store the parameters in a lookup table.

We can also model the reconstruction distortion as a function of prediction error. We use the RD coder to generate experimental data for distortion versus MAD, shown in Figure 4.4, and find a similar relationship as existed for bits versus MAD. Again, we can use (4.3)–(4.5) to model the distortion. As with the RD coder, we can consider jointly optimizing the heuristic estimates of rate and distortion with the following cost function:

$$C_{\mathrm{H}}(\vec{v}, Q) = B(\vec{v}) + H_{\mathrm{R}}(\xi, Q) + \lambda H_{\mathrm{D}}(\xi, Q), \tag{4.6}$$

where H_{R} is the model for rate and H_{D} is the model for distortion.

If we use one of (4.3)–(4.5) for both H_{R} and H_{D}, the combined heuristic function, $H = H_{\mathrm{R}} + \lambda H_{\mathrm{D}}$, would have the same form as H_{R} and H_{D}. Therefore, we can interpret the heuristic as modeling a combined rate-distortion function. In this case, we can perform curve fitting once for the combined heuristic function by training on the statistic $R + \lambda D$, where R is the DCT bits for a macroblock and D is the reconstruction distortion for the macroblock. As with Algorithm RD, the parameter λ can be determined from the operational rate-distortion curve, for example.

4.4.2 Experimental Results

To test the H1 and H2 coders, we used the same test sequences and followed the procedures outlined in Section 4.3.4. In the next section, we verify these results with eight different test sequences.

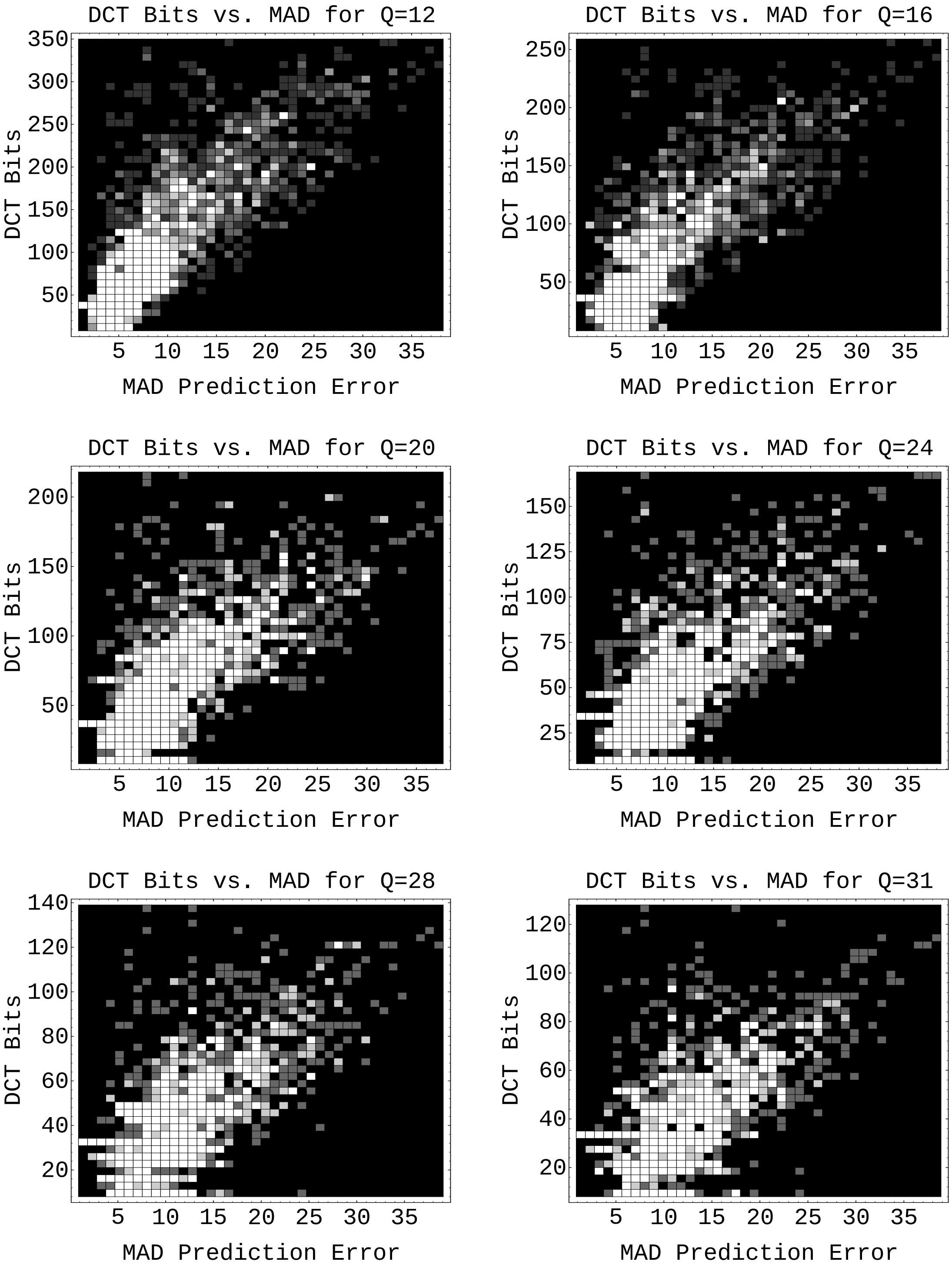


Figure 4.3: Density plots of DCT coding bits vs. MAD prediction error for first inter-coded frame of Miss America sequence at various levels of quantization.

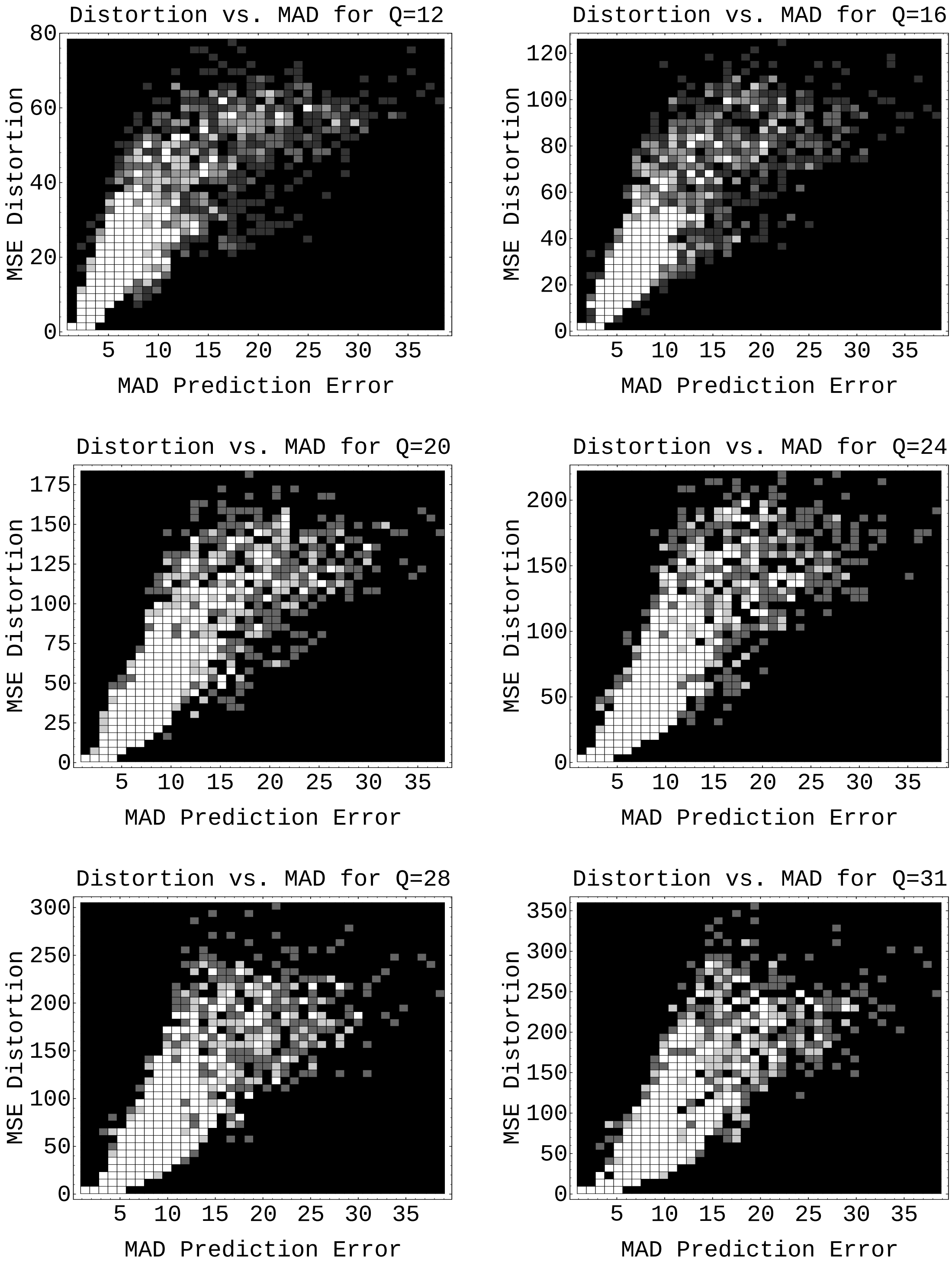
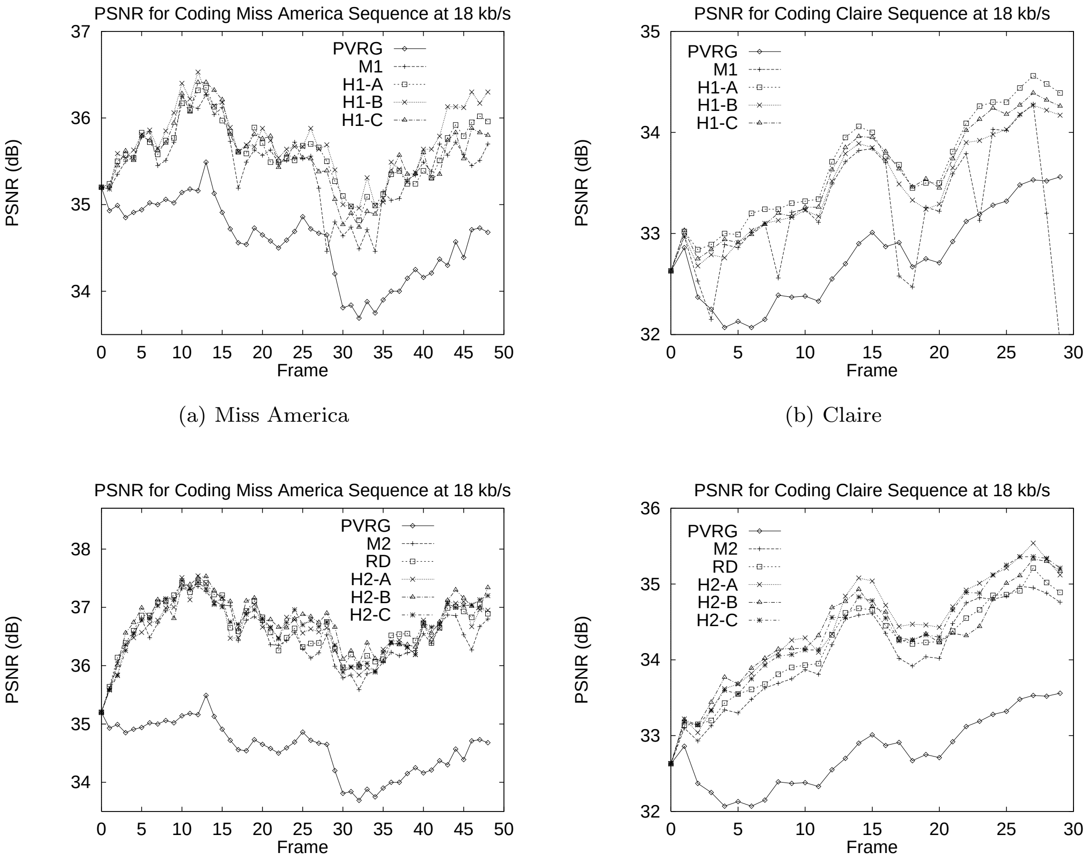


Figure 4.4: Density plots of MSE reconstruction distortion vs. MAD prediction error for first inter-coded frame of Miss America sequence at various levels of quantization.



(a) Miss America

(b) Claire

(c) Miss America

(d) Claire

Figure 4.5: Results of static heuristic cost function for coding Miss America and Claire sequences at 18 kbits/sec with rate control. In (a) and (b), we compare the H1 coder with the PVRG and M1 coders. In (c) and (d), we compare the H2 coder with PVRG, M2 and RD coders. H1-A (H2-A), H1-B (H2-B), and H1-C (H2-C) use the heuristic functions (4.3), (4.4), and (4.5), respectively.

4.4.2a Static Cost Function

Here, we present results using a static set of coefficients. To determine the coefficients for the heuristic functions, we performed linear least squares regression, fitting data generated by the RD coder to the $R + \lambda D$ statistic, as discussed earlier. A set of regression coefficients are stored in a lookup table, indexed by the quantizer step size Q. We tested the different forms for the heuristic function given in (4.3)–(4.5). Comparative plots of the resulting PSNR are shown in Figure 4.5. The average PSNR for coding at 18 kbits/sec is tabulated in Table 4.2. These results show that the heuristic coders perform comparably to the explicit minimization coders. In particular, the heuristic coders seem more robust than M1 and M2, most likely because the heuristic functions correlate well with both rate and distortion, whereas M1 and M2 only consider rate.

Sequence	PVRG	M1	M2	RD	H1-A	H1-B	H1-C	H2-A	H2-B	H2-C
Miss America	34.58	35.44	36.51	36.67	35.60	35.72	35.58	36.63	36.77	36.68
Claire	32.77	33.24	34.12	34.22	33.68	33.50	33.60	34.47	34.36	34.39

Table 4.2: Results of static heuristic cost function. Shown is average PSNR (in dB) of inter-coded frames for coding test sequences at 18 kbits/sec. H1-A (H2-A), H1-B (H2-B), and H1-C (H2-C) use the heuristic functions (4.3), (4.4), and (4.5), respectively.

4.4.2b Adaptive Cost Function

The above results rely on pretraining the model parameters c_i for each value of Q for each video sequence. This is a tedious and time-consuming operation. Instead, we can use an adaptive on-line technique, such as the Widrow-Hoff learning rule [109, 36], to train the model parameters. The training examples could be generated each time we encode a macroblock using motion compensation mode. However, we cannot possibly hope to train one model for each value of Q simply because there would not be enough training examples. We need a single model whose parameters are independent of Q. The curve fitting results from the pretraining trials show a strong correlation between the model parameters and Q^{-1}. This agrees well with previous work on rate-quantization modeling [23]. Therefore we propose the following form for the cost function:

$$H(\xi, Q) = c_1 \frac{\xi}{Q} + c_2. \tag{4.7}$$

This can be simplified as

$$H(\psi) = c_1 \psi + c_2, \tag{4.8}$$

where $\psi \equiv \xi/Q$. Since the simple linear model performed well with static cost functions, we do not consider more complex models here.

We conducted experiments using the Widrow-Hoff training rule on the Miss America and Claire sequences. As applied to the current context, the Widrow-Hoff rule is a technique for learning an objective function $f(\psi)$. With $H(\psi)$ as an estimate of $f(\psi)$, the Widrow-Hoff rule gives us a way to adapt the weights c_1 and c_2 of (4.8) when given ψ and the value of $f(\psi)$. For the experiments, we chose the objective function

$$f(\psi) = R(\psi) + \lambda D(\psi), \tag{4.9}$$

where $R(\psi)$ is the actual number of bits used to code the DCT coefficients and $D(\psi)$ is the resulting distortion, both of which can be evaluated by invoking encoder routines. Given an initial set of weights c_1 and c_2, a new set of weights c_1' and c_2' can be computed as:

$$c_1' = c_1 + \psi\eta \cdot \frac{f(\psi) - H(\psi)}{\psi^2 + 1}, \tag{4.10}$$

$$c_2' = c_2 + \eta \cdot \frac{f(\psi) - H(\psi)}{\psi^2 + 1}; \tag{4.11}$$

where η, the *learning rate*, is a parameter that determines how quickly the weights are adapted.

With the static cost function, we trained and evaluated the heuristic function based on the combined prediction error for the luminance blocks within a macroblock. To generate more training examples for the adaptive heuristics, we evaluate and update the heuristic function once for each luminance block. This strategy increased the PSNR slightly at the expense of some extra computation.

In the experiments, the learning rate η was determined in a trial-and-error phase and fixed for both sequences. The parameter λ was also determined by trial-and-error and held constant for both test sequences. Comparative plots of the resulting PSNR are shown in Figures 4.6. The average PSNR for coding at 18 kbits/sec is tabulated in Table 4.3. These results show that the adaptive heuristic coders perform comparably to and sometimes better than the static heuristic coders and the explicit minimization coders. Furthermore, the adaptive heuristic coders perform well on both sequences with the same initial parameter values.

As a comparison of visual quality, Frame 27 of the Miss America sequence is decoded and shown in Figure 4.7 for the PVRG and explicit-minimization coders and in Figure 4.8 for the heuristic coders. The motion vector field for the PVRG, RD, adaptive H1, and adaptive H2 coders are shown in Figure 4.9. Frame 27 was chosen because it is in a difficult scene with much head motion, resulting in more noticeable coding artifacts. The RD and adaptive heuristic coders give smoother motion fields than the reference PVRG coder, especially for the background region. Note also that for the former coders, no motion is indicated for the relatively uniform background *except* following macroblocks with detected foreground motion on the same row. Intuitively, this results in an economical encoding of the motion vectors, which are differentially encoded. Since the background is relatively uniform, coding motion in this area results in relatively small motion compensation residual.

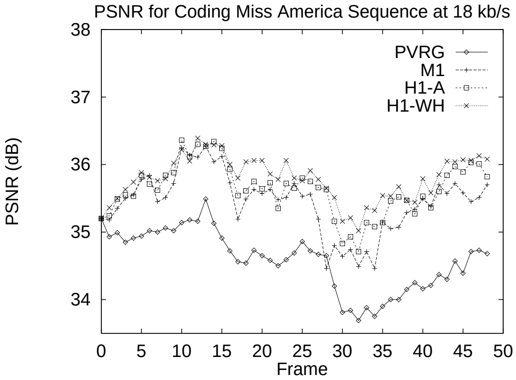
4.4.3 Further Experiments

Here, we present results of further experiments to confirm the efficacy of the various motion estimation algorithms operating within the $p \times 64$ standard. We applied the various algorithms to code eight test video sequences[4] without rate control, sweeping the quantization scale from 12 to 31 to determine the operational rate-distortion plots shown in Figure 4.10. Each test sequence consists of 50 frames in QCIF format coded at 10 frames/sec.

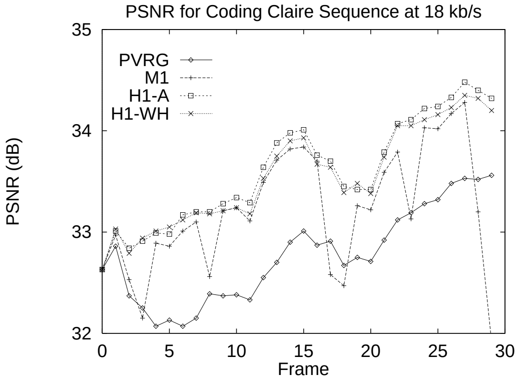
The results show that the adaptive heuristic algorithms perform consistently well compared to the base PVRG and explicit-minimization implementations, though the level of improvement varies among sequences. The anomalies observed in coding the `Grandma` sequence at low rates with the PVRG and adaptive H1 coders, as evidenced by the steep slope and unevenness in the RD curve, seem to indicate a breakdown of the RM8 coding control heuristics, which were not optimized for operation at very low rates. This conclusion is supported by the lack of such anomalies when bit-minimization is used to perform coding control, as with the M2, H2, and RD coders.

The distributions of bits for coding the Miss America sequence with the H1 and H2 coders are plotted in Figure 4.11. Compared to Figure 4.1, these plots show that the H1 and H2 coders both reduce the percentage of bits used for coding motion vectors, while increasing the percentage of bits used to code the DCT coefficients. Furthermore, with the H2 coder, which applies bit-minimization to coding control, the number of bits used for coding side information is also reduced.

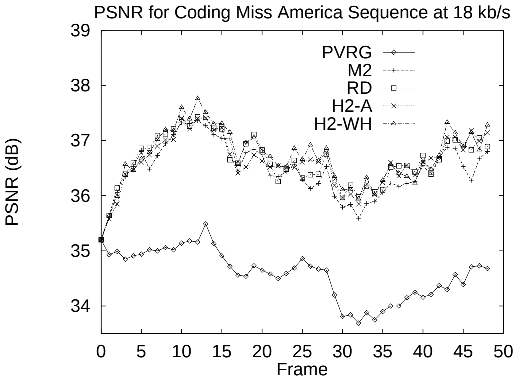
[4]The `Miss America` sequence in this test suite was obtained from a different source than the `Miss America` sequence used in the earlier experiments and has different rate-distortion characteristics.



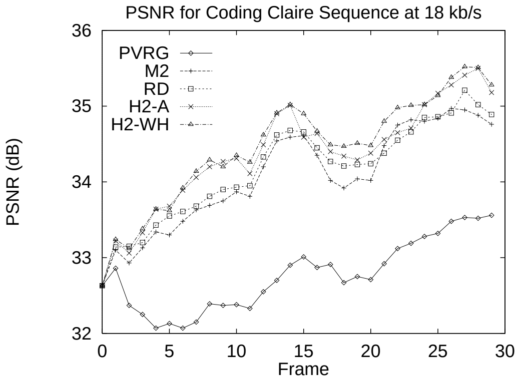
(a) Miss America



(b) Claire



(c) Miss America



(d) Claire

Figure 4.6: Results of adaptive heuristic cost function for coding Miss America and Claire sequences at 18 kbits/sec with rate control. In (a) and (b), we compare the adaptive H1 coder with the static H1, PVRG and M1 coders. In (c) and (d), we compare the adaptive H2 coder with the static H2, PVRG, M2 and RD coders. H1-A and H2-A use the heuristic function (3).

Video	PVRG	M1	M2	RD	H1-A	H1-WH	H2-A	H2-WH
Miss America	34.58	35.44	36.51	36.67	35.60	35.83	36.63	36.84
Claire	32.77	33.24	34.12	34.22	33.68	33.58	34.47	34.51

Table 4.3: Results of adaptive heuristic cost function. Shown is average PSNR (in dB) of inter-coded frames for coding test sequences at 18 kbits/sec. H1-A and H2-A use the heuristic function (3) with static parameters. H1-WH and H2-WH use adaptive parameters.

(a) PVRG

(b) RD

(c) M1

(d) M2

Figure 4.7: Frame 27 of the Miss America sequence as encoded using the PVRG and explicit-minimization motion estimation algorithms. Only the luminance component is shown.

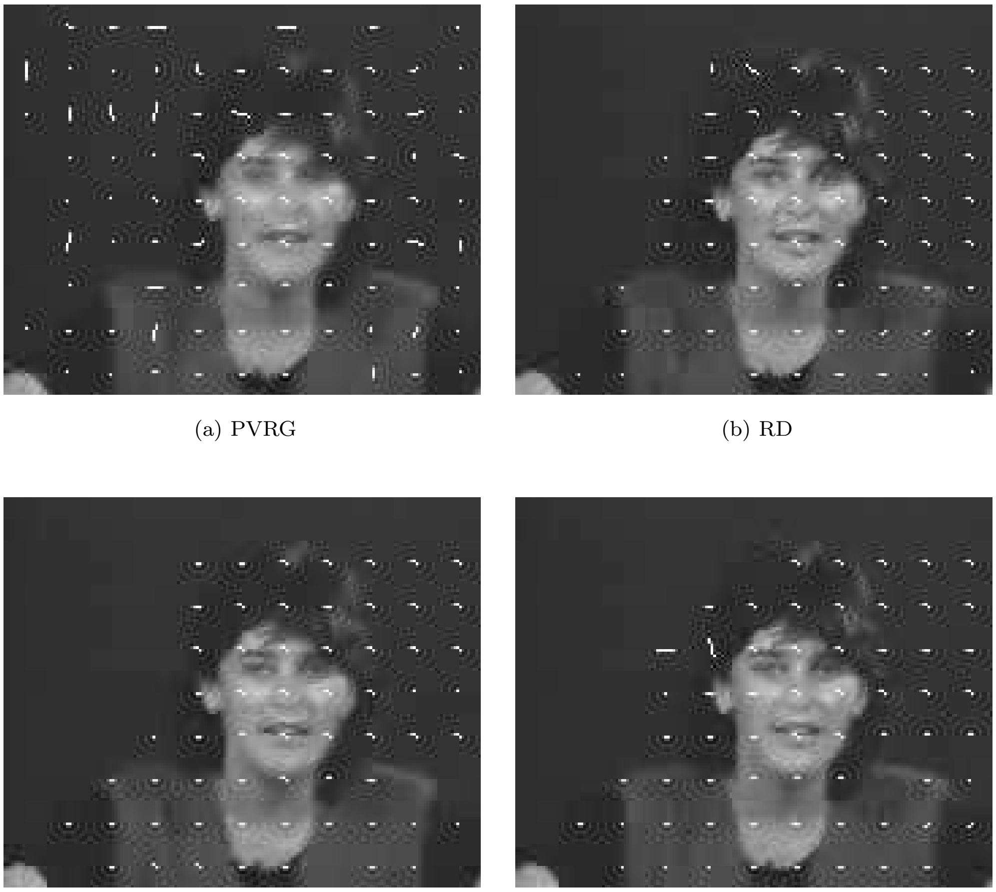
(a) H1-A

(b) H2-A

(c) H1-WH

(d) H2-WH

Figure 4.8: Frame 27 of the Miss America sequence as encoded using the heuristic motion estimation algorithms. Only the luminance component is shown.



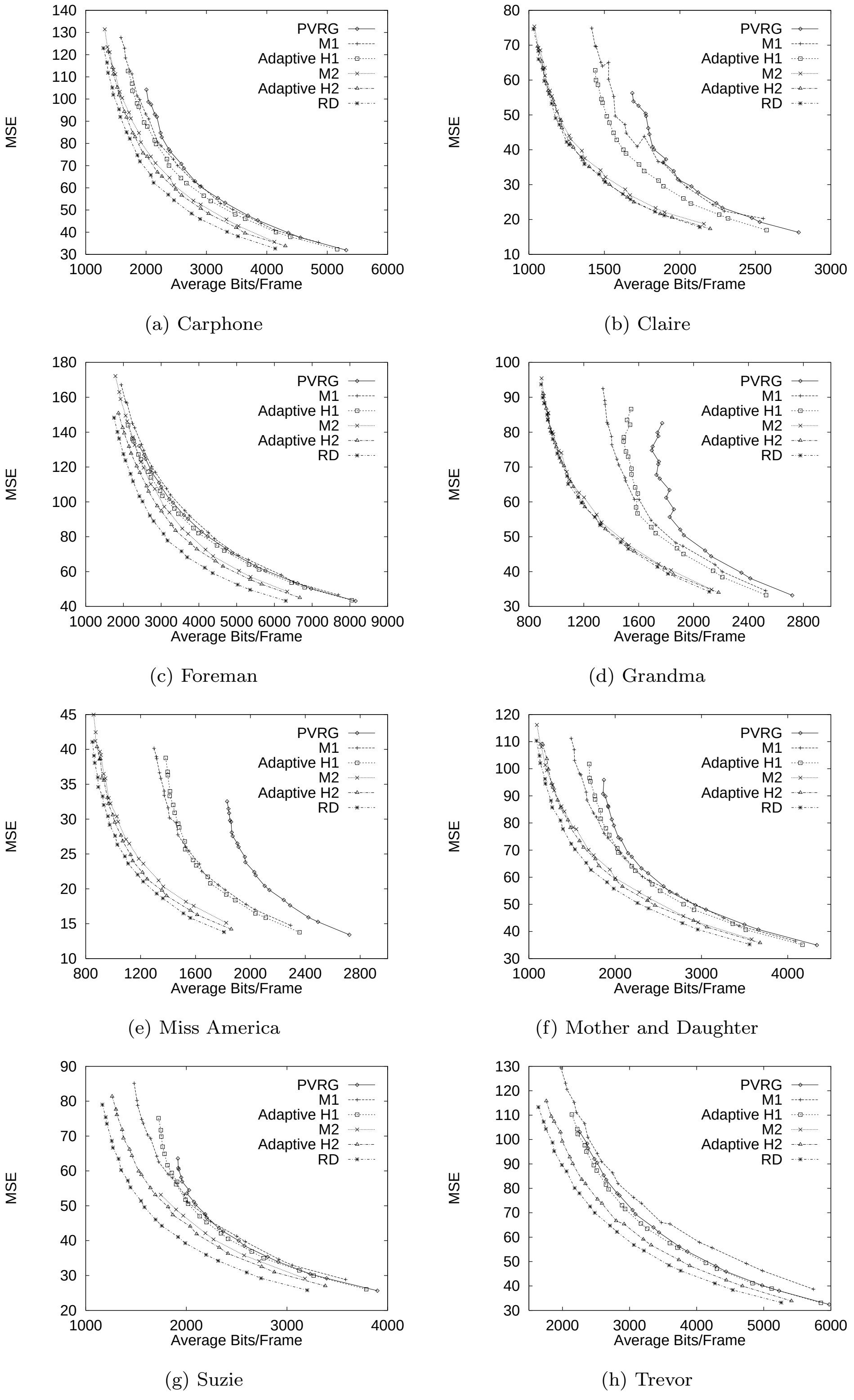
(a) PVRG

(b) RD

(c) H1-WH

(d) H2-WH

Figure 4.9: Estimated motion vectors for frame 27 of the Miss America sequence for the PVRG, RD, H1-WH, and H2-WH coders. Only the magnitude of the motion vectors are shown.



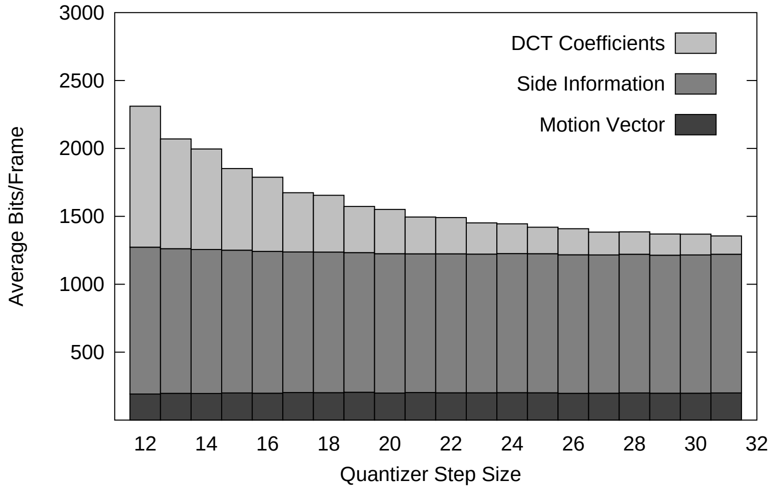
(a) Carphone (b) Claire

(c) Foreman (d) Grandma

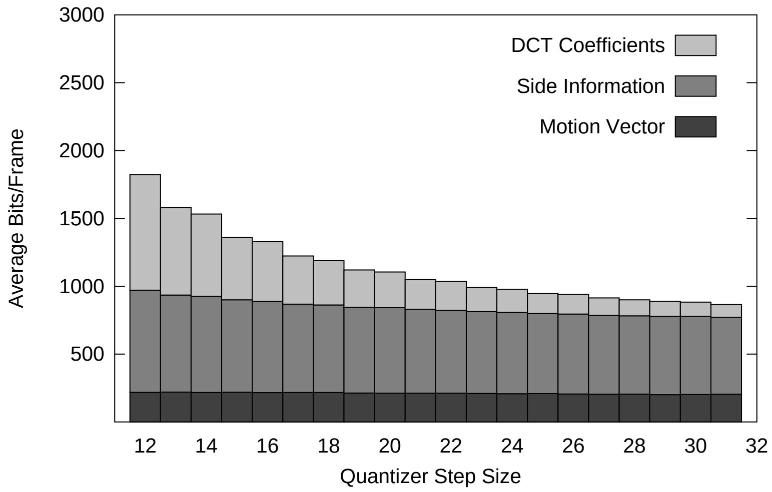
(e) Miss America (f) Mother and Daughter

(g) Suzie (h) Trevor

Figure 4.10: Performance of motion estimation algorithms on eight test sequences.



(a) H1 Coder



(b) H2 Coder

Figure 4.11: Distribution of bits for coding the Miss America sequence with adaptive heuristics.

4.5 Related Work

In related work, Chung, Kossentini and Smith [17] consider rate-distortion optimizations for motion estimation in a hybrid video coder based upon subband coding and block-matching motion compensation. The input frames are first decomposed into subbands, which are divided into uniform rectangular blocks. For each block, a Lagrangian cost function is used to select between intraframe and interframe modes and to select between a small number of candidate motion vectors, which are coded with a lossy two-dimensional vector quantizer.

Independent of our work, rate-distortion optimization for motion estimation has been reported in [11]. The authors consider operational rate-distortion optimization in a dependent-coding environment where motion vectors are coded using DPCM techniques. The authors formulate motion estimation as a budget-constrained bit allocation problem where the distortion to be minimized is the displaced frame difference (DFD), or prediction error, and propose to solve the problem using the Lagrange-multiplier approach. After first constructing a motion-vector dependency graph, the Viterbi dynamic programming algorithm is used to find a path that minimizes an additive Lagrangian cost function. Noting the computational complexity of this approach, the authors propose a reduced-complexity algorithm that considers only a small fraction of the possible states for each motion-compensated block. Even so, this reduced-complexity algorithm has a considerable processing and memory overhead associated with the dynamic programming algorithm, which is performed on top of traditional block matching. In comparison, our adaptive heuristic cost function requires minimal overhead over block matching.

In another independent work [95], the authors also apply operational rate-distortion optimization to motion estimation in a dependent-coding environment. Similar to [11], they cast motion estimation as a budget-constrained bit allocation problem and solve it using a combination of Lagrange minimization and dynamic programming. The authors also consider joint optimization of motion estimation and quantization using the same framework. Notably, they also provide a solution to both problems that minimizes the maximum (minimax) distortion.

In [110], rate-distortion optimization is applied to the selection of coding control for low-bit-rate video coding under the H.263 standard, a newer standard than the H.261 standard that we consider here. A greedy optimization strategy is adopted to avoid the exponential complexity that a global optimization would entail. Limited dependencies between the coding control of neighboring blocks is considered and the coding control is computed using the Viterbi dynamic programming algorithm to minimize a Lagrangian cost function. Even with simplifying assumptions, the rate-distortion optimization is computationally complex and may not be suitable for real-time implementation, as the authors readily admit.

Ribas-Corbera and Neuhoff [92] describe a procedure for minimizing rate in a lossless motion-compensated video coder. They explore the allocation of bits between the coding of motion vectors and the coding of prediction error. They assume that the prediction error has a discrete Laplacian distribution and derive an expression for the total rate as a function of the number of bits allocated to code the motion vectors. It is not clear whether this work can be extended to lossy coding since distortion is not taken into account in the formulation.

A linear relationship between MAD and both rate and distortion has been independently observed in [71]. The authors mention the possibility of performing motion vector search to minimize the bit rate, but conclude that just minimizing MAD would have a similar effect.

4.6 Discussion

In this chapter, we have demonstrated that, at low bit rates, choosing motion vectors to minimize an efficiently computed heuristic cost function gives substantially better rate-distortion performance than the conventional approach of minimizing prediction error. Furthermore, by adapting the heuristic function to the input sequence, we are able to achieve coding performance comparable to more computationally expensive coders that explicitly minimize rate or a combination of rate and distortion.

In the experiments, full-search block-matching was employed by all the coders. Our fast heuristic coders are also compatible with 2D logarithmic and many other reduced-search motion estimation techniques. Since the heuristic cost function factors in the motion vector code-length, the cost function has a strong monotonic component and is well-suited for the reduced-search techniques that assume monotonicity in the cost function.

We have considered only the simple case of using a fixed parameter λ to tradeoff rate and distortion. An on-line adaptation of λ to track variations in the input sequence is certainly possible and would result in more robust coders. On the other hand, we observed that the behavior of these algorithms is quite robust with respect to moderate variations in λ, and that, for example, the best setting of λ for one test sequence worked well when used for the other. Thus, it seems that fixing λ is safe in practice. Still, since λ influences rate to some extent, it can be used in conjunction with the quantization step size in performing rate control. Automatic control of λ based upon buffer feedback as described in [13] is a possibility.

Although the methods presented here have been implemented within the H.261 standard, it should be generally applicable to any video coder that employs motion compensation in a low bit rate setting. In particular, the H.263 standard is similar enough to H.261 that it seems clear that these methods will work well with H.263. As a case in point, in the next chapter we show how the bit-minimization framework can be applied in a non-standard coder that chooses motion vectors to optimize a hierarchical encoding of the motion information within a block-matching framework with variable block sizes.

Chapter 5

Bit-Minimization in a Quadtree-Based Video Coder

Like the underlying image data that they are computed from, motion vector fields often exhibit significant spatial correlation. Approaches such as the $p \times 64$ standard exploit this observation by coding the differences between successive motion vectors in a one-dimensional scan order. Potentially better results can be achieved by directly exploiting the two-dimensional correlation of motion vectors. A quadtree data structure can be used for this purpose by representing the motion field hierarchically in two dimensions [87, 5, 9, 24]. The quadtree approach works by identifying variable-sized regions of uniform motion and providing an efficient encoding of the motion field. For video sequences where there are large regions of uniform motion, a quadtree decomposition reduces the number of bits required to encode the motion field by coding only a few motion vectors. When the motion field is smoothly varying, the hierarchical structure of the quadtree allows the correlated motion vectors to be efficiently coded.

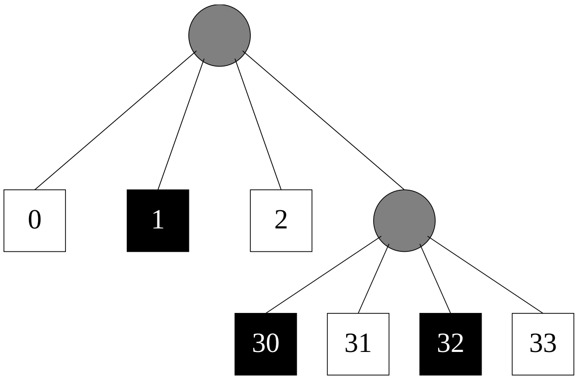
In this chapter, we consider the instantiation of the bit-minimization principle in a video coder that uses a quadtree to code motion vectors, thereby departing from the $p \times 64$ standard. The contents of this chapter were originally presented in [37].

5.1 Quadtree Data Structure

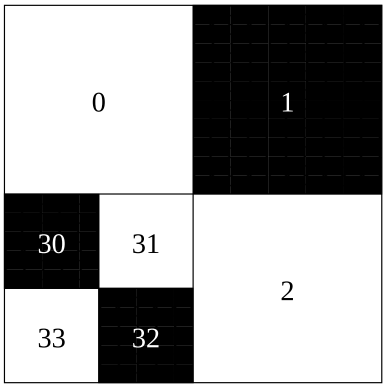
Often used as a representation for bi-level images, the quadtree can also be used to code motion vectors. We first describe the quadtree in the context of bi-level image representation and later show how to use it to code a motion field.

5.1.1 Quadtree Representation of Bi-Level Images

The root node of a quadtree represents the entire image. For the trivial case where an image contains all white or all black pixels, its quadtree representation is the root node colored either white or black, respectively. Otherwise the image is divided into four quadrants, each a child



(a) Quadtree



(b) Image

Figure 5.1: A simple quadtree and corresponding image.

of the root node. If a quadrant is totally white or totally black, it is represented as a leaf node appropriately colored; otherwise it is represented in the quadtree by an internal node and is further divided into subquadrants. The division proceeds recursively up to a maximum depth or until all subdivisions contain uniform pixels. A simple quadtree and its corresponding image is shown in Figure 5.1.

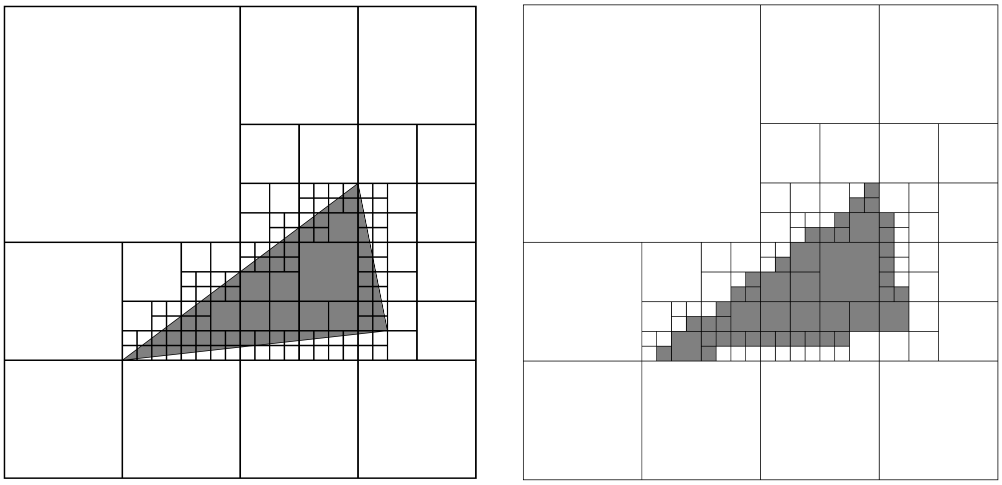
In Figure 5.2, a decomposition of a triangle is shown using a quadtree of depth 5. The image is divided into the regions shown in Figure 5.2(a). Since the depth of the quadtree is finite, the triangle can only be approximated with limited precision. Adopting a policy that each of the smallest subdivision is colored with the predominant color of the underlying image, we can represent the triangle using the quadtree structure shown in Figure 5.2(b). Note that some of the regions identified in (a) have been merged in (b) since they are assigned the same color and have a common parent.

The structure of a quadtree can be represented linearly by traversing the tree. Labeling an internal node as `I`, a black leaf as `B`, and a white leaf as `W`, we can represent the tree in Figure 5.1 with the string `IWBWIWBWB`. This corresponds to a preorder traversal of the quadtree.

The quadtree can be extended to code greyscale and color images by storing at each leaf a greyscale or color value, chosen to minimize some measure of distortion.

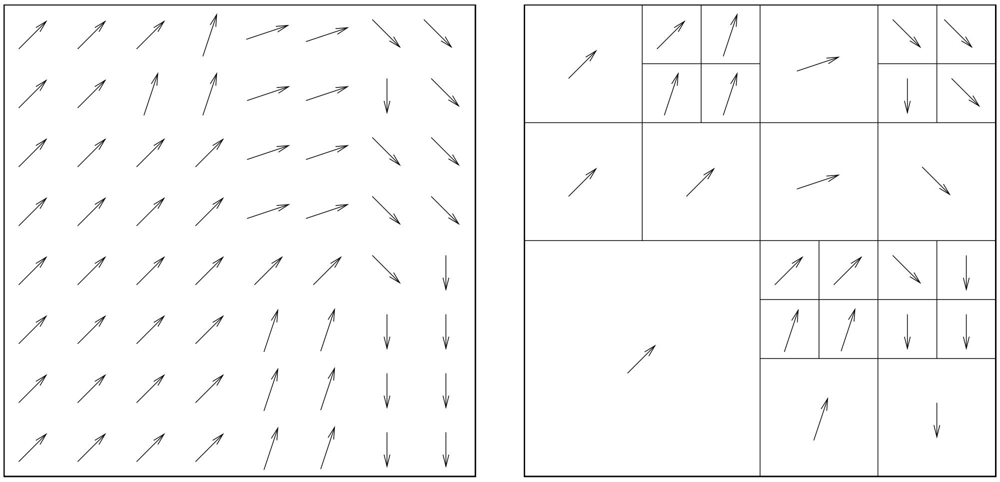
5.1.2 Quadtree Representation of Motion Vectors

The quadtree data structure can also be used to represent a motion field, as shown in Figure 5.3. Instead of storing a color value at each leaf, we store a motion vector that represents the motion of the region represented by the leaf. When combined with a block-transform coder for the residual, such as the 2D-DCT, we can also use the quadtree segmentation to determine the size of the block transform; the result is a hybrid motion-compensated video coder with variable-sized blocks.



(a) Segmentation (b) Representation

Figure 5.2: Representation of a triangle using a quadtree of depth 5.



(a) Motion Field (b) Quadtree Representation

Figure 5.3: Quadtree representation of a motion field.

The linear representation of a quadtree mentioned above can be used here to encode the motion field if there is a finite number of possible motion vectors. However, assuming that the motion field is smoothly varying, we can improve the coding efficiency by coding differences in neighboring motion vectors. We can do this in a two-dimensional manner by exploiting the segmentation that the quadtree already provides. With each internal node, we associate a motion vector representative of the node's descendents. The representative motion vector may be an average or a median or may be chosen by some other means. In each node, instead of storing the representative motion vector directly, we store its difference with that of the node's parent. Since the root node does not have a parent, its representative motion vector is stored as is. The value of each node's representative motion vector can be determined by summing the stored values encountered along the path from the root to that node. For a smoothly varying motion field, the difference values would tend to be small and can be coded efficiently with a statistical coder, for example.

5.2 Hybrid Quadtree/DCT Video Coder

We propose to combine the quadtree encoding of motion vectors with the basic motion-compensated video coder specified in the $p \times 64$ standard. Conceptually, the main difference between this coder and the $p \times 64$ coder is in the encoding of the motion vectors. Each leaf in the quadtree encodes a motion vector for a number of 8×8 blocks. Therefore, the quadtree decomposition ends at the 8×8 level.

Given a motion field and its quadtree decomposition, we code the structure of the tree using an adaptive arithmetic code to indicate whether a node is a leaf or not (a different adaptive coder is used for each level). The motion vector differences at each node are coded using another adaptive arithmetic code (again, using a different coder for each level). For each leaf node, the 8×8 transform coded blocks subsumed by the node are transmitted in scan order. The decision of how to code the block (choosing from among alternatives similar to those in the $p \times 64$ standard) is also transmitted using an adaptive arithmetic coder. If the quantized transform coefficients are transmitted, this is done using the run-length/Huffman coding method from the $p \times 64$ standard. The counts for the adaptive arithmetic coder are updated once after each frame is coded.

To perform motion estimation, we adopt the bit-minimization strategy elaborated in Chapter 4. The quadtree coding structure described in Section 5.1.2 has several nice properties that make a dynamic programming solution possible for finding an optimal set of motion vectors that minimizes the sum of the code-lengths needed to encode the motion vectors and the transform-coded prediction error. Since the arithmetic code used for the motion vector differences at each node doesn't change during the coding of a particular frame, the optimal number of bits to code the motion vector differences for any subtree is independent of the coding of any other disjoint subtree. Similarly, the transform coding of the prediction errors is independent for disjoint subtrees.

We now describe a dynamic programming algorithm for choosing an optimal quadtree. For each node in the tree, we store a table indexed by the (absolute) motion vector of the node's parent. For each possible motion vector $\vec{v}$ of the parent, this table gives the minimum code-length to code the subtree rooted at the current node given that the parent's motion vector is $\vec{v}$. Also stored with each table entry is a motion vector giving the minimum code-length. Construction of the tables is performed in a bottom-up fashion, starting at the 8×8 block level. For a node p, the table is constructed by finding, for each choice of motion vector $\vec{v}'$ for the parent node, a motion vector for p that results in the minimum code-length for the subtree rooted at p. If p is at the 8×8 block level, this is done by computing the transform code-length of the prediction error for each motion vector in the search range S and noting the minimum code-length and the corresponding motion vector. Otherwise we consider for each motion vector $\vec{v}$ in S the code-length needed to transform-code the prediction errors if the quadtree is pruned at p. (This quantity can be computed in a preprocessing step.) We also consider the code-length if the quadtree is not pruned at p. This code-length is computed by indexing the tables of children of p with $\vec{v}$ and summing. The minimum of these two quantities is added to the number of bits to code $\vec{v}' - \vec{v}$. The result is the minimum code-length required to code the subtree rooted at p given motion $\vec{v}'$ at p's parent node.

Once the minimum code-length is computed for the root of the quadtree, the motion vectors for each node in the tree are determined by going back down the tree, using the tables constructed on the way up. The optimal motion vector for the root node is made known to its children. Each child uses this to index its table to find its optimal motion vector. Pruning of the tree is also performed as a result.

The dynamic programming algorithm requires $O(N|S|^2)$ time, where N is the number of 8×8 blocks in the frame and S is the search region for block-matching. The space requirement is $O(N|S|)$.

5.3 Experimental Results

The quadtree coder is implemented using routines from the PVRG coder for motion compensation, transform coding, quantization, and run-length/Huffman coding of the quantized transform coefficients. The quadtree structure and motion vector information are not actually coded; however, adaptive probability models are maintained and used to compute the number of bits that an ideal arithmetic coder would produce. Our objective is to explore the limits in performance improvement by aggressively optimizing the coding of motion vectors.

We performed coding simulations using 50 frames of the grayscale 256×256 "Trevor" sequence. The $p \times 64$ coders were modified to accept input at 256×256 resolution. An operational rate-distortion plot for the quadtree and $p \times 64$ coders is given in Figure 5.4. The quadtree coder gives better rate-distortion performance for rates under about 0.25 bits/pixel. Furthermore, the range of achievable rates is extended to well below the 0.05 bits/pixel achievable with M2, albeit with higher distortion.

5.4 Previous Work

Puri and Hang [87] considered an algorithm for motion-compensated video coding which, when an 8×8 block B is not coded well (that is, when coding it requires a lot of bits), chooses a separate motion vector for each of B's four 4×4 subblocks. Bierling [5] described a hierarchical

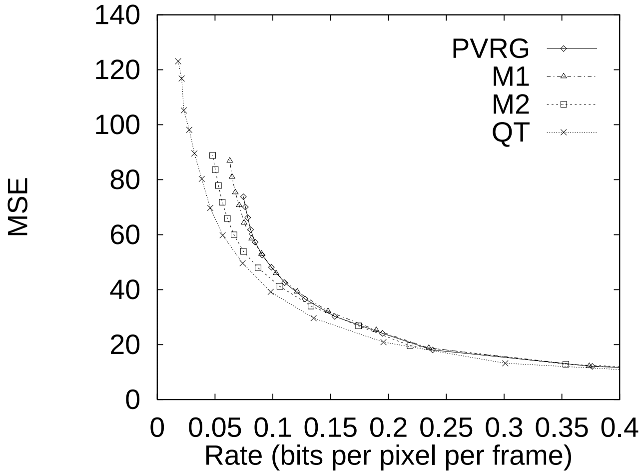


Figure 5.4: MSE vs. Rate for Trevor

algorithm for choosing motion vectors in which initially motion vectors are chosen for large blocks (64×64) by minimizing the prediction error.[1] Then for each large block B, motion vectors are chosen for subblocks of B again by minimizing prediction error, except looking only at motion vectors close to the motion vector chosen for B. This process results in a smoother motion field, and experiments suggest that it is closer to the "true" motion than is a motion field obtained by separately minimizing error on the small blocks. While Bierling did not discuss how to code motion vectors obtained through by his method, Chan, Yu, and Constantinides [9] described a method where motion vectors are again chosen in a top-down fashion, starting with large blocks and refining with smaller blocks. When the average squared prediction error for a given block B is above a given threshold, the algorithm refines the motion vector chosen for B by looking at subblocks of B. Similarly, if the use of separate motion vectors for the subblocks of B does not reduce the error significantly, the subblocks of B are "merged" with B. After this process is completed, the tree obtained by making a given block the parent of its subblocks is transmitted, together with motion vectors for each of the leaves. Methods for taking a tree structure like the above (except expanded completely) and then "smoothing" the motion vectors by making children tend to be like their parents and vice-versa, were discussed by Dufaux and Kunt [24]. Zhang, Cavenor, and Arnold [112] considered various ways of using quadtrees to code the prediction error.

5.5 Discussion

In this chapter, we have presented a non-standard hybrid block-matching DCT video coder that further optimizes the coding of motion vectors by representing the motion field using a quadtree decomposition. Although motivated for efficient coding of motion vectors, the quadtree decomposition can be viewed as performing motion compensation using variable-sized blocks.

Preliminary experimental results suggest that a quadtree-based technique may be suitable for video coding at rates below that achievable by the $p \times 64$ standard.

[1] In fact, a heuristic search [50, 55] was used to only approximately minimize the error.

While only the bit-minimization strategy has been examined for the quadtree-based coder, the heuristic cost function of Chapter 4 can certainly be used to speed up the operation of the coder.

Chapter 6

Lexicographically Optimal Bit Allocation

In any lossy coding system, there is an inherent trade-off between the rate of the coded data and the distortion of the reconstructed signal. Often the transmission (storage) medium is bandwidth (capacity) limited. The purpose of rate control is to allocate bits to coding units and to regulate the coding rate to meet the bit-rate constraints imposed by the transmission or storage medium while maintaining an acceptable level of distortion. We consider rate control in the context of the MPEG-1 and MPEG-2 standards.

In addition to specifying a syntax for the encoded bitstream and a mechanism for decoding it, the MPEG standards define a hypothetical decoder called the Video Buffering Verifier (VBV), which places quantifiable limits on the variability in coding rate. The VBV is diagrammed in Figure 3.23 and described in Section 3.7.

As outlined in Section 3.7.5, the overall rate control process can be broken down into three steps:

1. a high level bit allocation to coding units (video pictures),
2. a low level control of the quantization scale within a coding unit to achieve the bit allocation,
3. adjustment to the quantization scale to equalize perceptual quality.

In this chapter, we develop a framework for bit allocation under VBV constraints, with an additional constraint on the total number of bits coded. This framework consists of three components: 1) a bit-production model, 2) a novel lexicographic optimality criterion, and 3) a set of buffer constraints for constant and variable bit rate operation. Initially, we formalize bit allocation as a resource allocation problem with continuous variables and non-linear constraints, to which we apply a global lexicographic optimality criterion.

The global nature of the optimization necessitates the use of off-line techniques wherein the complexities of all the coded pictures, as specified with bit-production models, are known prior to computing a global bit allocation. One way to view this is as a serial computation with unlimited lookahead, wherein the inputs are the bit-production models for each picture. In practice, this would entail making multiple passes over the video sequence in order to construct the models, compute an optimal allocation, and compress the sequence using the computed allocation. In Chapter 9, we explore some techniques for reducing the computation by limiting the amount of lookahead used.

In the next two chapters, we use our new framework to analyze bit allocation under constant bit rate and variable bit rate operation. The analyses yield necessary and sufficient conditions for optimality that lead to efficient bit allocation algorithms. In Chapter 9, we describe an implementation of these algorithms within a software MPEG-2 encoder and present simulation results.

6.1 Perceptual Quantization

As shown in Figure 3.20, the output bit rate of a video coder can be regulated by adjusting the quantization scale Q_{s}. Increasing Q_{s} reduces the output bit rate but also decreases the visual quality of the compressed pictures. Similarly, decreasing Q_{s} increases the output bit rate and increases the picture quality. Therefore by varying Q_{s}, we can trace out a rate vs. distortion curve, such as that shown in Figure 3.5.

Although Q_{s} can be used to control rate and distortion, coding with a constant value of Q_{s} generally does not result in either constant bit rate or constant perceived quality. Both of these factors depend upon the scene content as well. Studies into human visual perception suggest that perceptual distortion is correlated to certain spatial (and temporal) properties of an image (video sequence) [78, 1]. These studies lead to various techniques, called *perceptual quantization* or *adaptive perceptual quantization*, that take into account properties of the Human Visual System (HVS) in determining the quantization scale [104, 86, 52, 16, 60, 108, 18].

Based upon this body of work, we propose a separation of the quantization scale Q_{s} into a *nominal quantization* Q and a *perceptual quantization function* $P(I, Q)$ such that $Q_{\mathrm{s}} = P(I, Q)$, where I denotes the block being quantized. The function P is chosen so that if the same nominal quantization Q were used to code two blocks then the blocks would have the same perceptual distortion. In this way, the nominal quantization parameter Q would correspond directly to the perceived distortion and can serve as the object for optimization. We favor a multiplicative model where $P(I, Q) = \alpha_{\mathrm{I}} Q$.[1] Where quantization noise is less noticeable, such as in highly-textured regions, we can use a larger value for α_{I} than regions where quantization noise is more noticeable, such as in relatively uniform areas. In this regards, α_{I} can be viewed as a perceptual weighting factor. Our bit rate allocation, however, works with any perceptual quantization function.

The problem of determining $P(I, Q)$ has been studied elsewhere [104, 15] and is not considered in this thesis. Here, we address the assignment of Q to each picture to give constant or near-constant quality among pictures while satisfying rate constraints imposed by the channel and decoder. We propose to compute Q at the picture level; that is, we compute one Q for each picture to be coded. Besides decreasing the computation over computing different Q for each block, this method

[1] The MPEG-2 Test Model 5 [48] also uses a multiplicative formulation while an additive formulation is proposed in [84].

results in constant perceptual quality within each picture given that perceptual quantization is employed at the block level. The framework can certainly be generalized to other coding units, and in principle can be applied to code other types of data, such as images and speech.

6.2 Constant Quality

Previous approaches in optimal rate control generally seeks to minimize a distortion measure, typically mean-squared error (MSE), averaged over coding blocks [80, 90]. While this approach leverages the wealth of tools from optimization theory and operations research, it does not guarantee the constancy of quality that is generally desired from a video coding system. For example, a video sequence with a constant or near-constant level of distortion is more desirable than one with lower average distortion but higher variability, because human viewers tend to find frequent changes in quality more noticeable and annoying. A long video sequence typically contains segments that, even if encoded at a fairly low bit rate, will not contain any disturbing quantization artifacts, so that improving the quality of pictures in those segments is far less important than improving the quality of pictures in segments that are more difficult to encode.

To address these issues, we propose a *lexicographic optimality* criterion that better expresses the desired constancy of quality. The idea is to minimize the maximum (perceptual) distortion of a block (or picture) and then minimize the second highest block distortion, and so on. The intuition is that doing so would equalize distortion by limiting peaks in distortion to their minimum. As we will show later, if a constant quality allocation is feasible, then it must necessarily be lexicographically optimal.

6.3 Bit-Production Modeling

For simplicity, we assume that each picture has a bit-production model that relates the picture's nominal quantization Q to the number of coded bits B. This assumes that the coding of one picture is independent of any other. This independence holds for an encoding that uses only intraframe (I) pictures, but not for one that uses forward predictive (P) or bidirectionally predictive (B) pictures, for example. In practice, the extent of the dependency is limited to small groups of pictures. Nonetheless, we initially assume independence to ease analysis and defer treatment of dependencies until a later chapter where we consider practical implementations.

We specify Q and B to be non-negative real-valued variables. In practice, the quantization scale Q_{s} and B are positive integers with $Q_{\mathrm{s}} = \lfloor \alpha_{\mathrm{I}} \cdot Q \rfloor$. However, to facilitate analysis, we assume that there is a continuous function for each picture that maps Q to B.

For a sequence of N pictures, we define N corresponding bit-production models $\{f_1, f_2, \ldots, f_N\}$ that map nominal quantization scale to bits: $b_i = f_i(q_i)$, where $f_i : [0, \infty] \mapsto [l_i, u_i]$, with $0 \leq l_i < u_i$. (We number pictures in encoding order and not temporal display order. See Section 3.15.) We require the models to have the following properties:

1. $f_i(0) = u_i$,
2. $f_i(\infty) = l_i$,
3. f_i is continuous and monotonically decreasing.

From these conditions, it follows that f_i is invertible with $q_i = g_i(b_i)$, where $g_i = f_i^{-1}$ and $g_i : [l_i, u_i] \mapsto [0, \infty]$. We note that g_i is also continuous and monotonically decreasing. Although monotonicity does not always hold in practice, it is a generally accepted assumption.

In video coding systems, the number of bits produced for a picture also depends upon a myriad of coding choices besides quantization scale, such as motion compensation and the mode used for coding each block. We assume that these choices are made independent of quantization and prior to performing rate control.

6.4 Buffer Constraints

The MPEG standards specify that an encoder should produce a bitstream that can be decoded by a hypothetical decoder referred to as the Video Buffering Verifier (VBV), as shown in Figure 3.23 and described in Section 3.7. Data can be transferred to the VBV either at a constant or variable bit rate.[2] In either mode of operation, the number of bits produced by each picture must be controlled so as to satisfy constraints imposed by the operation of the decoder buffer, whose size B_{VBV} is specified in the bitstream by the encoder. The encoder also specifies the maximum transfer rate $R_{\max}$ into the VBV buffer and the amount of time the decoder should wait before decoding the first picture. In this section, we consider constraints on the number of bits produced in each picture that follow from analysis of the VBV.

6.4.1 Constant Bit Rate

We first examine the mode of operation in which the compressed bitstream is to be delivered at a constant bit rate $R_{\max}$.

Definition 6.1 Given a sequence of N pictures, an *allocation* $s = \langle s_1, s_2, \ldots, s_N \rangle$ is an N-tuple containing bit allocations for all N pictures, so that s_n is the number of bits allocated to picture n.

Let B_{VBV} be the size of the decoder buffer. Let $B_{\mathrm{f}}(s, n)$ denote the buffer fullness (the number of bits in the VBV buffer), resulting from allocation s, just *before* the nth picture is removed from the buffer. Let $B_{\mathrm{f}}^*(s, n)$ denote the buffer fullness, resulting from allocation s, just *after* the nth picture is removed. Then

$$B_{\mathrm{f}}^*(s, n) = B_{\mathrm{f}}(s, n) - s_n. \tag{6.1}$$

Let $R_{\max}$ be the rate at which bits enter the decoding buffer. Let T_n be the amount of time required to display picture n. Then $B_{\mathrm{a}}(n) = R_{\max} T_n$ is the maximum number of bits that can enter the buffer in the time it takes to display picture n.

[2]The MPEG-1 standard only defines VBV operation with a constant bit rate while the MPEG-2 standard also allows for variable bit rate.

For constant bit rate (CBR) operation, the state of the VBV buffer is described by the recurrence

$$\begin{aligned} B_{\mathrm{f}}(s,1) &= B_1, \\ B_{\mathrm{f}}(s,n+1) &= B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - s_n, \end{aligned} \tag{6.2}$$

where B_1 is the initial buffer fullness. Unwinding the recurrence, we can also express (6.2) as

$$B_{\mathrm{f}}(s,n+1) = B_1 + \sum_{j=1}^{n} B_{\mathrm{a}}(j) - \sum_{j=1}^{n} s_j. \tag{6.3}$$

To prevent the decoder buffer from overflowing we must have

$$B_{\mathrm{f}}(s,n+1) \le B_{\mathrm{VBV}}. \tag{6.4}$$

The MPEG standards allow pictures to be skipped in certain applications. We assume that all pictures are coded, in which case all of picture n must arrive at the decoder by the time it is to be decoded and displayed; that is, we must have

$$B_{\mathrm{f}}(s,n) \ge s_n, \tag{6.5}$$

or equivalently,

$$B_{\mathrm{f}}^*(s,n) \ge 0. \tag{6.6}$$

A violation of this condition is called a buffer *underflow*.

We now have an upper bound and can derive a lower bound for the number of bits that we can use to code picture n. From (6.2) and (6.4) we have

$$s_n \ge B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - B_{\mathrm{VBV}}.$$

Since we cannot produce a negative number of bits, the lower bound on s_n is

$$s_n \ge \max\{B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - B_{\mathrm{VBV}}, 0\}. \tag{6.7}$$

In summary, for constant bit rate operation, in order to pass video buffer verification, an allocation s must satisfy the following for all n:

$$\max\{B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - B_{\mathrm{VBV}}, 0\} \le s_n \le B_{\mathrm{f}}(s,n). \tag{6.8}$$

An exemplary plot of the evolution of the buffer fullness over time for CBR operation is shown in Figure 6.1. In this example, the decoder waits T_0 seconds before decoding the first picture, at which time the buffer fullness is B_1. The time to display each picture is assumed to be a constant T seconds. In the plot, the upper and lower bounds for the number of bits to code picture 2 are shown as U_2 and L_2, respectively.

6.4.2 Variable Bit Rate

We now examine the scenario where the compressed video bitstream is to be delivered at a variable bit rate (VBR). Specifically, we adopt the second VBR mode of the MPEG-2 VBV model (see

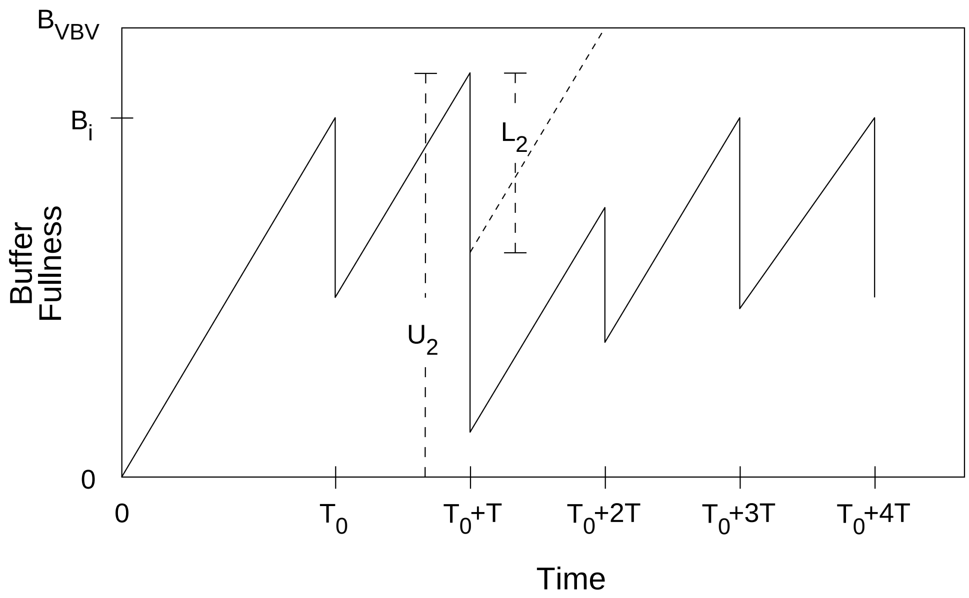


Figure 6.1: Sample plot of buffer fullness for CBR operation. Bits enter the decoder buffer at a constant rate until time T_0, when the first picture is removed. Successive pictures are removed after a time interval T. Upper and lower bounds on the number of bits that can be produced for the second picture are shown as U_2 and L_2, respectively.

Section 3.7.4) where bits always enter the decoder buffer at the peak rate $R_{\max}$ until the buffer is full.[3] Depending upon the state of the buffer, bits enter during each display interval at a rate that is effectively variable up to the peak rate $R_{\max}$. The maximum number of bits entering the buffer in the time it takes to display picture n is $B_{\mathrm{a}}(n) = R_{\max} T_n$.

For VBR operation, the state of the VBV buffer is described by:

$$\begin{aligned} B_{\mathrm{f}}(s,1) &= B_{\mathrm{VBV}}, \\ B_{\mathrm{f}}(s,n+1) &= \min\{B_{\mathrm{VBV}}, B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - s_n\}. \end{aligned} \tag{6.9}$$

Unlike the CBR case, the decoder buffer is prevented from overflowing by the minimization in (6.9). When $B_{\mathrm{f}}(s,n) + B_{\mathrm{a}}(n) - s_n > B_{\mathrm{VBV}}$, we say that picture n results in a *virtual overflow*. When a virtual overflow occurs, the effective input rate to the VBV buffer during that display interval is less than the peak rate. Like the CBR case, underflow is possible and to prevent it (6.5) must hold. The evolution of the buffer fullness is shown for VBR operation in Figure 6.2. The time to display each picture is assumed to be a constant T seconds. As shown in the plot, the number of bits that enter the buffer during each display interval is variable, with virtual overflows occurring for pictures 2 and 4.

[3] We note that with the same bit allocation, the VBV buffer fullness for the second VBR mode is always equal to or higher than the fullness when operating in the first VBR mode. Intuitively, if the channel bit rate is not further constrained, a lexicographically optimal bit allocation for the second VBR mode should not be worse (lexicographically) than an optimal bit allocation for the first VBR mode, given the same total bit budget.

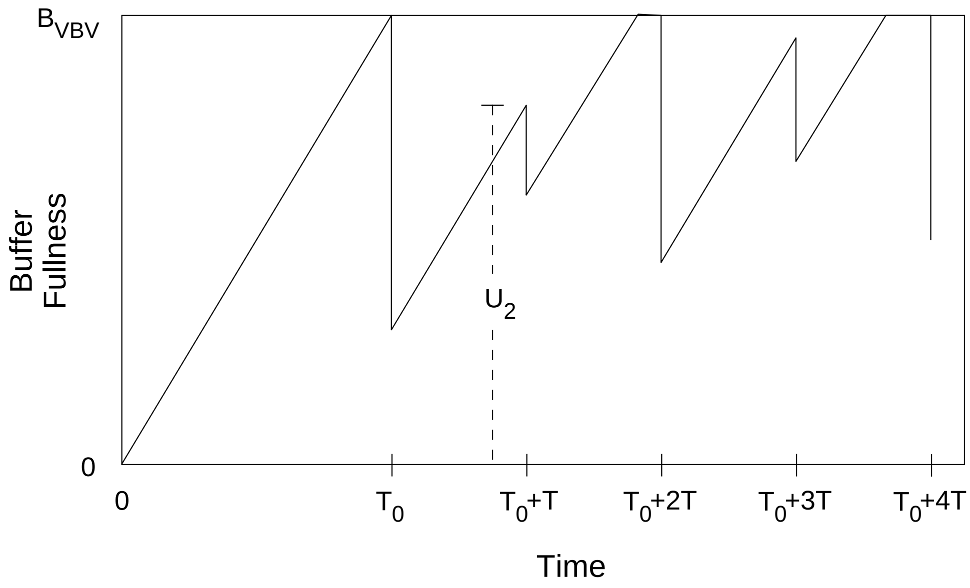


Figure 6.2: Sample plot of buffer fullness for VBR operation. Bits enter the decoder buffer at a constant rate until time T_0, when the first picture is removed. Successive pictures are removed after a time interval T. When the buffer becomes full, no more bits enter until the next picture is removed.

6.4.3 Encoder vs. Decoder Buffer

In the above discussion, we have focused solely on the decoder buffer whereas Figure 3.20 shows the Rate Controller monitoring the fullness of the encoder buffer. By assuming a fixed channel delay the encoder buffer fullness can be shown to mirror the decoder buffer fullness, except for an initial startup period. That is, an empty decoder buffer would correspond to a full encoder buffer, and vice versa. The reader is referred to [91] for a more complete discussion of buffer constraints in video coder systems.

6.5 Buffer-Constrained Bit-Allocation Problem

Using the bit-production model and VBV constraints defined above, we now formalize the buffer-constrained bit-allocation problem.

Definition 6.2 A *buffer-constrained bit-allocation problem* P is specified by a tuple

$$P = \langle N, F, B_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \rangle ,$$

where N is the number of pictures; $F = \langle f_1, f_2, \ldots, f_N \rangle$ is a sequence of N functions, as specified in Section 6.3, that model the relationship between the nominal quantization scale and the number of coded bits for each picture; B_{tgt} is the target number of bits to code all N pictures; B_{VBV} is the size of the VBV buffer in bits; B_1 is the number of bits initially in the VBV buffer; B_{a} is a function that gives the maximum number of bits that can enter the decoding buffer while each picture is being displayed.

For convenience, in the sequel we shall use the shorter term "bit-allocation problem" to refer to the buffer-constrained bit-allocation problem.

Definition 6.3 Given a bit-allocation problem $P = \langle N, F, B_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \rangle$, an allocation s is a *legal allocation* if the following conditions hold:

1. $\sum_{j=1}^{N} s_j = B_{\mathrm{tgt}}$
2. Equation (6.5) holds: $B_{\mathrm{f}}(s, n) \geq s_n$.
3. For CBR only, (6.7) holds: $s_n \geq \max\{B_{\mathrm{f}}(s, n) + B_{\mathrm{a}}(n) - B_{\mathrm{VBV}}, 0\}$.

In order for a CBR bit-allocation problem to have a legal allocation, we must have

$$B_{\mathrm{VBV}} \geq \max_j B_{\mathrm{a}}(j). \tag{6.10}$$

Also, the buffer fullness at the end of the sequence must be within bounds. For an allocation s, from (6.1) and (6.3) we have

$$B_{\mathrm{f}}^*(s, N) = B_1 + \sum_{j=1}^{N-1} B_{\mathrm{a}}(j) - B_{\mathrm{tgt}}. \tag{6.11}$$

The bound on $B_{\mathrm{f}}^*(s, N)$ is thus

$$0 \leq B_{\mathrm{f}}^*(s, N) \leq B_{\mathrm{VBV}}. \tag{6.12}$$

From (6.11) and (6.12), we have the following CBR bounds on B_{tgt}:

$$B_1 + \sum_{j=1}^{N-1} B_{\mathrm{a}}(j) - B_{\mathrm{VBV}} \leq B_{\mathrm{tgt}} \leq B_1 + \sum_{j=1}^{N-1} B_{\mathrm{a}}(j). \tag{6.13}$$

A VBR bit-allocation problem does not have a lower bound for the target bit rate B_{tgt} since the VBV does not impose a lower bound on the number of bits produced by each picture. The upper bound for B_{tgt} depends upon whether $\max_{1 \leq j \leq N}\{B_{\mathrm{a}}(j)\} > B_{\mathrm{VBV}}$. In general, the VBR upper bound on B_{tgt} is

$$B_{\mathrm{tgt}} \leq B_1 + \sum_{j=1}^{N-1} \min\{B_{\mathrm{a}}(j), B_{\mathrm{VBV}}\}. \tag{6.14}$$

However, in the sequel, we assume that $\max_j B_{\mathrm{a}}(j) \leq B_{\mathrm{VBV}}$. We also assume that bit-allocation problems are given so that a legal allocation exists.

6.6 Lexicographic Optimality

We now formally define the lexicographic optimality criterion. As mentioned in Section 6.1, we equate nominal quantization scale with perceptual distortion and define the optimality criterion based upon the nominal quantization Q assigned to each picture.

Let S be the set of all legal allocations for a bit-allocation problem P. For an allocation $s \in S$, let $\mathbf{Q}^s = \langle Q_1^s, Q_2^s, \ldots, Q_N^s \rangle$ be the values of Q to achieve the bit allocation specified by s. Thus $Q_i^s = g_i(s_i)$, where g_i is as defined in Section 6.3. Ideally, we would like an optimal allocation to use a constant nominal quantization scale. However, this may not be feasible because of buffer constraints. We could consider minimizing an l_k norm of $\mathbf{Q}^s$. However, as discussed earlier, such an approach does not guarantee constant quality where possible and may result in some pictures having extreme values of Q_i.

Instead, we would like to minimize the maximum Q_i. Additionally, given that the maximum Q_i is minimized, we want the second largest Q_i to be as small as possible, and so on. This is referred to as *lexicographic optimality* in the literature (e.g., [46]).

We define a sorted permutation DEC on $\mathbf{Q}^s$ such that for $\text{DEC}(\mathbf{Q}^s) = \langle q_{j_1}, q_{j_2}, \ldots, q_{j_N} \rangle$, we have $q_{j_1} \geq q_{j_2} \geq \cdots \geq q_{j_N}$. Let $\text{rank}(s, k)$ be the kth element of $\text{DEC}(\mathbf{Q}^s)$; that is, $\text{rank}(s, k) = q_{j_k}$. We define a binary relation $\succ$ on allocations as follows: $s = \langle s_1, \ldots, s_N \rangle \succ s' = \langle s'_1, \ldots, s'_N \rangle$ if and only if $\text{rank}(s, j) = \text{rank}(s', j)$ for $j = 1\ ,2\ ,\ldots,\ k-1$ and $\text{rank}(s, k) > \text{rank}(s', k)$ for some $1 \leq k \leq N$. We also define $s \prec s'$ if and only if $s' \succ s$; $s \asymp s'$ if and only if $\text{rank}(s, j) = \text{rank}(s', j)$ for all j; $s \succeq s'$ if and only if $s \succ s'$ or $s \asymp s'$; and $s \preceq s'$ if and only if $s \prec s'$ or $s \asymp s'$.

Definition 6.4 A legal allocation s^* is *lexicographically optimal* if $s^* \preceq s$ for all other legal allocation s.

Lemma 6.1 (Constant-Q) *Given a bit-allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{VBV}}, B_1, B_{\text{a}} \rangle$, if there exists a legal allocation s and a quantization q such that $g_n(s_n)$ is the constant quantization q for all n, where g_n is defined as in Section 6.3, then s is the only lexicographically optimal allocation for P.*

Proof: First we prove that s is optimal. Since s is a legal allocation, we have

$$\sum_{j=1}^{N} s_j = \sum_{j=1}^{N} f_j(q) = B_{\text{tgt}}.$$

Suppose that s is not optimal. Let s' be an optimal allocation. Then $\text{rank}(s', k) < \text{rank}(s, k) = q$ for some k, and $\text{rank}(s', j) \leq \text{rank}(s, j)$ for all j. Therefore $s'_l > f_l(q)$ for some l and $s'_j \geq f_j(q)$ for all j since f_j is a decreasing function. Thus

$$\sum_{j=1}^{N} s'_j > \sum_{j=1}^{N} f_j(q) = B_{\text{tgt}}.$$

So s' is not a legal allocation, a contradiction. Therefore s is optimal.

Now we show that s is the only optimal allocation. Let s' be an optimal allocation. Since s and s' are both optimal, $s \preceq s'$ and $s \succeq s'$, implying $s \asymp s'$. Then $\text{rank}(s, j) = \text{rank}(s', j)$ for all j. Therefore $\text{rank}(s', j) = q$ for all j. Thus $s' = s$. $\square$

Lemma 6.1 establishes a desirable property of the lexicographic optimality criterion: If a constant-Q allocation is legal, it is the only lexicographically optimal allocation. This meets our objective of obtaining a constant-quality allocation (via perceptual quantization) when feasible.

6.7 Related Work

In [98], the budget-constrained bit-allocation problem (see Section 3.3.3) is examined in the context of a discrete set of independent quantizers. A bit allocation algorithm based upon Lagrangian minimization is presented as a more efficient alternative to the well-known dynamic programming solution based upon the Viterbi algorithm [105, 28]. This work lays the foundation for much of the ensuing work on optimal bit allocation.

Optimal budget-constrained bit allocation in a dependent, predictive coding setting is examined in [103]. A parametric rate-distortion model is proposed for intraframe coding and forward predictive coding. The model has an exponential form and is motivated by theoretical rate-distortion results for stationary Gaussian sources. Lagrangian minimization is chosen as the optimization technique and a closed-form solution is obtained in terms of known statistics and the Lagrange multiplier. A search over the Lagrange multiplier then yields a solution to the budget-constrained problem. The authors acknowledge that minimizing sum-distortion does not lead to uniform distortion. They reformulate the problem to minimize the maximum (minimax) picture distortion by equating the distortion among pictures.[4]

Budget-constrained minimax bit allocation for dependent coding is also considered in [95]. The authors provide a minimax solution by first showing how to find a minimum-rate solution given a maximum distortion and then using an bisection search to find the maximum distortion corresponding to the desired rate. However, the bisection search is not guaranteed to converge in a finite number of iterations.

In [91], bit-rate constraints for buffered video coders are derived for a general variable-bit-rate channel, such as that provided by an ATM network. The constraints take into account both the encoder and decoder buffers. The bit-rate constraints are used in an ad-hoc algorithm that jointly selects the channel and encoder rates.

The problem of optimal bit allocation in a buffered video coder is considered in [79]. The authors consider video coding with CBR buffer constraints and formulate bit allocation as an integer programming problem. They assume a finite set of quantization scales, an integral number of coded bits, and independent coding. The problem is optimally solved using a dynamic programming algorithm based upon the Viterbi algorithm (as described in Section 3.3.3a). Heuristic methods based upon Lagrangian minimization and other ad-hoc techniques are proposed to provide more efficient, but sub-optimal, solutions.

The discrete optimization framework of [79] is extended in [89] to handle dependent coding. Except for a simple illustrative case, computing an optimal bit allocation under the dependent framework requires time and space exponential in the number of coding units. A heuristic pruning technique is proposed to reduce the number of states considered. However, the effectiveness of the heuristic depends upon the rate-distortion characteristics of the source.

In [43], the work of [79] is further extended to include transmission over a variable-bit-rate channel with delay constraints. Besides buffer and delay constraints, the authors also consider constraints imposed by several policing mechanisms proposed for ATM networks. Assuming a

[4] A minimax solution is produced by our lexicographic framework since lexicographic optimality is a refinement of minimax.

discrete set of quantizers and a discrete set of transmission rates, the quantization and transmission rate can be jointly optimized using the Viterbi algorithm to produce a minimum sum-distortion encoding. In the construction of the trellis used by the Viterbi algorithm, states that violate the various constraints are discarded. Unlike our framework, there is no explicit constraint on the total number of bits used.

Joint control of encoder and channel rate is also considered in [22]. Instead of considering global optimality, this work focuses on real-time control algorithms. An algorithm is proposed that separates rate control into a "short-term" process and a "long-term" process. The long term rate control sets a base quantization scale Q_s called the *sequence quantization parameter*.[5] In normal operation, Q_s is used to code each picture. Long-term rate control monitors the average fullness of a virtual encoder buffer and adjusts Q_s to maintain the buffer fullness between two thresholds. Short-term rate control is applied when the upper bound on encoder rate needs to be enforced. Several methods are proposed for performing short-term rate control.

In [10], a model relating bits, distortion, and quantization scale is derived for block-transform video coders. Assuming a stationary Gaussian process, the authors derive a bit-production model containing transcendental functions. The model is applied to control the frame rate of motion-JPEG and H.261 video coders.

In the operations research literature, lexicographic optimality has been applied to such problems as resource location and allocation (e.g., [31, 53, 66, 68, 77, 85]) and is sometimes referred to as *lexicographic minimax*, since it can be viewed as a refinement of minimax theory.

6.8 Discussion

As described above, much of the previous work on optimal rate control for video coding use the conventional rate-distortion approach of minimizing a sum-distortion measure with budget and buffer constraints. A drawback of this approach is that it does not directly achieve the constancy in quality that is generally desired. In contrast, our proposed framework, based upon a novel lexicographic optimality criterion, is aimed at achieving constant or near-constant quality video. We incorporate into the framework a set of buffer constraints based upon the Video Buffering Verifier of the popular MPEG video coding standards.

In the following chapters, we analyze rate control under both CBR and VBR constraints. We derive a set necessary and sufficient conditions for optimality. These conditions, intuitive and elegant, lead to efficient algorithms for computing optimal allocations in polynomial time and linear space.

In Chapter 10, we describe some extensions to the lexicographic framework. We show how to apply the framework to allocate bits to multiple VBR streams for transport over a CBR channel. We also show how to adapt the discrete framework of [79] to perform lexicographic optimization.

[5] This is similar to the nominal quantization scale defined in Section 6.1.

Chapter 7

Lexicographic Bit Allocation under CBR Constraints

In this chapter, we analyze the buffer-constrained bit-allocation problem under constant-bit-rate VBV constraints, as described in Section 6.4.1. The analysis leads to an efficient dynamic programming algorithm for computing a lexicographically optimal solution.

Before proceeding with a formal theoretical treatment, we first present some intuition for the results that follow. If we consider a video sequence as being composed of segments of differing coding difficulty, a segment of "easy" pictures can be coded at a higher quality (lower distortion) than an immediately following segment of "hard" pictures if we code each segment at a constant bit rate. Since we have a decoder buffer, we can vary the bit rate to some degree, depending upon the size of the buffer. If we could somehow "move" bits from the easy segment to the hard segment, we would be able to code the easy segment at a lower quality than before and the hard segment at a higher quality, thereby reducing the difference in quality between the two segments. In terms of the decoder buffer, this corresponds to filling up the buffer during the coding of the easy pictures, which are coded with less than the average bit rate. By use of the accumulated bits in the buffer, the hard pictures can be coded with effectively more than the average bit rate.

Similarly, suppose we have a hard segment followed by an easy segment. We would like to empty the buffer during the coding of the hard pictures to use as many bits as the buffer allows to code the hard pictures at above the average bit rate. This simultaneously leaves room in the buffer to accumulate excess bits resulting from coding the easy pictures below the average bit rate.

This behavior of emptying and filling the buffer is intuitively desirable since this means that we are taking advantage of the full capacity of the buffer. In the following analysis, we will show that such a behavior is indeed exhibited by a lexicographically optimal bit allocation.

7.1 Analysis

First, we seek to prove necessary conditions for lexicographic optimality. To do so, we use the following lemma.

Lemma 7.1 *Given two allocations s and s' of size N that satisfy $s_k = s'_k$ if and only if $k \notin \{u, v\}$, if $\max\{g_u(s'_u), g_v(s'_v)\} < \max\{g_u(s_u), g_v(s_v)\}$ then $s' \prec s$.*

Proof: Suppose $\max\{g_u(s'_u), g_v(s'_v)\} < \max\{g_u(s_u), g_v(s_v)\}$. Let j be the greatest index such that $\text{rank}(s, j) = \max\{g_u(s_u), g_v(s_v)\}$. Then $\text{rank}(s, j) > \text{rank}(s, j+1)$. Consider $\text{rank}(s', j)$. Either $\text{rank}(s', j) = \text{rank}(s, j+1)$ or $\text{rank}(s', j) = \max\{g_u(s'_u), g_v(s'_v)\}$. In either case, $\text{rank}(s, j) > \text{rank}(s', j)$. Therefore $s' \prec s$. □

The following lemma establishes necessary conditions for an optimal allocation.

Lemma 7.2 *Given a CBR bit-allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{VBV}}, B_1, B_{\text{a}} \rangle$, if s^* is an optimal allocation, the following are true:*

1. *If $g_j(s^*_j) > g_{j+1}(s^*_{j+1})$ for some $1 \le j < N$ then $B_{\text{f}}(s^*, j) = s^*_j$.*
2. *If $g_j(s^*_j) < g_{j+1}(s^*_{j+1})$ for some $1 \le j < N$ then $B_{\text{f}}(s^*, j+1) = B_{\text{VBV}}$.*

Lemma 7.2 gives us a set of necessary "switching" conditions for optimality. It states that an optimal allocation consists of segments of constant Q, with changes in Q occurring only at buffer boundaries. Also, Q must change in a specific manner depending upon whether the buffer is full or empty. We observe that in an optimal allocation, the decoder buffer is full before decoding starts on a relatively difficult scene, which is marked by an increase in Q (case 2). This policy makes the entire capacity of the decoder buffer available to code the more difficult pictures. On the other hand, before decoding a relatively easy scene, which is marked by a decrease in Q (case 1), the buffer is emptied in order to provide the most space to accumulate bits when the easy scene uses less than the average bit rate. These observations agree with the intuitions provided earlier.

A sketch of the proof of Lemma 7.2 is shown in Figure 7.1. The proof is by contradiction. In the figure, the VBV buffer is shown for a hypothetical situation in which $Q_1 < Q_2$ and $Q_2 > Q_3$ and the switching conditions are not met.

In the first case, for $Q_1 < Q_2$, if the buffer is not full before picture 2 is decoded, an alternate allocation can be constructed that is the same as the allocation shown except that the VBV plot follows the dashed line for the segment between pictures 1 and 2. The dashed line results from increasing Q_1 and decreasing Q_2 while still maintaining $Q_1 < Q_2$ and not causing the buffer to overflow. This results in a better allocation than before. Intuitively, this corresponds to shifting bits left-to-right from a relatively easy picture (lower Q) to a relatively hard picture (higher Q). This shifting of bits can take place until the buffer becomes full.

In the second case, for $Q_2 > Q_3$, if the buffer is not empty after picture 2 is decoded, an alternate allocation can be constructed that is the same as the allocation shown except that the VBV plot follows the dotted line for the segment between pictures 2 and 3. The dotted line results from decreasing Q_2 and increasing Q_3 while still maintaining $Q_2 > Q_3$ and not causing the buffer to underflow. This results in a better allocation than before. Intuitively, this corresponds to shifting bits right-to-left from a relatively easy picture (lower Q) to a relatively hard picture (higher Q). This shifting of bits can take place until the buffer becomes empty.

We note that Lemma 6.1 follows directly from Lemma 7.2.

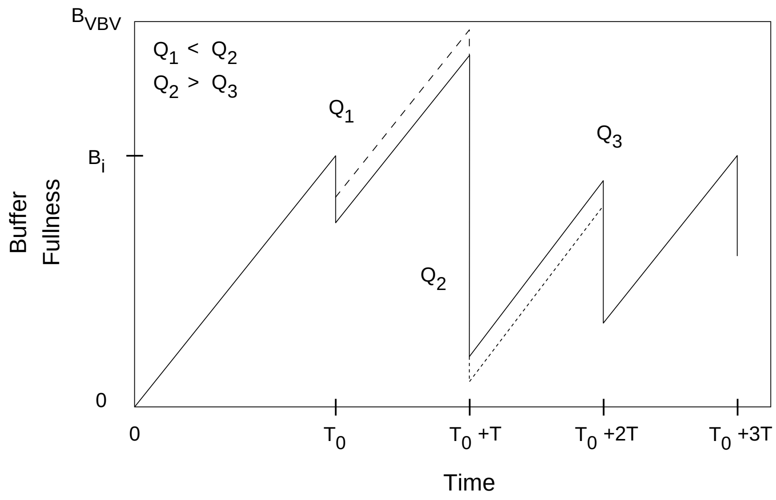


Figure 7.1: Sketch for proof of Lemma 7.2.

Proof of Lemma 7.2:
Case 1. We prove Case 1 by contradiction. Suppose $B_{\rm f}(s^*, j) \neq s_j^*$. Let $\Delta = B_{\rm f}(s^*, j) - s_j^*$. Then by (6.5), $\Delta > 0$. Consider an allocation s' that differs from s^* only for pictures j and $j+1$; that is,

$$s'_k = s^*_k \quad \text{for } k \in \{1, \ldots, N\} \setminus \{j, j+1\} \qquad \text{and} \qquad s'_k \neq s^*_k \quad \text{for } k \in \{j, j+1\}\,. \tag{7.1}$$

In order to show a contradiction, we want to find an assignment to s'_j and s'_{j+1} that makes s' a legal allocation and "better" than s^*. By "better" we mean

$$g_j(s'_j), g_{j+1}(s'_{j+1}) < g_j(s^*_j). \tag{7.2}$$

Equivalently, we want

$$s'_j > s^*_j \tag{7.3}$$

and

$$s'_{j+1} > f_{j+1}(g_j(s^*_j)). \tag{7.4}$$

To meet the target bit rate, we must have

$$s'_j + s'_{j+1} = s^*_j + s^*_{j+1}. \tag{7.5}$$

Let $\delta = s'_j - s^*_j$. Then $s^*_{j+1} - s'_{j+1} = \delta$. By (7.3), we want $\delta > 0$. We want to show that s' is a legal allocation for some value of $\delta > 0$. To avoid VBV violations, (6.8) must hold for all pictures under the allocation s'. From (7.1) and (7.5), we have

$$B_{\rm f}(s', k) = B_{\rm f}(s^*, k) \quad \text{for } k \neq j+1. \tag{7.6}$$

Since s^* is a legal allocation, there are no VBV violations for pictures 1, 2, …, $j-1$ under s'. Furthermore, if our choice for s'_j does not cause a VBV violation for picture j, then we are assured that there would be no VBV violations in pictures $j+1$, $j+2$, …, N. So we must choose s'_j subject to (6.8) and (7.3). Therefore

$$s^*_j < s'_j \le s^*_j + \Delta. \tag{7.7}$$

If $0 < \delta \le \Delta$, then s' is a legal allocation. We also want (7.4) to hold. For this we need

$$\delta < s^*_{j+1} - f_{j+1}(g_j(s^*_j)). \tag{7.8}$$

Since $g_j(s^*_j) > g_{j+1}(s^*_{j+1})$, we have $f_{j+1}(g_j(s^*_j)) < s^*_{j+1}$. Therefore $s^*_{j+1} - f_{j+1}(g_j(s^*_j)) > 0$. So for

$$0 < \delta \le \min\{\Delta, s^*_{j+1} - f_{j+1}(g_j(s^*_j))\} \tag{7.9}$$

s' is a legal allocation that meets condition (7.2). By Lemma 7.1, $s^* \succ s'$ and s^* is not an optimal allocation, a contradiction.

Case 2. We prove Case 2 by contradiction. Suppose $B_{\mathrm{f}}(s^*, j+1) \ne B_{\mathrm{VBV}}$. Let $\Delta = B_{\mathrm{VBV}} - B_{\mathrm{f}}(s^*, j+1)$. Then by (6.4), $\Delta > 0$. Consider an allocation s' that differs from s^* only for pictures j and $j+1$; that is,

$$s'_k = s^*_k \text{ for } k \in \{1, \ldots, N\} \setminus \{j, j+1\}\,. \tag{7.10}$$

We want to find an assignment to s'_j and s'_{j+1} that makes s' a legal allocation and "better" than s^*, in order to show a contradiction. By "better" we mean

$$g_j(s'_j), g_{j+1}(s'_{j+1}) < g_{j+1}(s^*_{j+1}). \tag{7.11}$$

Equivalently, we want

$$s'_{j+1} > s^*_{j+1} \tag{7.12}$$

and

$$s'_j > f_j(g_{j+1}(s^*_{j+1})). \tag{7.13}$$

To meet the target bit rate, we must have

$$s'_j + s'_{j+1} = s^*_j + s^*_{j+1}. \tag{7.14}$$

Let $\delta = s'_{j+1} - s^*_{j+1}$. Then $s^*_j - s'_j = \delta$. By (7.12), we want $\delta > 0$. We want to show that s' is a legal allocation for some value of $\delta > 0$. To avoid VBV violations, (6.8) must hold for all pictures under the allocation s'. From (7.10) and (7.14), we have

$$B_{\mathrm{f}}(s', k) = B_{\mathrm{f}}(s^*, k) \quad \text{for } k \ne j+1. \tag{7.15}$$

Since s^* is a legal allocation, there are no VBV violations for pictures 1 to $j-1$ under s'. Furthermore, if our choice for s'_j does not cause a VBV violation, then we are assured that there would be no VBV violations in pictures $j+1$ to N. So we must choose s'_j subject to (6.8) and (7.12).

$$\begin{aligned}
B_{\mathrm{f}}(s', j+1) &= B_{\mathrm{f}}(s', j) + B_{\mathrm{a}}(j) - s'_j \\
B_{\mathrm{f}}(s', j+1) &= B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s'_j \\
B_{\mathrm{f}}(s^*, j+1) &= B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s^*_j \\
B_{\mathrm{f}}(s', j+1) &= B_{\mathrm{f}}(s^*, j+1) + s^*_j - s'_j \\
B_{\mathrm{f}}(s', j+1) &= B_{\mathrm{VBV}} - \Delta + \delta \le B_{\mathrm{VBV}}
\end{aligned}$$

From the above, we require $\delta \le \Delta$. If $0 < \delta \le \Delta$, then s' is a legal allocation. We also want (7.13) to hold. For this we need

$$\delta < s_j^* - f_j(g_{j+1}(s_{j+1}^*)). \tag{7.16}$$

Since $g_{j+1}(s_{j+1}^*) > g_j(s_j^*)$, we have $f_j(g_{j+1}(s_{j+1}^*)) < s_j^*$. Therefore $s_j^* - f_j(g_{j+1}(s_{j+1}^*)) > 0$. So for

$$0 < \delta \le \min\{\Delta, s_j^* - f_j(g_{j+1}(s_{j+1}^*))\} \tag{7.17}$$

s' is a legal allocation that meets condition (7.13). By Lemma 7.1, $s^* \succ s'$ and s^* is not an optimal allocation, a contradiction.

□

The theorem that follows is the main result of this section and shows that the switching conditions are also sufficient for optimality. But first we prove a useful lemma that will be helpful in the proof of the theorem.

Lemma 7.3 *Given bit-allocations s and s' with $s_j \le s_j'$ for $u \le j \le v$ and $B_\mathrm{f}(s,u) \ge B_\mathrm{f}(s',u)$, we have $B_\mathrm{f}(s,v+1) = B_\mathrm{f}(s',v+1)$ if and only if $B_\mathrm{f}(s,u) = B_\mathrm{f}(s',u)$ and $s_j = s_j'$ for $u \le j \le v$.*

Proof: We use (6.3) to express $B_\mathrm{f}(s,v+1)$ in terms of $B_\mathrm{f}(s,u)$.

$$\begin{aligned} B_\mathrm{f}(s,v+1) &= B_1 + \sum_{j=1}^{v} (B_\mathrm{a}(j) - s_j) \\ B_\mathrm{f}(s,u) &= B_1 + \sum_{j=1}^{u-1} (B_\mathrm{a}(j) - s_j) \\ B_\mathrm{f}(s,v+1) &= B_\mathrm{f}(s,u) + \sum_{j=u}^{v} (B_\mathrm{a}(j) - s_j)\,. \end{aligned}$$

Similarly,

$$B_\mathrm{f}(s',v+1) = B_\mathrm{f}(s',u) + \sum_{j=u}^{v} \left(B_\mathrm{a}(j) - s_j'\right).$$

First we prove the "if" part. Suppose $B_\mathrm{f}(s,u) = B_\mathrm{f}(s',u)$ and $s_j = s_j'$ for $u \le j \le v$. Then

$$\begin{aligned} B_\mathrm{f}(s,v+1) &= B_\mathrm{f}(s,u) + \sum_{j=u}^{v} (B_\mathrm{a}(j) - s_j) \\ &= B_\mathrm{f}(s',u) + \sum_{j=u}^{v} \left(B_\mathrm{a}(j) - s_j'\right) \\ &= B_\mathrm{f}(s',v+1). \end{aligned}$$

Now we prove the "only if" part. Suppose $B_\mathrm{f}(s,v+1) = B_\mathrm{f}(s',v+1)$. Then

$$\begin{aligned} B_\mathrm{f}(s,v+1) &= B_\mathrm{f}(s',v+1) \\ B_\mathrm{f}(s,u) + \sum_{j=u}^{v} (B_\mathrm{a}(j) - s_j) &= B_\mathrm{f}(s',u) + \sum_{j=u}^{v} \left(B_\mathrm{a}(j) - s_j'\right) \\ B_\mathrm{f}(s,u) - B_\mathrm{f}(s',u) &= \sum_{j=u}^{v} (s_j - s_j'). \end{aligned} \tag{7.18}$$

But $B_{\mathrm{f}}(s,u) \geq B_{\mathrm{f}}(s',u)$ and $s_j \leq s'_j$ for $u \leq j \leq v$. Therefore $B_{\mathrm{f}}(s,u) - B_{\mathrm{f}}(s',u) \geq 0$ and $\sum_{j=u}^{v}(s_j - s'_j) \leq 0$. Combined with (7.18), this implies that $B_{\mathrm{f}}(s,u) = B_{\mathrm{f}}(s',u)$ and $\sum_{j=u}^{v} s_j = \sum_{j=u}^{v} s'_j$. Since $s_j \leq s'_j$ for $u \leq j \leq v$, this implies that $s_j = s'_j$ for $u \leq j \leq v$. □

Theorem 7.1 *Given a CBR bit-allocation problem $P = \langle N, F, B_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \rangle$, a legal allocation s is optimal if and only if the following conditions hold. Also, the optimal allocation is unique.*

1. *If $g_j(s_j) > g_{j+1}(s_{j+1})$ for some $1 \leq j < N$, then $B_{\mathrm{f}}(s,j) = s_j$.*
2. *If $g_j(s_j) < g_{j+1}(s_{j+1})$ for some $1 \leq j < N$, then $B_{\mathrm{f}}(s,j+1) = B_{\mathrm{VBV}}$.*

Proof: Lemma 7.2 established these condition as necessary for optimality. Now we need to show that these conditions are also sufficient for optimality and imply uniqueness. Let s be a legal allocation that meets both conditions of the theorem.

Let s^* be an optimal allocation for P. Let $Q_{\max} = \max_{1 \leq j \leq N}\{g_j(s_j)\}$. Consider the segments of consecutive pictures that are assigned $Q_{\max}$ by allocation s. Let u be the index of the start of such a segment. There are two cases: $u = 1$ and $u > 1$. If $u = 1$, then $B_{\mathrm{f}}(s,u) = B_{\mathrm{f}}(s^*,u) = B_1$. If $u > 1$, then since u is the index of the start of the segment, $g_{u-1}(s_{u-1}) < g_u(s_u)$ which implies that $B_{\mathrm{f}}(s,u) = B_{\mathrm{VBV}}$ by condition 2. Since s^* is a legal allocation, $B_{\mathrm{f}}(s^*,u) \leq B_{\mathrm{VBV}} = B_{\mathrm{f}}(s,u)$. In either case we have

$$B_{\mathrm{f}}(s,u) \geq B_{\mathrm{f}}(s^*,u). \tag{7.19}$$

Let v be the index of the end of the segment. Since s^* is optimal, $g_j(s^*_j) \leq Q_{\max}$ for all j, and

$$s^*_j \geq s_j \quad \text{for } u \leq j \leq v. \tag{7.20}$$

Therefore

$$B_{\mathrm{f}}(s^*,j) \leq B_{\mathrm{f}}(s,j) \quad \text{for } u \leq j \leq v. \tag{7.21}$$

There are two cases for v: $v = N$ and $v < N$. If $v = N$, then $B_{\mathrm{f}}(s,v+1) = B_{\mathrm{f}}(s^*,v+1) = B_1 + \sum_{j=1}^{N-1} B_{\mathrm{a}}(j) - B_{\mathrm{tgt}}$. If $v < N$, then since v is the index of the end of the segment, $g_v(s_v) > g_{v+1}(s_{v+1})$ which implies that

$$B_{\mathrm{f}}(s,v) = s_v \tag{7.22}$$

by condition 1. Since s^* is a legal allocation, $B_{\mathrm{f}}(s^*,v) \geq s^*_v$. Combine this with (7.22) and (7.20) we have

$$B_{\mathrm{f}}(s^*,v) \geq B_{\mathrm{f}}(s,v). \tag{7.23}$$

Combining (7.21) and (7.23), we have $B_{\mathrm{f}}(s^*,v) = B_{\mathrm{f}}(s,v)$ and $s^*_v = s_v$. As a result, $B_{\mathrm{f}}(s^*,v+1) = B_{\mathrm{f}}(s,v+1)$. In either case, $B_{\mathrm{f}}(s^*,v+1) = B_{\mathrm{f}}(s,v+1)$. By Lemma 7.3, $B_{\mathrm{f}}(s^*,u) = B_{\mathrm{f}}(s,u)$ and $s^*_j = s_j$ for $u \leq j \leq v$. As a consequence, $B_{\mathrm{f}}(s,j) = B_{\mathrm{f}}(s^*,j)$ for $u \leq j \leq v$.

We partition pictures 1 through N into segments, where, according to allocation s, the pictures in a segment use the same value of Q, either the first picture in the segment is picture 1 or the first picture in the segment uses a value of Q different from the previous picture, and either the last picture in the segment is picture N or the last picture in the segment uses a value of Q different from the next picture. Let M be the number of such segments. We order the segments so that segment k uses a value of Q greater than or equal to the value of Q used in segment j for $j < k$, and denote the value of Q used by segment k as $Q_k^{(s)}$.

We will show that allocation s^* uses the same number of bits as allocation s for each picture in segment k, for $1 \leq k \leq M$. This will establish the conditions in the theorem as necessary and show that the optimal allocation is unique. We will prove this by induction on k. The base case of $k = 1$ has already been proven.

Inductive Hypothesis: For all segments of pictures u to v with $u = 1$ or $g_{u-1}(s_{u-1}) \neq g_u(s_u)$, $v = N$ or $g_v(s_v) \neq g_{v+1}(s_{v+1})$, and $g_j(s_j) = Q_k^{(s)}$, we have $s_j^* = s_j$ and $B_{\mathrm{f}}(s, u) = B_{\mathrm{f}}(s^*, u)$ for $u \leq j \leq v$.

Let us assume that the hypothesis is true for $1 \leq k < m$. We will show that is is also true for $k = m$. Consider a segment of consecutive pictures that are assigned quantization $Q_m^{(s)}$ by allocation s. Let u be the index of the start of the segment and v the index of the end of the segment.

By the inductive hypothesis, s and s^* use the same values of Q for all pictures for which s uses $Q > Q_m^{(s)}$. Because s^* is optimal, $g_j(s_j^*) \leq g_j(s_j) = Q_m^{(s)}$ for $u \leq j \leq v$, and thus

$$s_j^* \geq s_j \quad \text{for } u \leq j \leq v. \tag{7.24}$$

We consider all cases for the segment boundaries. For the left segment boundary there are three cases: $u = 1$, $g_{u-1}(s_{u-1}) > g_u(s_u)$, and $g_{u-1}(s_{u-1}) < g_u(s_u)$. If $u = 1$, then $B_{\mathrm{f}}(s^*, u) = B_{\mathrm{f}}(s, u) = B_1$. If $g_{u-1}(s_{u-1}) > g_u(s_u)$, then from the inductive hypothesis, we have $B_{\mathrm{f}}(s^*, u-1) = B_{\mathrm{f}}(s, u-1)$ and $s_{u-1}^* = s_{u-1}$; therefore $B_{\mathrm{f}}(s^*, u) = B_{\mathrm{f}}(s, u)$. If $g_{u-1}(s_{u-1}) < g_u(s_u)$, then from condition 2, we have $B_{\mathrm{f}}(s, u) = B_{\mathrm{VBV}}$; since s^* is a legal allocation, $B_{\mathrm{f}}(s^*, u) \leq B_{\mathrm{VBV}} = B_{\mathrm{f}}(s, u)$. For all three cases we have

$$B_{\mathrm{f}}(s^*, u) \leq B_{\mathrm{f}}(s, u). \tag{7.25}$$

For the right segment boundary there are three cases: $v = N$, $g_v(s_v) < g_{v+1}(s_{v+1})$, and $g_v(s_v) > g_{v+1}(s_{v+1})$. If $v = N$, then $B_{\mathrm{f}}(s^*, v+1) = B_{\mathrm{f}}(s, v+1) = B_1 + \sum_{j=1}^{N-1} B_{\mathrm{a}}(j) - B_{\mathrm{tgt}}$. If $g_v(s_v) < g_{v+1}(s_{v+1})$, then from the inductive hypothesis, we have $B_{\mathrm{f}}(s^*, v+1) = B_{\mathrm{f}}(s, v+1)$.

If $g_v(s_v) > g_{v+1}(s_{v+1})$, then from condition 1 we have $B_{\mathrm{f}}(s, v) = s_v$. From (7.24) and (7.25) we have

$$B_{\mathrm{f}}(s^*, j) \leq B_{\mathrm{f}}(s, j) \quad \text{for } u \leq j \leq v. \tag{7.26}$$

Since s^* is a legal allocation,

$$B_{\mathrm{f}}(s^*, v) \geq s_v^* \geq s_v = B_{\mathrm{f}}(s, v). \tag{7.27}$$

Combining (7.26) and (7.27), we have $B_{\mathrm{f}}(s^*, v) = B_{\mathrm{f}}(s, v)$ and $s_v^* = s_v$. Therefore $B_{\mathrm{f}}(s^*, v+1) = B_{\mathrm{f}}(s, v+1)$.

For all three cases we have

$$B_{\mathrm{f}}(s^*, v+1) = B_{\mathrm{f}}(s, v+1). \tag{7.28}$$

From (7.24), (7.25), (7.28), and Lemma 7.3, we have $s_j^* = s_j$ for $u \leq j \leq v$. It follows that $B_{\mathrm{f}}(s, j) = B_{\mathrm{f}}(s^*, j)$ for $u \leq j \leq v$. By induction, we have $s_j^* = s_j$ for all j, and so $s = s^*$. □

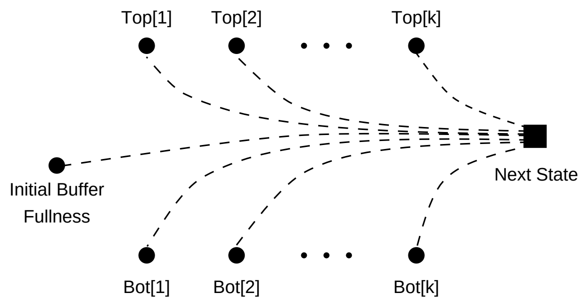


Figure 7.2: Illustration of search step in dynamic programming algorithm.

7.2 CBR Allocation Algorithm

Theorem 7.1 is a powerful result. It says that to find the optimal allocation we need only to find a legal allocation that meets the stated switching conditions. In this section, we use the technique of dynamic programming (DP) to develop an algorithm to compute a lexicographically optimal CBR allocation in polynomial time and linear space.

7.2.1 DP Algorithm

The basic idea behind dynamic programming is to decompose a given problem in terms of optimal solutions to smaller problems. All we need to do is maintain invariant the conditions stated in Theorem 7.1 for each subproblem we solve. We do this by constructing optimal bit allocations for pictures 1 to k that end up with the VBV buffer in one of two states: *full* or *empty*. These states are exactly the states where a change in Q may occur. Let Top^k be the optimal allocation for pictures 1 to k that end up with the VBV buffer *full*, if such an allocation exists. Similarly, let Bot^k be the optimal allocation for pictures 1 to k that end up with the VBV buffer *empty*. Suppose that we have computed Top^i and Bot^i for $1 \le i \le k$. To compute Top^{k+1}, we search for a legal allocation among $\left\{\emptyset, \mathrm{Top}^1, \ldots, \mathrm{Top}^k, \mathrm{Bot}^1, \ldots, \mathrm{Bot}^k\right\}$, where $\emptyset$ denotes the empty allocation, to which we can concatenate a constant-Q segment to give a legal allocation s such that the switching conditions are met and the buffer ends up full, that is, $B_{\mathrm{f}}(s, k+1) = B_{\mathrm{VBV}}$. Similarly, for Bot^{k+1} we search for a previously computed allocation that, when extended by a constant-Q segment, meets the switching conditions and results in the buffer being empty, that is, $B_{\mathrm{f}}(s, k+1) = s_{k+1}$.

The basic step in the DP algorithm is illustrated in Figure 7.2. The round nodes represent buffer states for which we have previously computed optimal allocations. Each node stores the last Q used in the optimal allocation for that state and the origin of the last constant-Q segment leading to that state. The square node represents the next state that we wish to compute. The dashed lines represent a constant-Q allocation that connects the respective nodes. To compute a solution for the square node, we need to search for an edge that connects the square node with a round node such that the switching conditions are met. For each edge, the switching conditions are checked by comparing the Q used for the edge against the last Q used in the optimal solution for the round node that the edge connects. The allocation implied by each edge is also checked for VBV compliance.

Once we have computed Top^{N-1} and Bot^{N-1}, we can compute the optimal allocation for all N pictures in a process similar to the one above for computing Top^k and Bot^k, except that the final allocation results in a final buffer state that gives the desired target number of bits B_{tgt}.

7.2.2 Correctness of DP Algorithm

When computing Top^k and Bot^k for $1 \leq k \leq N-1$, we have insured that the conditions of Theorem 7.1 are met. Additionally in the final computation, the conditions are also met. Therefore we end up with a legal allocation that meets the conditions of Theorem 7.1 and is thus optimal.

7.2.3 Constant-Q Segments

We have used the concept of a constant-Q segment extensively in the above discussion. We now formalize this concept. First, we define a family of bit-production functions $\{F_{i,j}(q)\}$ that gives the number of bits resulting from allocating a constant value of Q for pictures i to j, inclusive:

$$F_{i,j}(q) = \sum_{i \leq k \leq j} f_k(q). \tag{7.29}$$

What we are really interested in, though, is the inverse of $F_{i,j}$. We denote the inverse as $G_{i,j}$ so that $G_{i,j} = F_{i,j}^{-1}$. Then $G_{i,j}(B)$ gives the constant Q that results in B bits being produced by pictures i to j collectively. Since f_i is monotonically decreasing, so is $F_{i,j}$, and thus $G_{i,j}$ is monotonically increasing.

7.2.4 Verifying a Constant-Q Allocation

The DP algorithm for CBR bit allocation needs to verify whether a constant-Q allocation meets VBV buffer constraints. This can be done in time linear in the length of the allocation by simulating the VBV. In the DP algorithm, $O(N^2)$ verifications of constant-Q allocations are needed. If each verification requires linear time, this translates to at least cubic time complexity for the DP algorithm.

We observe that the constant-Q allocations to be verified start with the buffer either full, empty, or at its initial state; and end with the buffer either full, empty, or at its final state. We also note that for an allocation to a segment of pictures, say from i to j, with a fixed initial buffer state, say B_1, and using B_{T} bits, there is a continuous range of Q values that results in a legal allocation. When additional pictures are considered, this range of legal Q values never widens. Furthermore, the upper bound for Q is simply the minimum Q among the constant-Q allocations for pictures i to j in which the buffer is exactly full for some picture k, where $i \leq k < j$. More formally,

$$G_{i,j}(B_{\mathrm{T}}) \leq \min_{i \leq k < j} \left\{ G_{i,k}\Big(B_1 + \sum_{i \leq m \leq k} B_{\mathrm{a}}(m) - B_{\mathrm{VBV}}\Big) \right\}. \tag{7.30}$$

Similarly, the lower bound for Q is the maximum Q among the constant-Q allocations for pictures i to j in which the buffer is exactly empty for some picture k, where $i \leq k < j$. More formally,

$$G_{i,j}(B_{\mathrm{T}}) \geq \max_{i \leq k < j} \left\{ G_{i,k}\Big(B_1 + \sum_{i \leq m < k} B_{\mathrm{a}}(m)\Big) \right\}. \tag{7.31}$$

We can use these observations to perform all the VBV verifications in constant time per verification with linear-time preprocessing.

7.2.5 Time and Space Complexity

The time complexity of the DP algorithm depends upon two main factors: the time to compute a constant-Q allocation and the time to verify whether a sub-allocation is legal.

We assume that f_i and $G_{i,j}$ can be evaluated in constant time with $O(N)$ preprocessing time and space. An example is $f_i(q) = \alpha_i/q + \beta_i$, where

$$G_{i,j}(B) = \frac{\sum_{i \le k \le j} \alpha_k}{B - \sum_{i \le k \le j} \beta_k}.$$

We can precompute the prefix sums of α_i and β_i in linear time and space and then use these to compute $G_{i,j}$ in constant time. The same technique can be used for bit-production models of the form: $f_i(q) = \alpha_i/q^2 + \beta_i/q + \gamma_i$, $f_i(q) = \alpha_i/q^3 + \beta_i/q^2 + \gamma_i/q + \phi_i$, and $f_i(q) = \alpha_i/q^4 + \beta_i/q^3 + \gamma_i/q^2 + \phi_i/q + \theta_i$. Examples of other functional forms for f_i with a closed-form solution for $G_{i,j}$ can be found in [67]. Of course, we need to insure that the models are monotonically decreasing.

Since VBV verification and constant-Q calculation can be done in constant time with linear-time preprocessing, computing Top^k and Bot^k takes $O(k)$ time. Therefore, to compute an optimal allocation for a sequence of N pictures would take $\sum_{k=1}^{N} O(k) = O(N^2)$ time. If we store pointers for tracing the optimal sequence of concatenations, the algorithm requires $O(N)$ space.

7.3 Related Work

Conditions similar to the switching conditions of Theorem 7.1 have been described in [59] for optimal buffered bit allocation under a minimum sum-distortion criterion and assuming independent convex rate-distortion functions. In this work, the Lagrange multiplier method is used to find a bit allocation that is optimal within a convex-hull approximation. The optimal vector of Lagrange multipliers consists of constant-valued segments that increase (decrease, respectively) only when the decoder buffer is full (empty).

In [94], the theory of *majorization* [69] is applied to reduce the variability in transmission rate for stored video. In this setting, the problem is to determine a feasible transmission schedule by which a pre-compressed video bitstream can be transmitted over a communications channel to the decoder without underflowing or overflowing the decoder buffer. As applied to this problem, majorization results in minimizing the peak and variance in transmission rate. The authors provide an optimal smoothing algorithm that runs in time linear in the length of the video sequence.

It can be easily shown that the majorization solution to optimal smoothing of [94] is in fact equivalent to lexicographic minimization of the transmitted rates, subject to the constraint that the total number of bits transmitted is fixed. The linear running time is possible because the buffering constraints are manifested as fixed upper and lower bounds on the cumulated number of bits transmitted. In comparison, in buffer-constrained bit allocation, there is no fixed relationship between the buffering constraints and the nominal quantization (equivalently, distortion) that is the object of optimization. In a sense, the buffer-constrained bit allocation problem that we consider is "harder" than optimal smoothing.

7.4 Discussion

The analyses presented here are based upon simplifications of the actual video coding process. In particular, one of the prerequisites is an accurate bit-production model for each picture of the video sequence. Another prerequisite is proper adaptive perceptual quantization.

The bit-production model described in Section 6.3 assumes that the coding of a particular picture is independent of the coding of any other picture. As noted earlier, this independence assumption does not hold for video coders that employ a differential coding scheme such as motion compensation. In this case, the coding of a reference picture affects the coding of subsequent pictures that are coded with respect to it. Therefore, the bit-production model for picture i would depend causally not only upon the quantization scale used for picture i but also upon the quantization scales used for its reference pictures. The dependent coding problem has been addressed in the traditional distortion-minimization framework in [90]. Whether the results of this chapter can be extended to a dependent-coding framework is an open problem.

The dynamic programming solution in Section 7.2 ignores some of the structure that exists in the framework due to the monotonicity assumption and focuses solely on achieving the switching conditions of Theorem 7.1 by "brute force." For example, a "blind" search strategy is used to find the edge that connects the end state with a previously-computed optimal sub-allocation and meets the switching conditions. A feasible connector that fails the switching conditions may yield some knowledge that can be used to reduce the search space. Improving the time-complexity is a promising open problem.

Chapter 8

Lexicographic Bit Allocation under VBR Constraints

In this chapter, we analyze the buffer-constrained bit-allocation problem under variable-bit-rate VBV constraints, as described in Section 6.4.2. The analysis leads to an efficient iterative algorithm for computing a lexicographically optimal solution.

In CBR operation, the total number of bits that a CBR stream can use is dictated by the channel bit rate and the buffer size. With VBR operation, the total number of bits has no lower bound, and its upper bound is determined by the peak bit rate and the buffer size. Consequently, VBR is useful and most advantageous over CBR when the average bit rate needs to be lower than the peak bit rate. This is especially critical in storage applications, where the storage capacity, and not the transfer rate, is the limiting factor. Another important application of VBR video coding is for multiplexing multiple video bitstreams over a CBR channel [35]. In this application, statistical properties of the multiple video sequences allow more VBR bitstreams with a given peak rate $R_{\max}$ to be multiplexed onto the channel than CBR bitstreams coded at a constant rate of $R_{\max}$.

For typical VBR applications, then, the average bit rate is lower than the peak. In this case, bits enter the decoder buffer at an effective bit rate that is less than the peak during the display interval of many pictures. In interesting cases, there will be segments of pictures that will be coded with an average bit rate that is higher than the peak. This is possible because of the buffering. During the display of these pictures, the VBV buffer fills at the peak rate. Since these pictures require more bits to code than the peak rate, they are "harder" to code than the other "easier" pictures.

In order to equalize quality, the easy pictures should be coded at the same base quality. It does not pay to code any of the hard pictures at a quality higher than that of the easy pictures. The bits expended to do so could instead be better distributed to raise the quality of the easy pictures. Among the hard pictures, there are different levels of coding difficulty. Using the same intuitions from the CBR case, we can draw similar conclusions about the buffer emptying and filling behavior among the hard pictures.

In the following analysis, we show that a lexicographically optimal VBR bit allocation possesses the properties described above. In particular, the hard segments of pictures in a VBR bit allocation behave as in a CBR setting. In fact, the VBR algorithm invokes the CBR algorithm to allocate bits to segments of hard pictures.

8.1 Analysis

The following two lemmas characterize the "easy" pictures in an optimal allocation, that is, the pictures that are coded with the best quality (lowest Q).

Lemma 8.1 *Given a VBR bit-allocation problem $P = \langle N, F, B_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}}\rangle$ and an optimal allocation s^*, if $B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s_j^* > B_{\mathrm{VBV}}$ for $1 \le j \le N$, then $g_j(s_j^*) = \min_{1\le k\le N}\{g_k(s_k^*)\}$.*

Proof: Let $Q_{\min} = \min_{1\le k\le N}\{g_k(s_k^*)\}$. Let j be an index such that $B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s_j^* > B_{\mathrm{VBV}}$. Let $\Delta = B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s_j^* - B_{\mathrm{VBV}}$. Thus, $\Delta > 0$.

Suppose that $g_j(s_j^*) > Q_{\min}$. Let u be an index such that $g_u(s_u^*) = Q_{\min}$. Consider an allocation s that differs from s^* only for pictures j and u. We want to assign values to s_j and s_u that make s a legal allocation with $g_j(s_j), g_u(s_u) < g_j(s_j^*)$, from which $s^* \succ s$, a contradiction.

The idea is that we want to shift a (positive) number of bits, say δ, from picture u to picture j but still have a legal allocation. Let $s_j = s_j^* + \delta$ and $s_u = s_u^* - \delta$ with $\delta > 0$. Then $g_j(s_j) < g_j(s_j^*)$ and $\sum_{k=1}^{N} s_k = \sum_{k=1}^{N} s_k^* = B_{\mathrm{tgt}}$. We now need to show that s does not result in a VBV buffer underflow, that is, $B_{\mathrm{f}}(s, k) \ge s_k$ for $1 \le k \le N$. There are two cases to consider: $u < j$ and $u > j$.

Case 1: $u < j$. Since $s_k = s_k^*$ for $k < u$, we have $B_{\mathrm{f}}(s, k) = B_{\mathrm{f}}(s^*, k)$ for $1 \le k \le u$. Since $s_u < s_u^*$ and $s_k = s_k^*$ for $u < k < j$, we have $B_{\mathrm{f}}(s, k) \ge B_{\mathrm{f}}(s^*, k)$ for $u < k \le j$. Therefore pictures 1 to $j-1$ cannot cause any VBV buffer underflows. If we choose $0 < \delta < \Delta$, then $B_{\mathrm{f}}(s, j+1) = B_{\mathrm{VBV}}$ and picture j also cannot cause a VBV buffer underflow. Since $s_k = s_k^*$ for $k > j$ and $B_{\mathrm{f}}(s, j+1) = B_{\mathrm{f}}(s^*, j+1)$, pictures $j+1$ to N also cannot cause any VBV buffer underflows.

Case 2: $u > j$. Since $s_k = s_k^*$ for $k < j$, we have $B_{\mathrm{f}}(s, k) = B_{\mathrm{f}}(s^*, k)$ for $1 \le k \le j$. If we choose $0 < \delta < \Delta$, then $B_{\mathrm{f}}(s, j+1) = B_{\mathrm{VBV}}$ and picture j also cannot cause a VBV buffer underflow. Since $s_k = s_k^*$ for $j < k < u$, and $B_{\mathrm{f}}(s, j+1) = B_{\mathrm{f}}(s^*, j+1)$ (by a suitable choice of δ), pictures $j+1$ to $u-1$ also cannot cause any VBV buffer underflows. Since $s_u < s_u^*$, we have $B_{\mathrm{f}}(s, k) \ge B_{\mathrm{f}}(s^*, k)$ for $k : k \ge u$. Therefore pictures u to N also cannot cause any VBV buffer underflows.

Therefore s is a legal allocation with $g_j(s_j) < g_j(s_j^*)$. We need to guarantee that $g_u(s_u) < g_j(s_j^*)$. Let $\gamma = g_j(s_j^*) - g_u(s_u^*)$. Since $g_j(s_j^*) > g_u(s_u^*)$, we have $\gamma > 0$. Let $\alpha = s_u^* - f_u(g_u(s_u^*) + \gamma/2)$. Since f_u is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$
\begin{aligned}
s_u^* - \alpha &= f_u(g_u(s_u^*) + \gamma/2) \\
g_u(s_u^* - \alpha) &= g_u(s_u^*) + \gamma/2 \\
&< g_u(s_u^*) + \gamma \\
&= g_j(s_j^*).
\end{aligned}
$$

Consider the assignment $\delta = \min\{\alpha, \Delta/2\}$. There are two cases: $\alpha \le \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \le \Delta/2$, we have $\delta = \alpha$ from which $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \alpha) < g_j(s_j^*)$; since $0 < \delta < \Delta$, the allocation s is legal. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since g_u is decreasing and $\alpha > \Delta/2$, we have $g_u(s_u^* - \Delta/2) < g_u(s_u^* - \alpha)$ and thus $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \Delta/2) < g_j(s_j^*)$. Since $0 < \delta < \Delta$, the allocation s is legal.

Since s is a legal allocation that differs from s^* only for pictures u and j with $g_u(s_u) < g_j(s_j^*)$ and $g_j(s_j) < g_j(s_j^*)$, from Lemma 7.1 we have $s^* \succ s$, but s^* is not optimal, a contradiction. Therefore $g_j(s_j^*) = \min_{1\le k\le N}\{g_k(s_k^*)\}$. □

Lemma 8.2 *Given a VBR bit-allocation problem* $P = \langle N, F, B_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}}\rangle$ *and an optimal allocation* s^**, if* $B_{\mathrm{f}}(s^*, N) > s_N^*$ *then* $g_N(s_N^*) = \min_{1\le k\le N}\{g_k(s_k^*)\}$.

Proof: Let $Q_{\min} = \min_{1\le k\le N}\{g_k(s_k^*)\}$. Let j be an index such that $B_{\mathrm{f}}(s^*, j) + B_{\mathrm{a}}(j) - s_j^* > B_{\mathrm{VBV}}$. Let $\Delta = B_{\mathrm{f}}(s^*, N) - s_N^*$. Since $B_{\mathrm{f}}(s^*, N) > s_N^*$, we have $\Delta > 0$. Suppose that $g_N(s_N^*) > Q_{\min}$. Let u be an index such that $g_u(s_u^*) = Q_{\min}$. Now consider an allocation s that differs from s^* only for pictures u and N. We want to assign values to s_N and s_u that make s a legal allocation with $g_N(s_N), g_u(s_u) < g_N(s_N^*)$, from which $s^* \succ s$, thereby arriving at a contradiction. Let $s_N = s_N^* + \delta$ and $s_u = s_u^* - \delta$ with $\delta > 0$. Then $g_N(s_N) < g_N(s_N^*)$ and $\sum_{k=1}^N s_k = \sum_{k=1}^N s_k^* = B_{\mathrm{tgt}}$. We now need to show that s does not result in a VBV buffer underflow, that is, $B_{\mathrm{f}}(s, k) \ge s_k$ for $1 \le k \le N$.

Since $s_k = s_k^*$ for $k < u$, we have $B_{\mathrm{f}}(s, k) = B_{\mathrm{f}}(s^*, k)$ for $1 \le k \le u$. Since $s_u < s_u^*$ and $s_k = s_k^*$ for $u < k < N$, we have $B_{\mathrm{f}}(s, k) \ge B_{\mathrm{f}}(s^*, k)$ for $u < k \le N$. Therefore pictures 1 to $N-1$ cannot cause any VBV buffer underflows. For picture N, we have $B_{\mathrm{f}}(s, N) \ge B_{\mathrm{f}}(s^*, N) = \Delta + s_N^* = \Delta + s_N - \delta$. Therefore if we choose $\delta < \Delta$, then $B_{\mathrm{f}}(s, N) > s_N$ and picture N also cannot cause a VBV buffer underflow.

Therefore s is a legal allocation with $g_N(s_N) < g_N(s_N^*)$. We need to guarantee that $g_u(s_u) < g_N(s_N^*)$. Let $\gamma = g_N(s_N^*) - g_u(s_u^*)$. Since $g_N(s_N^*) > g_u(s_u^*)$, we have $\gamma > 0$. Let $\alpha = s_u^* - f_u(g_u(s_u^*) + \gamma/2)$. Since f_u is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$\begin{aligned}
s_u^* - \alpha &= f_u(g_u(s_u^*) + \gamma/2) \\
g_u(s_u^* - \alpha) &= g_u(s_u^*) + \gamma/2 \\
&< g_u(s_u^*) + \gamma \\
&= g_N(s_N^*).
\end{aligned}$$

Consider the assignment $\delta = \min\{\alpha, \Delta/2\}$. There are two cases: $\alpha \le \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \le \Delta/2$, we have $\delta = \alpha$ from which $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \alpha) < g_N(s_N^*)$; since $0 < \delta < \Delta$, the allocation s is legal. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since g_u is decreasing and $\alpha > \Delta/2$, we have $g_u(s_u^* - \Delta/2) < g_u(s_u^* - \alpha)$ and thus $g_u(s_u) = g_u(s_u^* - \delta) = g_u(s_u^* - \Delta/2) < g_N(s_N^*)$. Since $0 < \delta < \Delta$, the allocation s is legal.

Since s is a legal allocation that differs from s^* only for pictures u and N with $g_u(s_u) < g_N(s_N^*)$ and $g_N(s_N) < g_N(s_N^*)$, from Lemma 7.1, we have $s^* \succ s$, and s^* is not optimal, a contradiction. Therefore $g_N(s_N^*) = \min_{1\le k\le N}\{g_k(s_k^*)\}$. □

The next lemma gives a set of *switching* conditions for changes in Q that are similar to the results of Lemma 7.2.

Lemma 8.3 *Given a VBR bit-allocation problem $P = \langle N, F, B_{\text{tgt}}, B_{\text{VBV}}, B_1, B_{\text{a}} \rangle$, if s^* is an optimal allocation, the following are true:*

1. *If $g_j(s_j^*) > g_{j+1}(s_{j+1}^*)$ for $1 \le j < N$, then $B_{\text{f}}(s^*, j) = s_j^*$.*
2. *If $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$ for $1 \le j < N$, then $B_{\text{f}}(s^*, j+1) = B_{\text{VBV}}$ and $B_{\text{f}}(s^*, j+1) + B_{\text{a}}(j+1) - s_{j+1}^* \le B_{\text{VBV}}$.*

Proof:
Case 1. The proof is identical to the proof of Case 1 of Lemma 7.2, except that condition (6.5) now holds instead of (6.8).
Case 2. Suppose that Case 2 is false. Then either $B_{\text{f}}(s^*, j+1) < B_{\text{VBV}}$ or $B_{\text{f}}(s^*, j+1) + B_{\text{a}}(j+1) - s_{j+1}^* > B_{\text{VBV}}$. Suppose that $B_{\text{f}}(s^*, j+1) + B_{\text{a}}(j+1) - s_{j+1}^* > B_{\text{VBV}}$. Then by Lemma 8.1, $g_{j+1}(s_{j+1}^*) \le g_j(s_j^*)$, a contradiction. Therefore $B_{\text{f}}(s^*, j+1) + B_{\text{a}}(j+1) - s_{j+1}^* \le B_{\text{VBV}}$.

Suppose that $B_{\text{f}}(s^*, j+1) < B_{\text{VBV}}$. Let $\Delta = B_{\text{VBV}} - B_{\text{f}}(s^*, j+1)$. Then $\Delta > 0$. Consider an allocation s that differs from s^* only for pictures j and $j+1$. We want to assign values to s_j and s_{j+1} that make s a legal allocation with $g_j(s_j), g_{j+1}(s_{j+1}) < g_{j+1}(s_{j+1}^*)$, from which $s^* \succ s$, thereby arriving at a contradiction.

Let $\gamma = g_{j+1}(s_{j+1}^*) - g_j(s_j^*)$ and $\alpha = s_j^* - f_j(g_j(s_j^*) + \gamma/2)$. Since $g_j(s_j^*) < g_{j+1}(s_{j+1}^*)$, we have $\gamma > 0$. Since f_j is decreasing and $\gamma > 0$, we have $\alpha > 0$ and

$$\begin{aligned} s_j^* - \alpha &= f_j(g_j(s_j^*) + \gamma/2) \\ g_j(s_j^* - \alpha) &= g_j(s_j^*) + \gamma/2 \\ &< g_j(s_j^*) + \gamma \\ &= g_{j+1}(s_{j+1}^*). \end{aligned}$$

Consider the assignments $s_j = s_j^* - \delta$ and $s_{j+1} = s_{j+1}^* + \delta$, where $\delta = \min\{\alpha, \Delta/2\}$. By this assignment, we have $g_{j+1}(s_{j+1}) < g_{j+1}(s_{j+1}^*)$. We now show that $g_j(s_j) < g_{j+1}(s_{j+1}^*)$. There are two cases: $\alpha \le \Delta/2$ and $\alpha > \Delta/2$. If $\alpha \le \Delta/2$, we have $\delta = \alpha$ from which $g_j(s_j) = g_j(s_j^* - \delta) = g_j(s_j^* - \alpha) < g_{j+1}(s_{j+1}^*)$. If $\alpha > \Delta/2$, we have $\delta = \Delta/2$. Since g_j is decreasing and $\alpha > \Delta/2$, we have $g_j(s_j^* - \Delta/2) < g_j(s_j^* - \alpha)$ and thus $g_j(s_j) = g_j(s_j^* - \delta) = g_j(s_j^* - \Delta/2) < g_{j+1}(s_{j+1}^*)$. In either case, we have $g_j(s_j) < g_{j+1}(s_{j+1}^*)$.

We now need to show that allocation s as defined above is a legal allocation. Since $s_k = s_k^*$ for $k < j$, we have $B_{\text{f}}(s, k) = B_{\text{f}}(s^*, k)$ for $1 \le k \le j$. Therefore there are no VBV buffer violations in pictures 1 to $j-1$. Since $s_j < s_j^*$, we have $B_{\text{f}}(s, j+1) > B_{\text{f}}(s^*, j+1)$. Therefore picture j cannot cause a VBV buffer underflow.

Now we need to show that pictures $j+1$ to N also cannot cause a VBV buffer underflow. Since $B_{\text{f}}(s^*, j+1) < B_{\text{VBV}}$, we have $B_{\text{f}}(s^*, j+1) = B_{\text{f}}(s^*, j) + B_{\text{a}}(j) - s_j^*$ and

$$\begin{aligned} B_{\text{f}}(s, j) + B_{\text{a}}(j) - s_j &= B_{\text{f}}(s^*, j) + B_{\text{a}}(j) - (s_j^* - \delta) \\ &= B_{\text{f}}(s^*, j+1) + \delta \\ &< B_{\text{f}}(s^*, j+1) + \Delta \\ &= B_{\text{VBV}}. \end{aligned}$$

Thus $B_{\rm f}(s, j+1) = B_{\rm f}(s, j) + B_{\rm a}(j) - s_j$. We have already shown that $B_{\rm f}(s^*, j+1) + B_{\rm a}(j+1) - s^*_{j+1} \le B_{\rm VBV}$. Therefore $B_{\rm f}(s^*, j+2) = B_{\rm f}(s^*, j+1) + B_{\rm a}(j+1) - s^*_{j+1}$. Now,

$$\begin{aligned} B_{\rm f}(s, j+1) + B_{\rm a}(j+1) - s_{j+1} &= B_{\rm f}(s, j) + B_{\rm a}(j) - s_j + B_{\rm a}(j+1) - s_{j+1} \\ &= B_{\rm f}(s^*, j+1) + \delta + B_{\rm a}(j+1) - (s^*_{j+1} + \delta) \\ &= B_{\rm f}(s^*, j+1) + B_{\rm a}(j+1) - s^*_{j+1} \\ &= B_{\rm f}(s^*, j+2) \\ &\le B_{\rm VBV}. \end{aligned}$$

Therefore $B_{\rm f}(s, j+2) = B_{\rm f}(s^*, j+2)$. Since $s_k = s^*_k$ for $k > j+1$, we have $B_{\rm f}(s, k) = B_{\rm f}(s^*, k)$ for $k > j+1$. Therefore pictures $j+1$ to N cannot cause a VBV buffer underflow and s is a legal allocation.

Since s is a legal allocation that differs from s^* only for pictures j and $j+1$ and we have $g_j(s_j) < g_{j+1}(s^*_{j+1})$ and $g_{j+1}(s_{j+1}) < g_{j+1}(s^*_{j+1})$, from Lemma 7.1 we have $s^* \succ s$, but s^* is not optimal, a contradiction. $\square$

The following theorem is the main result of this section. It shows that the minimum-Q and switching conditions in the previous lemmas are also sufficient for optimality.

Theorem 8.1 *Given a VBR bit-allocation problem $P = \langle N, F, B_{\rm tgt}, B_{\rm VBV}, B_1, B_{\rm a}\rangle$, a legal allocation s is optimal if and only if the following conditions hold. Also, the optimal allocation is unique.*

1. *If $B_{\rm f}(s, j) + B_{\rm a}(j) - s_j > B_{\rm VBV}$ for $1 \le j \le N$, then $g_j(s_j) = \min_{1\le k\le N}\{g_k(s_k)\}$.*
2. *If $B_{\rm f}(s^*, N) > s^*_N$ then $g_N(s^*_N) = \min_{1\le k\le N}\{g_k(s^*_k)\}$.*
3. *If $g_j(s_j) > g_{j+1}(s_{j+1})$ for $1 \le j < N$, then $B_{\rm f}(s, j) = s_j$.*
4. *If $g_j(s_j) < g_{j+1}(s_{j+1})$ for $1 \le j < N$, then $B_{\rm f}(s, j+1) = B_{\rm VBV}$ and $B_{\rm f}(s, j+1) + B_{\rm a}(j+1) - s_{j+1} \le B_{\rm VBV}$.*

Proof: Lemmas 8.1, 8.2, and 8.3 establish these as necessary conditions. Now we need to show that these conditions are also sufficient for optimality and imply uniqueness.

The proof for sufficiency and uniqueness is similar to that of Theorem 7.1 except for segments with the minimum Q. Here we consider only segments that use the minimum Q.

Let s^* be an optimal allocation, $Q_{\min} = \min_{1\le j\le N}\{g_j(s_j)\}$, and $J_{\min} = \{j : g_j(s_j) = Q_{\min}\}$. By condition 2, if $g_N(s_N) > Q_{\min}$ then it must be that $B_{\rm f}(s, N) = s_N$, or equivalently, $B_{\rm f}(s, N+1) = B_{\rm a}(N)$. Therefore $B_{\rm f}(s, N)$ is known if picture N does not use the minimum Q, and we can use arguments of Theorem 7.1. Following the steps of Theorem 7.1, we can show that $s^*_j = s_j$ for $j : g_j(s_j) > Q_{\min}$.

Since s^* is optimal, we have $g_j(s^*_j) \le g_j(s_j)$ for $j \in J_{\min}$. Therefore

$$s^*_j \ge s_j \text{ for } j \in J_{\min}. \tag{8.1}$$

Since the total number of bits allocated is the same for s and s^*, we have the number of bits to be allocated to pictures in J must also be the same. That is,

$$\sum_{j \in J_{\min}} s_j^* = \sum_{j \in J_{\min}} s_j. \tag{8.2}$$

But (8.1) and (8.2) both hold if and only if $s_j^* = s_j$ for $j \in J_{\min}$. Therefore $s = s^*$. $\square$

Although Theorem 8.1 is an important result, it does not show us how to compute the minimum Q with which to code the "easy" pictures. The following lemmas and theorem show that, if we relax the bit budget constraint, we can find the minimum Q, and therefore the optimal allocation, to meet the bit budget by an iterative process. Furthermore, the iterative process is guaranteed to converge to the optimal allocation in a finite number of steps.

Lemma 8.4 *Given two VBR bit-allocation problems* $P^{(1)} = \left\langle N, F, B_{\mathrm{tgt}}^{(1)}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *and* $P^{(2)} = \left\langle N, F, B_{\mathrm{tgt}}^{(2)}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *that have optimal allocations* $s^{(1)}$ *and* $s^{(2)}$*, respectively, with* $B_{\mathrm{tgt}}^{(1)} < B_{\mathrm{tgt}}^{(2)}$*, then* $s^{(1)} \succ s^{(2)}$.

Proof: Let $J_{\mathrm{over}} = \left\{ j : B_{\mathrm{f}}(s^{(1)}, j) + B_{\mathrm{a}}(j) - s_j^{(1)} > B_{\mathrm{VBV}} \right\}$. Then J_{over} contains exactly the pictures that result in virtual overflows, as defined in Section 6.4.2. If we start with allocation $s^{(1)}$, it is clear that we can use more bits for the pictures in J_{over} without changing the buffer fullness $B_{\mathrm{f}}(s^{(1)}, n)$. Let $B_{\mathrm{over}} = \sum_{j \in J_{\mathrm{over}}} \left(B_{\mathrm{f}}(s^{(1)}, j) + B_{\mathrm{a}}(j) - s_j^{(1)} - B_{\mathrm{VBV}} \right)$. Then B_{over} is the maximum number of bits we can add to the pictures in J_{over} without changing the buffer fullness. Let $\Delta = B_{\mathrm{tgt}}^{(2)} - B_{\mathrm{tgt}}^{(1)}$. There are two cases to consider: $\Delta \le B_{\mathrm{over}}$ and $\Delta > B_{\mathrm{over}}$.

Case 1: $\Delta \le B_{\mathrm{over}}$. Consider an allocation s for problem $P^{(2)}$ constructed as follows. Let $s_j = s_j^{(1)}$ for $j \notin J_{\mathrm{over}}$. We then distribute Δ bits to the pictures in J_{over} without changing the buffer fullness. Then $s_j \ge s_j^{(1)}$ which implies that $g_j(s_j) \le g_j(s_j^{(1)})$. Since $\Delta > 0$, we also have $s_j > s_j^{(1)}$ for some $j \in J_{\mathrm{over}}$. Since $B_{\mathrm{f}}(s, j) = B_{\mathrm{f}}(s^{(1)}, j)$ for all j, s does not cause any buffer underflows. Since we used $B_{\mathrm{tgt}}^{(1)} + \Delta = B_{\mathrm{tgt}}^{(2)}$ bits in s, s is a legal allocation for $P^{(2)}$.

Case 2: $\Delta > B_{\mathrm{over}}$. Consider an allocation s for problem $P^{(2)}$ constructed as follows. Let $s_j = s_j^{(1)}$ for $j \notin J_{\mathrm{over}} \cup \{N\}$. We then distribute B_{over} bits to pictures in J_{over}. We do this with the assignments: $s_j = s^{(1)} + (B_{\mathrm{f}}(s^{(1)}, j) + B_{\mathrm{a}}(j) - s_j^{(1)} - B_{\mathrm{VBV}})$ for $j \in J_{over}$. Finally, we distribute the remaining $\Delta - B_{over}$ bits to picture N with $s_N = s_N^{(1)} + \Delta - B_{over}$.

We have shown how to create a legal allocation s for $P^{(2)}$ starting with $s^{(1)}$. When we add more bits to $s^{(1)}$ to form s, we strictly decrease Q for the pictures that we add bits to and never increase Q anywhere. Therefore $s^{(1)} \succ s$. Since $s^{(2)}$ is the optimal allocation for $P^{(2)}$, we have $s \succeq s^{(2)}$. Therefore $s^{(1)} \succ s^{(2)}$. $\square$

Lemma 8.5 *Given two VBR bit-allocation problems* $P^{(1)} = \left\langle N, F, B_{\mathrm{tgt}}^{(1)}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *and* $P^{(2)} = \left\langle N, F, B_{\mathrm{tgt}}^{(2)}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *that have optimal allocations* $s^{(1)}$ *and* $s^{(2)}$*, respectively, with* $B_{\mathrm{tgt}}^{(1)} < B_{\mathrm{tgt}}^{(2)}$*, then* $s_j^{(1)} = s_j^{(2)}$ *for* j *such that* $g_j(s_j^{(1)}) > \min_{1 \le k \le N}\{g_k(s_k^{(1)})\}$.

Proof: We provide an inductive proof similar to that used to prove Theorem 7.1. First we assume that $s^{(1)}$ is not a constant-Q allocation, for if it were, the lemma would hold vacuously.

Let $Q_k^{(1)}$ be the kth largest value of Q assigned by allocation $s^{(1)}$. Let $Q_{\min}^{(1)}$ be the minimum value of Q.

Inductive Hypothesis: For all segments of pictures u to v with $u = 1$ or $g_{u-1}(s_{u-1}) \neq g_u(s_u)$, $v = N$ or $g_v(s_v) \neq g_{v+1}(s_{v+1})$, and $g_j(s_j) = Q_k^{(1)} > Q_{\min}^{(1)}$, we have $s_j^{(1)} = s_j^{(2)}$ and $B_{\mathrm{f}}(s^{(1)}, j) = B_{\mathrm{f}}(s^{(2)}, j)$ for $u \leq j \leq v$.

We first prove the base case of $k = 1$. Consider the segments of consecutive pictures that are assigned quantization $Q_1^{(1)}$ by allocation $s^{(1)}$. Let u be the index of the start of such a segment. We consider two cases: $u = 1$ and $u > 1$. If $u = 1$, then $B_{\mathrm{f}}(s^{(1)}, u) = B_{\mathrm{f}}(s^{(2)}, u) = B_1$. If $u > 1$, then since u is the index of the start of the segment, we have $g_{u-1}(s_{u-1}^{(1)}) < g_u(s_u^{(1)})$, which implies that $B_{\mathrm{f}}(s^{(1)}, u) = B_{\mathrm{VBV}}$ by Lemma 8.3; since $s^{(2)}$ is a legal allocation, we have $B_{\mathrm{f}}(s^{(2)}, u) \leq B_{\mathrm{VBV}}$. In either case we have

$$B_{\mathrm{f}}(s^{(1)}, u) \geq B_{\mathrm{f}}(s^{(2)}, u) \tag{8.3}$$

Let v be the index of the end of the segment. We consider two cases: $v = N$ and $v < N$. If $v = N$, then by the contrapositive of Lemma 8.2, $B_{\mathrm{f}}(s^{(1)}, v) = s_v^{(1)}$. (Here we use the condition that $Q_1^{(1)} > Q_{\min}^{(1)}$.) If $v < N$, then since v is the index of the end of the segment, we have $g_v(s_v^{(1)}) > g_{v+1}(s_{v+1}^{(1)})$, which implies that $B_{\mathrm{f}}(s^{(1)}, v) = s_v^{(1)}$ by Lemma 8.3. In either case we have

$$B_{\mathrm{f}}(s^{(1)}, v) = s_v^{(1)}. \tag{8.4}$$

From Lemma 8.4, we have $s^{(1)} \succ s^{(2)}$. Therefore $g_j(s_j^{(2)}) \leq Q_1^{(1)}$ for all j and thus

$$s_j^{(1)} \leq s_j^{(2)} \text{ for } u \leq j \leq v. \tag{8.5}$$

From (8.3) and (8.5) we have

$$B_{\mathrm{f}}(s^{(1)}, j) \geq B_{\mathrm{f}}(s^{(2)}, j) \quad \text{for } u \leq j \leq v. \tag{8.6}$$

Since $s^{(2)}$ is a legal allocation, we have

$$B_{\mathrm{f}}(s^{(2)}, v) \geq s_v^{(2)} \geq s_v^{(1)} = B_{\mathrm{f}}(s^{(1)}, v). \tag{8.7}$$

Combining (8.6) and (8.7), we have $B_{\mathrm{f}}(s^{(1)}, v) = B_{\mathrm{f}}(s^{(2)}, v)$ and $s_v^{(1)} = s_v^{(2)}$. Therefore $B_{\mathrm{f}}(s^{(1)}, v+1) = B_{\mathrm{f}}(s^{(2)}, v+1)$. Since $Q_1^{(1)} > Q_{\min}^{(1)}$, by the contrapositive of Lemma 8.2, we see that the buffer fullness for pictures u to v is updated as with CBR operation. Therefore we can use the results of Lemma 7.3, which implies that $B_{\mathrm{f}}(s_j^{(1)}) = B_{\mathrm{f}}(s^{(2)}, j)$ and $s_j^{(1)} = s_j^{(2)}$ for $u \leq j \leq v$.

Let us assume that the inductive hypothesis is true for $1 \leq k < m$. We need to show that it is also true for $k = m$ where $Q_m^{(1)} > Q_{\min}^{(1)}$. Consider a segment of consecutive pictures that are assigned quantization $Q_m^{(1)}$. Let u be the index of the start of the segment and v the index of the end of the segment. We consider all cases for the segment boundaries. For the left segment boundary we consider three cases: $u = 1$, $g_{u-1}(s_{u-1}^{(1)}) > g_u(s_u^{(1)})$, and $g_{u-1}(s_{u-1}^{(1)}) < g_u(s_u^{(1)})$. If $u = 1$, then we have $B_{\mathrm{f}}(s^{(1)}, u) = B_{\mathrm{f}}(s^{(2)}, u) = B_1$. If $g_{u-1}(s_{u-1}^{(1)}) > g_u(s_u^{(1)})$, then from the inductive hypothesis, we have $B_{\mathrm{f}}(s^{(1)}, u-1) = B_{\mathrm{f}}(s^{(2)}, u-1)$ and $s^{(1)}_{u-1} = s^{(2)}_{u-1}$; therefore $B_{\mathrm{f}}(s^{(1)}, u) = B_{\mathrm{f}}(s^{(2)}, u)$. If $g_{u-1}(s^{(1)}_{u-1}) < g_u(s^{(1)}_u)$, then from Lemma 8.3, we have $B_{\mathrm{f}}(s^{(1)}, u) = B_{\mathrm{VBV}}$; since $s^{(2)}$ is a legal allocation, we have $B_{\mathrm{f}}(s^{(2)}, u) \le B_{\mathrm{VBV}} = B_{\mathrm{f}}(s^{(1)}, u)$. For all three cases we have

$$B_{\mathrm{f}}(s^{(2)}, u) \le B_{\mathrm{f}}(s^{(1)}, u). \tag{8.8}$$

For the right segment boundary we consider three cases: $v = N$, $g_v(s^{(1)}_v) > g_{v+1}(s^{(1)}_{v+1})$, and $g_v(s^{(1)}_v) < g_{v+1}(s^{(1)}_{v+1})$. If $v = N$, then by the contrapositive of Lemma 8.2, $B_{\mathrm{f}}(s^{(1)}, v) = s^{(1)}_v$. (We use the condition that $Q^{(1)}_m \neq Q^{(1)}_{\min}$.) If $g_v(s^{(1)}_v) > g_{v+1}(s^{(1)}_{v+1})$, then by Lemma 8.3, $B_{\mathrm{f}}(s^{(1)}, v) = s^{(1)}_v$. If $g_v(s^{(1)}_v) < g_{v+1}(s^{(1)}_{v+1})$, then from the inductive hypothesis, we have $B_{\mathrm{f}}(s^{(1)}, v+1) = B_{\mathrm{f}}(s^{(2)}, v+1)$. For the first two cases, we have

$$B_{\mathrm{f}}(s^{(1)}, v) = s^{(1)}_v \tag{8.9}$$

From Lemma 8.4, we have $s^{(1)} \succ s^{(2)}$. Therefore $g_j(s^{(2)}_j) \le Q^{(1)}_m$ for $u \le j \le v$ and thus

$$s^{(1)}_j \le s^{(2)}_j \text{ for } u \le j \le v. \tag{8.10}$$

From (8.8) and (8.10) we have

$$B_{\mathrm{f}}(s^{(1)}, j) \ge B_{\mathrm{f}}(s^{(2)}, j) \quad \text{for } u \le j \le v. \tag{8.11}$$

Since $s^{(2)}$ is a legal allocation, we have

$$B_{\mathrm{f}}(s^{(2)}, v) \ge s^{(2)}_v \ge s^{(1)}_v = B_{\mathrm{f}}(s^{(1)}, v). \tag{8.12}$$

Combining (8.11) and (8.12), we have $B_{\mathrm{f}}(s^{(1)}, v) = B_{\mathrm{f}}(s^{(2)}, v)$ and $s^{(1)}_v = s^{(2)}_v$. Therefore $B_{\mathrm{f}}(s^{(1)}, v+1) = B_{\mathrm{f}}(s^{(2)}, v+1)$. So for all three cases for v, we have $B_{\mathrm{f}}(s^{(1)}, v+1) = B_{\mathrm{f}}(s^{(2)}, v+1)$.

Since $Q^{(1)}_n > Q^{(1)}_{\min}$, by the contrapositive of Lemma 8.2, we see that the buffer fullness for pictures u to v is updated the same as with CBR operation. Therefore we can use the results of Lemma 7.3, which implies that $B_{\mathrm{f}}(s^{(1)}_j) = B_{\mathrm{f}}(s^{(2)}, j)$ and $s^{(1)}_j = s^{(2)}_j$ for $u \le j \le v$.

By induction, we have $s^{(1)}_j = s^{(2)}_j$ for all j such that $g_j(s^{(1)}, j) > Q^{(1)}_{\min}$. □

Lemma 8.6 *Given two VBR bit-allocation problems* $P^{(1)} = \left\langle N, F, B^{(1)}_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *and* $P^{(2)} = \left\langle N, F, B^{(2)}_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *that have optimal allocations* $s^{(1)}$ *and* $s^{(2)}$*, respectively, with* $B^{(1)}_{\mathrm{tgt}} < B^{(2)}_{\mathrm{tgt}}$*, then* $\min_{1\le j\le N}\{g_j(s^{(1)}_j)\} > \min_{1\le j\le N}\{g_j(s^{(2)}_j)\}$.

Proof: Let $Q^{(1)}_{\min} = \min_{1\le j\le N}\{g_j(s^{(1)}_j)\}$ and $Q^{(2)}_{\min} = \min_{1\le j\le N}\{g_j(s^{(2)}_j)\}$. From Lemma 8.4 we have $s^{(1)} \succ s^{(2)}$. From Lemma 8.5, the only pictures that can be assigned a different Q by $s^{(1)}$ and $s^{(2)}$ are those that are assigned quantization $Q^{(1)}_{\min}$ by $s^{(1)}$. But $s^{(1)} \succ s^{(2)}$ which implies that $s^{(2)}$ must assign to some picture a quantization lower than $Q^{(1)}_{\min}$. Therefore $Q^{(1)}_{\min} > Q^{(2)}_{\min}$. □

We summarize Lemmas 8.4, 8.5, and 8.6 with the following theorem.

Theorem 8.2 *Given two VBR bit-allocation problems* $P^{(1)} = \left\langle N, F, B^{(1)}_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *and* $P^{(2)} = \left\langle N, F, B^{(2)}_{\mathrm{tgt}}, B_{\mathrm{VBV}}, B_1, B_{\mathrm{a}} \right\rangle$ *that have optimal allocations* $s^{(1)}$ *and* $s^{(2)}$*, respectively, with* $B^{(1)}_{\mathrm{tgt}} < B^{(2)}_{\mathrm{tgt}}$*, then*

1. $s^{(1)} \succ s^{(2)}$,

2. $s_j^{(1)} = s_j^{(2)}$ *for* j *such that* $g_j(s_j^{(1)}) > \min_{1 \le k \le N}\{g_k(s_k^{(1)})\}$, *and*

3. $\min_{1 \le j \le N}\{g_j(s_j^{(1)})\} > \min_{1 \le j \le N}\{g_j(s_j^{(2)})\}$.

8.2 VBR Allocation Algorithm

Theorems 8.1 and 8.2 give us a way to find the optimal allocation for a given VBR allocation problem. If we know the minimum Q that the optimal allocation uses, then it would be straightforward to find the optimal allocation. However, in general we do not know what that minimum Q would be. Theorem 8.2 gives us an iterative way to find the minimum Q.

8.2.1 VBR Algorithm

Here we sketch an iterative algorithm for computing a VBR allocation.

1. Mark all pictures as *easy*. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}}$.

2. Allocate B_{easy} bits to easy pictures using a constant Q. Let Q_{min} be the value of Q used.

3. Simulate operation of VBV to identify *hard* and *easy* segments of pictures. A hard segment contains pictures that lead to a buffer underflow and consists of pictures that follow the most recent virtual overflow up to and including the picture that caused the overflow. After identifying a hard segment, reset the buffer fullness to empty and continue the simulation.

4. Allocate bits to each newly identified hard segment according to the CBR algorithm, with a bit budget such that the underflow is just prevented. By preventing underflow in the hard segments, we are left with extra unallocated bits.

5. Let B_{hard} be the total number of bits allocated to hard pictures. Let $B_{\text{easy}} \leftarrow B_{\text{tgt}} - B_{\text{hard}}$.

6. If a new hard segment has been identified in Step 3, goto Step 2.

8.2.2 Correctness of VBR Algorithm

Here we prove that the VBR algorithm computes a lexicographically optimal allocation. We do this by showing that the algorithm computes an allocation that satisfies the switching conditions of Theorem 8.1.

First, we make several observations about the VBR algorithm.

1. Pictures marked "easy" are assigned the same value of Q,

2. "Hard" pictures are marked in segments that start either at the beginning of the video sequence or with the buffer full and that end with the buffer empty.

3. Segments of hard pictures are allocated using the CBR algorithm.

The correctness of the CBR algorithm insures that within hard segments conditions 3 and 4 of Theorem 8.1 hold. In order to show that the other conditions also hold, we first need to show that the CBR algorithm does not assign a Q lower than the $Q_{\min}$ computed in Step 2.

Lemma 8.7 *Let s be an allocation computed by the VBR algorithm. Let i and j denote the indices of the beginning and end, respectively, of a hard segment as identified in Step 3. Then*

$$\min_{i \le k \le j} \{g_k(s_k)\} \ge Q_{\min}.$$

Proof: Let s' be an allocation that is the same as s except for pictures i to j, where s' uses $Q_{\min}$. Thus, in a VBV simulation using s' for pictures i to j, s' does not cause a virtual overflow and underflows only at picture j. Let u and v mark the beginning and end, respectively, of a segment with the minimum Q in the CBR allocation for pictures i to j. We consider two cases for u: $u = i$ and $u > i$. If $u = i$, then we have $B_{\mathrm{f}}(s, u) = B_{\mathrm{f}}(s', u)$ since $s_k = s'_k$ for $k < i$. If $u > i$, then since u marks the beginning of a segment with minimum Q in the CBR allocation for pictures i to j, from Theorem 7.1, $B_{\mathrm{f}}(s, u-1) = s_{u-1}$. This implies that $B_{\mathrm{f}}(s, u) = B_{\mathrm{a}}(u-1)$. Since s' does not cause an underflow for picture $u-1$, $B_{\mathrm{f}}(s', u-1) \ge s'_{u-1}$, which implies that $B_{\mathrm{f}}(s', u) \ge B_{\mathrm{a}}(u-1)$. In either case, we have

$$B_{\mathrm{f}}(s', u) \ge B_{\mathrm{f}}(s, u). \tag{8.13}$$

We consider two cases for v: $v = j$ and $v < j$. If $v = j$, then $B_{\mathrm{f}}(s', v) < s'_v$ since an underflow occurs at picture j. Thus $B_{\mathrm{f}}(s', v{+}1) < B_{\mathrm{a}}(v)$. But since s is a legal allocation, $B_{\mathrm{f}}(s, v{+}1) \ge B_{\mathrm{a}}(v)$. If $v < j$, then since v marks the end of a segment with minimum Q in the CBR allocation for pictures i to j, from Theorem 7.1, $B_{\mathrm{f}}(s, v+1) = B_{\mathrm{VBV}}$. Since s' does not cause virtual overflow, $B_{\mathrm{f}}(s', v+1) \le B_{\mathrm{VBV}}$. In either case,

$$B_{\mathrm{f}}(s', v+1) \le B_{\mathrm{f}}(s, v+1). \tag{8.14}$$

Expanding for $B_{\mathrm{f}}(s, u)$ and $B_{\mathrm{f}}(s, v+1)$ we have

$$\begin{aligned} B_{\mathrm{f}}(s, u) &= B_1 + \sum_{k=1}^{u-1} B_{\mathrm{a}}(k) - \sum_{k=1}^{u-1} s_k, & (8.15)\\ B_{\mathrm{f}}(s, v+1) &= B_1 + \sum_{k=1}^{v} B_{\mathrm{a}}(k) - \sum_{k=1}^{v} s_k. & (8.16) \end{aligned}$$

Subtracting (8.16) from (8.15), canceling like terms, and rearranging, we have

$$\sum_{k=u}^{v} s_k = \sum_{k=u}^{v} B_{\mathrm{a}}(k) + B_{\mathrm{f}}(s, u) - B_{\mathrm{f}}(s, v+1). \tag{8.17}$$

The same manipulations with $B_{\mathrm{f}}(s', u)$ and $B_{\mathrm{f}}(s', v+1)$ yield

$$\sum_{k=u}^{v} s'_k = \sum_{k=u}^{v} B_{\mathrm{a}}(k) + B_{\mathrm{f}}(s', u) - B_{\mathrm{f}}(s', v+1). \tag{8.18}$$

Combining (8.13), (8.14), (8.17), and (8.18) we have

$$\sum_{k=u}^{v} s_k \leq \sum_{k=u}^{v} s'_k. \tag{8.19}$$

Pictures u to v use a constant Q in both allocations s and s', where s uses $Q = \min_{i \leq k \leq j}\{g_k(s_k)\}$ and s' uses $Q_{\min}$. Therefore we have

$$F_{u,v}\Big(\min_{i \leq k \leq j}\{g_k(s_k)\}\Big) \leq F_{u,v}(Q_{\min}). \tag{8.20}$$

Since $F_{u,v}$ is a monotonically decreasing function (see Section 7.2.3), we have

$$\min_{i \leq k \leq j}\{g_k(s_k)\} \geq Q_{\min}.$$

□

From Lemma 8.7, we can conclude that after each iteration of the VBR algorithm, $Q_{\min}$ is indeed the minimum Q. Since hard segments do not include pictures that cause a virtual overflow and does not include the last picture if it does not cause a buffer underflow, conditions 1 and 2 of Theorem 8.1 also hold.

We are now ready to state the main result of this section.

Theorem 8.3 (Correctness of VBR Algorithm) *Each pass through the VBR algorithm results in an allocation that is lexicographically optimal for the number of bits actually allocated.*

8.2.3 Time and Space Complexity

We note that the loop in the VBR algorithm terminates when no more hard segments are identified. This implies that the algorithm terminates after at most N iterations, where N is the number of pictures.

Assuming that $G_{i,j}$ can be evaluated in constant time, we have shown in Section 7.2.5 that the CBR algorithm operates in $O(N^2)$ time and uses $O(N)$ space. We now show that the VBR algorithm also executes in $O(N^2)$ time and uses $O(N)$ space.

Not counting the executions of the CBR algorithm, each iteration of the VBR algorithm takes $O(N)$ time and space. Since at most $O(N)$ iterations are performed, the time complexity excluding the executions of the CBR algorithm is $O(N^2)$.

We can defer actually invoking the CBR algorithm in Step 4 of the VBR algorithm until the end. This would avoid invoking the CBR algorithm more than once for each hard picture. Let M be the number of hard segments found by the VBR algorithm and L_i be the size of the ith segment. The time consumed by execution of the CBR algorithm can be expressed as

$$T_{\text{CBR}}(N) = \sum_{i=1}^{M} O(L_i^2) = O\Big(\sum_{i=1}^{M} L_i^2\Big). \tag{8.21}$$

Since $\sum_{i=1}^{M} L_i \leq N$, we have

$$\sum_{i=1}^{M} L_i^2 \leq \Big(\sum_{i=1}^{M} L_i\Big)^2 \leq N^2.$$

Therefore the time complexity of the VBR algorithm is $O(N^2)$, the same as that for the CBR algorithm. For cases where there are relatively few hard segments, computing an optimal VBR allocation will likely be faster in practice than computing a CBR allocation. Furthermore, Theorem 8.3 guarantees that we can halt the VBR algorithm after any number of iterations and have an optimal allocation. The decision to continue depends upon whether the achieved bit consumption is acceptable. With each iteration the number of bits allocated increases.

8.3 Discussion

The above complexity analysis is performed in the context of off-line global optimization. The vast majority of CBR video coders in operation today work in real-time mode without the luxury of lookahead processing. Since VBR coders can potentially give better quality for the same bit budget, they are targeted for quality-sensitive applications (such as encoding a Hollywood movie) where expensive off-line processing is a viable option. However, the above analysis does allow for "one-pass" VBR encoding. By substituting a real-time CBR algorithm for the optimal one invoked by the VBR algorithm, we can construct a one-pass real-time VBR encoder. Though necessarily sub-optimal, the resulting coder would have complexity comparable to existing CBR coders.

Chapter 9

Implementation of Lexicographic Bit Allocation

In this chapter, we describe an implementation of rate control using the lexicographically optimal bit allocation algorithms presented in Chapters 7 and 8 within a publicly available software MPEG-2 encoder [74]. With this implementation, we aim to: 1) verify the effectiveness of lexicographic optimality, 2) assess the practical implications of the assumptions made in the framework, namely independent coding and continuous variables, 3) explore various bit-production models, and 4) develop robust techniques for recovering from errors with the approximate models.

9.1 Perceptual Quantization

For perceptual quantization, we use the TM5 adaptive quantization scheme (described in Section 3.7.5), where the nominal quantization scale is modulated by an activity factor that is computed from the spatial activity of the luminance blocks within a macroblock. In TM5, the actual quantization scale used for coding a particular macroblock is determined from an initially computed (global) reference quantization scale, a (local) feedback factor that is dependent of the state of a virtual encoding buffer, and the activity factor. For modeling purposes, we define the nominal quantization for a picture as the average of the product of the reference quantization scale and the buffer-feedback factor over all coded macroblocks.

9.2 Bit-Production Modeling

The framework in Chapter 6 presumes the existence of an exact continuous bit-production model for each picture. In practice, the rate-distortion function of a complex encoding system, such as MPEG, cannot be determined exactly for non-trivial classes of input. Therefore, approximate models are used in practice.

As the complexity analyses in Sections 7.2.5 and 8.2.3 show, the running time for the optimal bit allocation algorithms depends on the time to evaluate $G_{i,j}$, the function that is used to compute

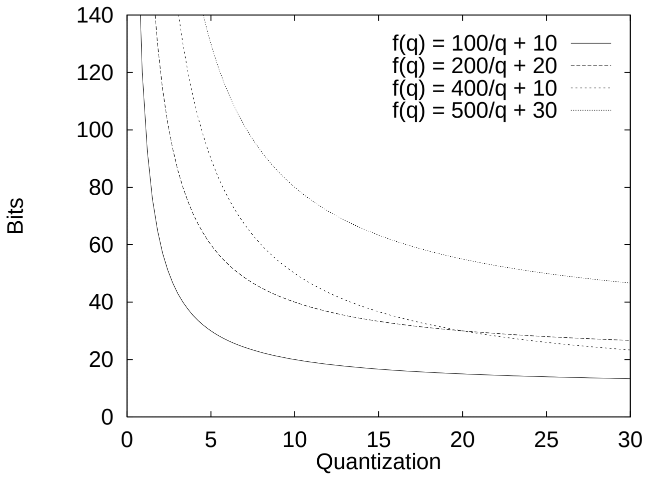


Figure 9.1: Several instances of a simple "hyperbolic" bit-production model.

a constant-Q sub-allocation. In practice, therefore, the chosen models should admit efficient computation of $G_{i,j}$. In this section we examine two classes of models for which $G_{i,j}$ can be efficiently computed.

9.2.1 Hyperbolic Model

In [104], the following simple "hyperbolic" model forms the basis of an adaptive bit allocation algorithm:

$$f_i(q_i) = \frac{\alpha_i}{q_i} + \beta_i, \tag{9.1}$$

where α_i is associated with the complexity of coding picture i and β_i with the overhead for coding the picture. The hyperbolic model is one of the simplest models to exhibit the monotonicity and concavity characteristic of rate-distortion functions.[1] Several instances of the hyperbolic model are plotted in Figure 9.1. TM5 adopts a similar model where only the complexity term is used. With adaptive quantization techniques, α_i and β_i are typically estimated from the results of encoding previous pictures. The parameters can also be determined by coding a sampling of blocks in picture i and fitting the parameters to the coding statistics.

With the hyperbolic model, there is a simple closed-form expression for $G_{i,j}$:

$$G_{i,j}(b) = \frac{\sum_{i \le k \le j} \alpha_k}{b - \sum_{i \le k \le j} \beta_k}. \tag{9.2}$$

As previously discussed in Section 7.2.5, we can precompute the cumulative sums for α_i and β_i in linear time and space and then use these to compute $G_{i,j}$ in constant time. This results in a time complexity of $O(N^2)$ for both optimal CBR and VBR allocation.

In related work, Ding and Liu [23] propose the following more general class of bit-production models and describe its use in rate control:

$$f_i(q) = \frac{\alpha_i}{q^{\gamma_i}} + \beta_i. \tag{9.3}$$

[1] Our framework only assumes monotonicity and not concavity.

The extra parameter γ_i is dependent on the picture type (I, P, or B) and is intended to capture the different rate-distortion characteristics for each picture type. One drawback to (9.3) is that the model is non-linear with respect to the parameters, and we know of no closed-form solution to $G_{i,j}$ in the general setting. Although numerical techniques can be used to solve for $G_{i,j}$, this could adversely affect the computational efficiency of the bit allocation algorithms.

In preliminary experiments, we find that the hyperbolic model works well near the operating point where α_i and β_i have been determined, but is not reliable at a distant operating point. This observation leads us to formulate the following encoding strategy.

1. Encode the sequence using the standard TM5 coder, keeping statistics (for each picture) of the average nominal quantization scale, the total number of bits used, and the number of bits used to code the quantized DCT coefficients.

2. Compute α_i and β_i from the statistics gathered in the previous encoding pass. Allocate bits to pictures with the lexicographic bit allocation algorithm and encode the sequence using this allocation, gathering statistics as before.

3. Repeat Step 2.

The idea is that with each encoding, the accuracy of the bit models will improve as the operating Q is determined and refined for each picture.

9.2.2 Linear-Spline Model

As noted above, the hyperbolic model works well with small changes in the quantization scale Q. However, with a large variation in Q between successive pictures, as may occur with a scene change, the model becomes less reliable. This is because the model is defined by only two parameters α_i and β_i. Previously, we have compensated for this by performing multiple encoding passes to ensure that the parameters are determined close to the actual operating point. We now consider a different approach where more effort is expended to construct more accurate bit models that are then used to encode the video sequence in a single pass.

Lin, Ortega, and Kuo [64, 63] propose using cubic-spline interpolation models of rate and distortion in conjunction with a gradient-based rate control algorithm [61, 62]. The spline models are computed by first encoding each picture several times using a select set of M quantization scales, $\{x_1, x_2, \ldots, x_M\}$ with $x_1 < x_2 < \cdots < x_M$, and measuring the actual rate. Each quantization/rate pair is called a *control point*. For picture i, the function between two consecutive control points $(x_k, y_{i,k})$ and $(x_{k+1}, y_{i,k+1})$ has the form

$$f_i^k(x_k) = a_{ik}x^3 + b_{ik}x^2 + c_{ik}x + d_{ik}. \tag{9.4}$$

The parameters a_{ik}, b_{ik}, c_{ik}, and d_{ik} are computed from four control points $(x_{k-1}, y_{i,k-1})$, $(x_k, y_{i,k})$, $(x_{k+1}, y_{i,k+1})$, and $(x_{k+2}, y_{i,k+2})$, such that $f_i^k(x_k) = y_{i,k}$ and $f_i^{k+1}(x_{k+1}) = y_{i,k+1}$ and the first-order derivatives of f_i^k and f_i^{k+1} are continuous at the control points. The authors suggest using the Fibonacci-like set $\{1, 2, 3, 5, 8, 13, 21, 31\}$ for the control quantization scales to exploit the exponential-decay typical of rate-distortion functions.

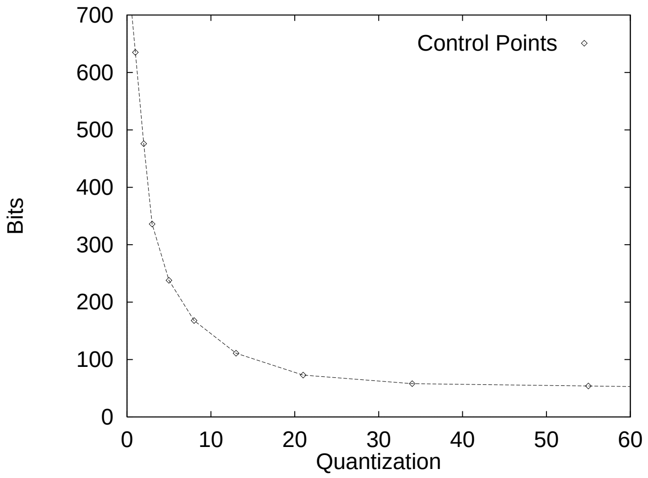


Figure 9.2: Example of a linear-spline interpolation model.

One drawback of a cubic-spline model is that it is generally not monotonic. To ensure monotonicity, we consider a simpler linear-spline interpolation model, where a line segment is used to interpolate the bit-production function between control points. For picture i, the function between two consecutive control points $(x_k, y_{i,k})$ and $(x_{k+1}, y_{i,k+1})$ has the form

$$f_i^k(x_k) = \alpha_{i,k}x + \beta_{i,k}. \tag{9.5}$$

As in [61, 62], we choose the control quantization scales to be $\{1, 2, 3, 5, 8, 13, 21, 31\}$ to exploit the exponential-decay property of rate-distortion functions. In case the control points themselves do not exhibit monotonicity, we enforce monotonicity by skipping those control points where the monotonicity property is violated. For quantization scales less than x_1 or greater than x_M, we extrapolate using the parameters $(\alpha_{i,k}, \beta_{i,k})$ or $(\alpha_{i,M-1}, \beta_{i,M-1})$, respectively. An example of a linear-spline model is shown in Figure 9.2.

The linear-spline model has a simple closed-form expression for $G_{i,j}$ if we know the two control points that bracket the operating point. Because of the monotonicity property, we can determine the two bracketing points using binary search. Between the control points x_k and x_{k+1}, $G_{i,j}$ can be computed as

$$G_{i,j}(b) = \frac{b - \sum_{i \le m \le j} \beta_{m,k}}{\sum_{i \le m \le j} \alpha_{m,k}}. \tag{9.6}$$

If $x_k \le G_{i,j} \le x_{k+1}$ then the correct operating point has been found. If $G_{i,j} < x_k$, the operating point must lie between two control points with lower indices. Similarly, if $G_{i,j} > x_{k+1}$, the operating point must lie between two control points with higher indices. A simple binary search procedure can be used to find the operating point.

Since there are a fixed number of control points, we can compute $G_{i,j}$ in constant time with linear-time preprocessing. As with the hyperbolic model, we can compute optimal CBR and VBR allocations in quadratic time.

The cubic-spline model of [64, 63] is used in a dependent-coding framework, where the effects of coding previous pictures are taken into account in the modeling. Our framework assumes independent coding and does not take these effects into account. However, from the switching theorems, we note that an optimal allocation has segments of constant Q. This provides a basis for estimating the linear-spline model parameters. By encoding the video sequence multiple times with a constant Q determined from the control points, we can construct a linear-spline interpolation model for each picture. We expect these models to be reasonably accurate within a segment of constant Q. At the boundary between segments, however, we can expect some discrepancy in the models for dependent pictures (P and B types).

9.3 Picture-Level Rate Control

Even with accurate bit-production models, the actual number of bits produced will inevitably depart from the model. There are essentially two ways to cope with bit-modeling errors.

9.3.1 Closed-Loop Rate Control

A popular approach taken in TM5 is to regulate the quantization scale at the macroblock level while coding a picture so that the desired bit allocation is met. This is achieved with a *closed-loop* feedback mechanism using the fullness of a virtual encoder buffer to control the macroblock quantization. One drawback of this technique is that the coded quality within a picture may vary considerably, especially for a picture that contains regions of varying complexity. With gross errors in the bit-production models, the actual average quantization scale may differ markedly from the desired quantization scale, thereby adversely affecting the coded quality.

9.3.2 Open-Loop Rate Control

Another approach is to perform *open-loop* control where the assigned (nominal) quantization scale is used to code a picture. We can then adjust the bit allocation of the remaining uncoded pictures to compensate for the difference between desired and actual bit production. An advantage of this approach is that the quality is more constant within a picture. In addition, less processing is required to code each picture. A disadvantage is that, since the bit production is not controlled below the picture layer, the actual bit production may vary from the target and potentially cause the buffer to overflow or underflow.

After coding a picture, we can reallocate bits to the remaining pictures optimally (for the given models). Instead of recomputing an optimal allocation from scratch and incurring an extra factor of N in the time complexity, we can take advantage of dynamic programming to increase the time complexity by only a constant factor. We do this for a CBR allocation and for hard pictures in a VBR allocation by constructing the dynamic programming table in the CBR algorithm *in reverse.*

As presented in Section 7.2, the dynamic programming algorithm works by solving for sub-allocations for pictures 1 to k for increasing values of k. We can also rework the dynamic programming to compute optimal sub-allocations for pictures k to N for decreasing values of k. We do this by computing optimal allocations that start with the buffer empty or full at picture k and ends with the buffer at the final buffer state after picture N.

With a reverse dynamic programming table, we can compute a revised allocation for picture k, after encoding picture $k - 1$, by searching for a proper constant-Q connector starting with the known VBV buffer fullness before picture k is removed. With the reverse dynamic programming table available, this search consumes $O(N)$ time for the hyperbolic and linear-spline interpolation models. The total additional time to recover from bit-production errors is then $O(N^2)$, the same as the time complexity for computing the initial allocation.

As stated, the above procedure applies to a CBR allocation and to hard pictures in a VBR allocation (which are allocated using the CBR routine). For easy pictures in a VBR allocation, we can simply recompute a new value for $Q_{\min}$. Here, we assume that errors in bit-production modeling are not severe enough to change the classification of hard and easy pictures.

9.3.3 Hybrid Rate Control

In early experiments, we observed that closed-loop rate control resulted in rapid fluctuations in the nominal quantization scale[2] between pictures owing to the buffer-feedback mechanism. With accurate bit-production models, however, the need to perform low-level rate control below the picture level is questionable. This suggests using open-loop control. As noted earlier, since we assume independent coding, we can expect more errors in the bit-production models at pictures where the assigned Q changes. With these observations, we propose a hybrid rate control strategy where closed-loop control is used for pictures at the boundaries of a constant-Q segment and open-loop control is used for the rest. Another motivation for using closed-loop control for boundary pictures is that the VBV buffer should be either nearly empty or nearly full for these pictures, and the bit rate must be carefully controlled to avoid underflowing or overflowing the buffer.

9.4 Buffer Guard Zones

Even with the picture-level rate control strategies outlined above, there is still the possibility of the VBV buffer overflowing or underflowing. To safeguard against this, we compute a bit allocation using a slightly smaller buffer than that specified in the MPEG bitstream so that we can have *guard zones* near the top and bottom of the buffer. For the experiments with CBR, have we chosen to place the guard zones at 5% and 95% of maximum buffer size. This is illustrated in Figure 9.3. For VBR mode, the upper guard zone is not needed since buffer overflow is not a concern.

9.5 Encoding Simulations

To assess the behavior and effectiveness of the lexicographic bit allocation algorithms, the bit-production models, and the rate control strategies outlined above, we conducted encoding simulations using several short (≈ 100 pictures) benchmark video sequences (`flower garden`, `football`, `mobile`, and `table tennis`) in SIF format and a longer (3,660 pictures) promotional video clip courtesy of IBM Corporation.

[2]By nominal quantization scale, we mean the average measured macroblock quantization scale with perceptual quantization factored out.

Figure 9.3: Guard zones to safeguard against underflow and overflow of VBV buffer. A bit allocation is computed so that the buffer fullness remains between the guard zones.

9.5.1 Initial Experiments

Initially, we used the short video clips to evaluate the bit-production models and rate control strategies. To simulate scene changes, we concatenated the four short video clips into a 418–picture video sequence in the following order: `flower garden`, `mobile`, `football`, `table tennis`.

We implemented the rate control algorithms within the software encoder provided by the MPEG-2 Simulation Group [74]. The coding parameters are listed in Table 9.1. For CBR mode, we specified a peak bit rate of 1.0 Mbits/sec. For VBR, we used an average bit rate of 1.0 Mbits/sec and a peak bit rate of 1.2 Mbits/sec. The VBV buffer size was set to 720,896 bits. For reference, we also ran the encoder with TM5 rate control using the sample parameters.[3] In order to reduce factors that would affect the actual bit production, full-search motion estimation was initially performed using a fixed nominal quantization scale of 13, and the same motion vectors were then used for all the encodings. The coding decisions, however, were still determined on-line.

In the first set of simulations, we used the hyperbolic model and performed multiple encoding passes. The results of the encodings are presented in Table 9.2 and Figures 9.4 to 9.9. The table collects some summary statistics for the various coders. Figures 9.4 and 9.5 show the evolution of the buffer fullness. Figures 9.6 and 9.7 plot the computed and observed sequence of nominal quantization **Q**. The computed **Q** consists of piecewise constant segments, as dictated by theory.

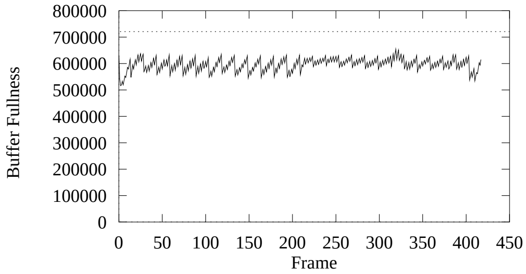
The initial pass of the Hyperbolic CBR and VBR coders used statistics gathered with the TM5 coder in order to determine the parameters of the bit-production models. Later passes of the Hyperbolic CBR (VBR) coder used statistics gathered from the previous pass. From the results in Table 9.2 and Figure 9.4, the Hyperbolic CBR coder does not exhibit much change between passes. However, the Hyperbolic VBR coder does show reduction in the standard deviation in PSNR and nominal Q and better usage of the VBV buffer with later passes.

As evident from Figure 9.4, the TM5 coder uses only a fraction of the VBV buffer and maintains the buffer relatively level. In contrast, the lexicographic coders make better use of the VBV buffer.

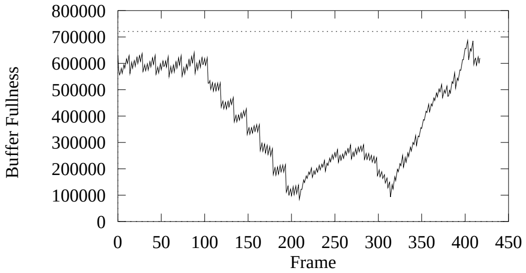
[3]A minor modification was made to the TM5 model in that the levels of the virtual encoding buffers used to regulate the quantization scale are restricted to the range $[0, 2r]$, where r is the reaction parameter defined in Section 3.7.5.

Value	Description
418	number of frames
1	number of first frame
15	N (# of frames in GOP)
3	M (I/P frame distance)
0	ISO/IEC 11172-2 stream
0	0:frame pictures, 1:field pictures
352	horizontal_size
240	vertical_size
2	aspect_ratio_information 1=square pel, 2=4:3, 3=16:9, 4=2.11:1
5	frame_rate_code 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec
1000000.0	bit_rate (bits/sec)
44	vbv_buffer_size (in multiples of 16 kbit)
0	low_delay
0	constrained_parameters_flag
4	Profile ID: Simple = 5, Main = 4, SNR = 3, Spatial = 2, High = 1
8	Level ID: Low = 10, Main = 8, High 1440 = 6, High = 4
1	progressive_sequence
1	chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4
2	video_format: 0=comp., 1=PAL, 2=NTSC, 3=SECAM, 4=MAC, 5=unspec.
5	color_primaries
5	transfer_characteristics
4	matrix_coefficients
352	display_horizontal_size
240	display_vertical_size
0	intra_dc_precision (0: 8 bit, 1: 9 bit, 2: 10 bit, 3: 11 bit
0	top_field_first
1 1 1	frame_pred_frame_dct (I P B)
0 0 0	concealment_motion_vectors (I P B)
1 1 1	q_scale_type (I P B)
1 0 0	intra_vlc_format (I P B)
0 0 0	alternate_scan (I P B)
0	repeat_first_field
1	progressive_frame
0	P distance between complete intra slice refresh
0	rate control: r (reaction parameter)
0	rate control: avg_act (initial average activity)
0	rate control: Xi (initial I frame global complexity measure)
0	rate control: Xp (initial P frame global complexity measure)
0	rate control: Xb (initial B frame global complexity measure)
0	rate control: d0i (initial I frame virtual buffer fullness)
0	rate control: d0p (initial P frame virtual buffer fullness)
0	rate control: d0b (initial B frame virtual buffer fullness)
3 3 23 23	P: forw_hor_f_code forw_vert_f_code search_width/height
1 1 7 7	B1: forw_hor_f_code forw_vert_f_code search_width/height
2 2 15 15	B1: back_hor_f_code back_vert_f_code search_width/height
2 2 15 15	B2: forw_hor_f_code forw_vert_f_code search_width/height
1 1 7 7	B2: back_hor_f_code back_vert_f_code search_width/height

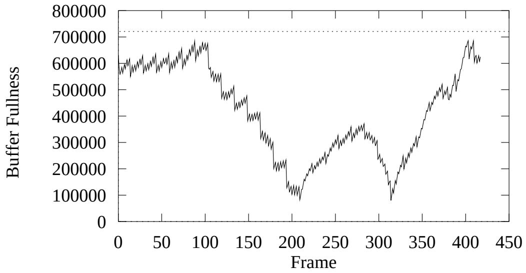
Table 9.1: Parameters for MPEG-2 Simulation Group software encoder used to encode the SIF-formatted video clips.



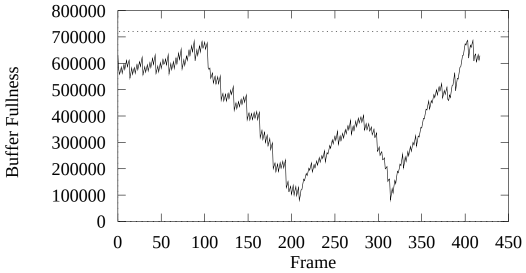
(a) TM5 CBR



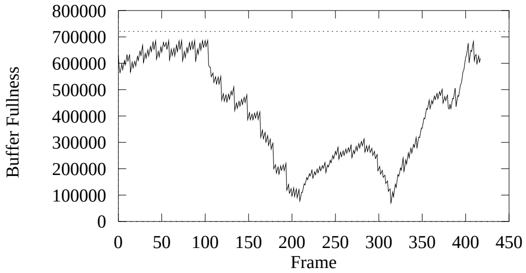
(b) Hyperbolic CBR Pass 1



(c) Hyperbolic CBR Pass 2



(d) Hyperbolic CBR Pass 3

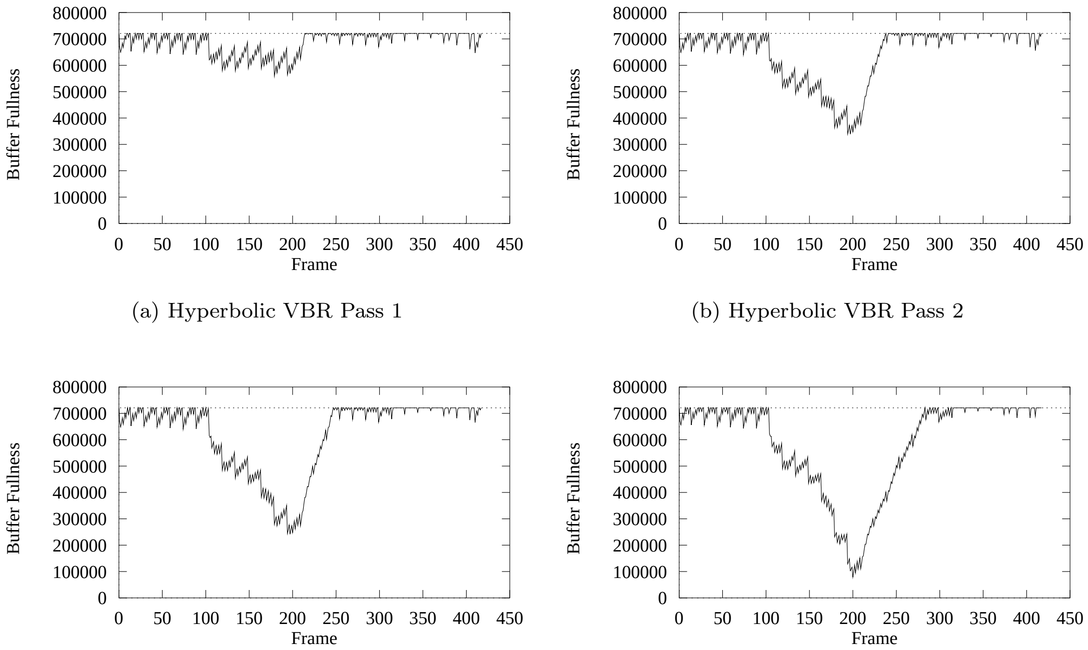


(e) Linear-Spline CBR

Figure 9.4: Evolution of buffer fullness for CBR coders.

Method	Average PSNR (dB)	Std. Dev. of PSNR	Average Nom. Q	Std. Dev. of Nom. Q	Maximum Nom. Q	Minimum Nom. Q
TM5 CBR	26.66	3.06	16.19	4.48	26.20	6.70
Hyperbolic CBR, Pass 1	26.48	2.67	16.19	3.44	22.63	7.78
Hyperbolic CBR, Pass 2	26.48	2.66	16.23	3.40	21.44	7.43
Hyperbolic CBR, Pass 3	26.48	2.67	16.28	3.41	20.68	7.20
Hyperbolic VBR, Pass 1	26.54	2.36	15.95	2.77	21.15	9.16
Hyperbolic VBR, Pass 2	26.52	2.10	16.00	2.14	19.60	9.34
Hyperbolic VBR, Pass 3	26.49	1.99	16.09	1.83	19.22	9.76
Linear-Spline CBR, Hybrid	26.73	2.68	15.80	3.36	19.48	8.29
Linear-Spline VBR, Hybrid	26.66	1.87	15.83	1.13	17.59	12.97

Table 9.2: Summary of initial coding experiments.



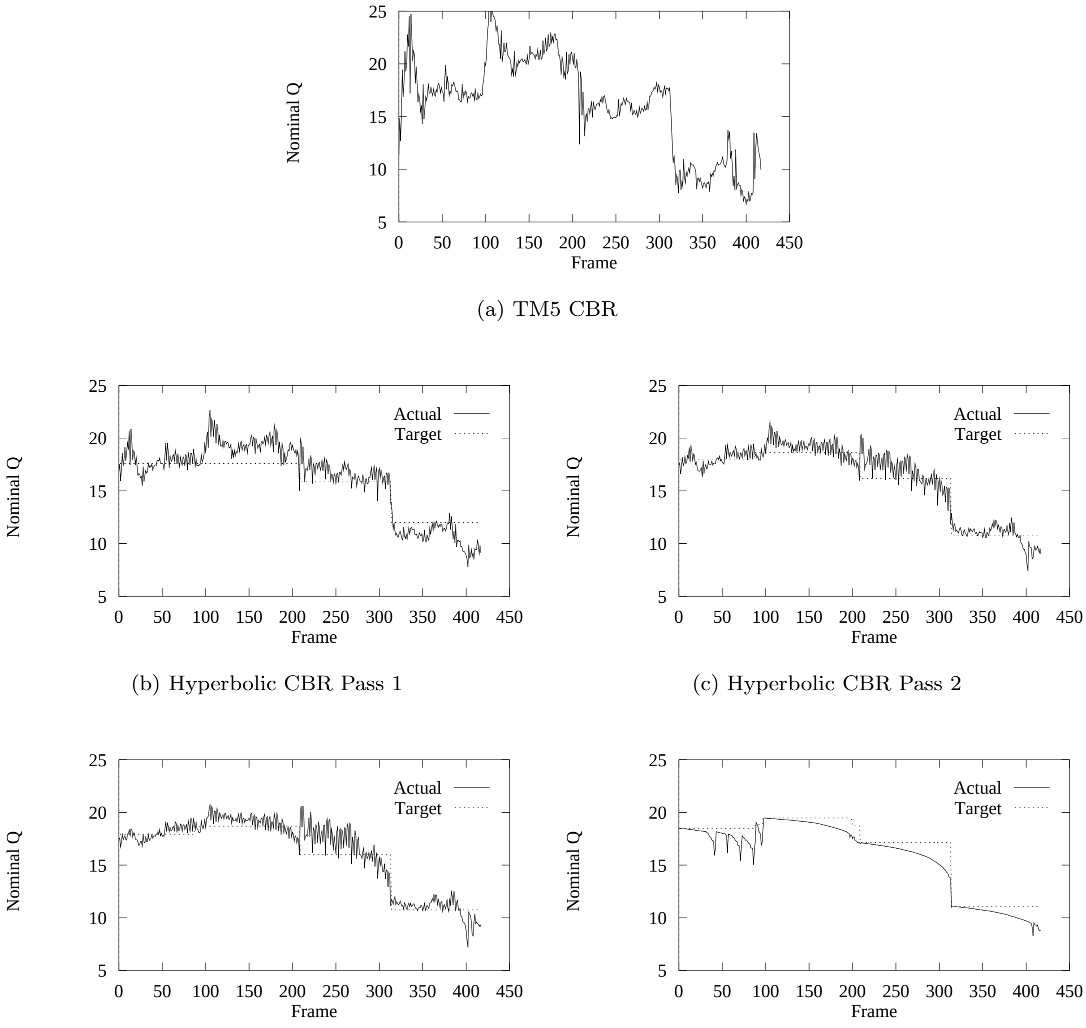
(a) Hyperbolic VBR Pass 1

(b) Hyperbolic VBR Pass 2

(c) Hyperbolic VBR Pass 3

(d) Linear-Spline VBR

Figure 9.5: Evolution of buffer fullness for VBR coders.



(a) TM5 CBR

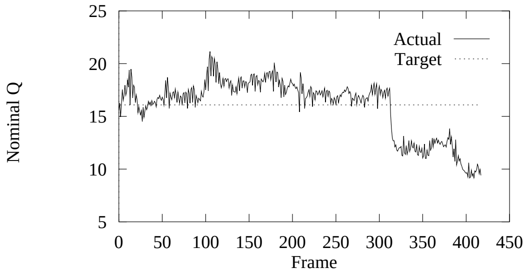
(b) Hyperbolic CBR Pass 1

(c) Hyperbolic CBR Pass 2

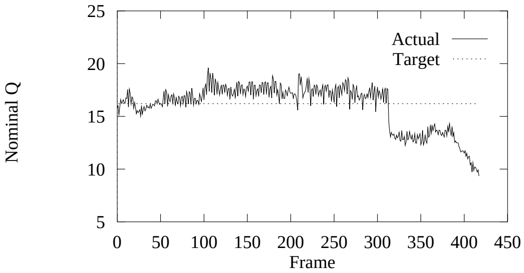
(d) Hyperbolic CBR Pass 3

(e) Linear-Spline CBR

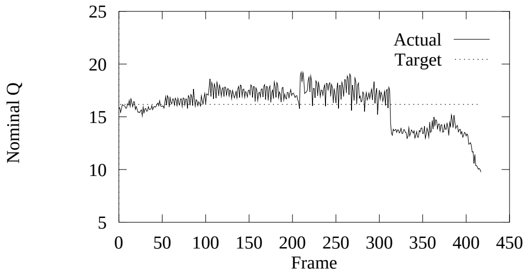
Figure 9.6: Nominal quantization scale for CBR coders.



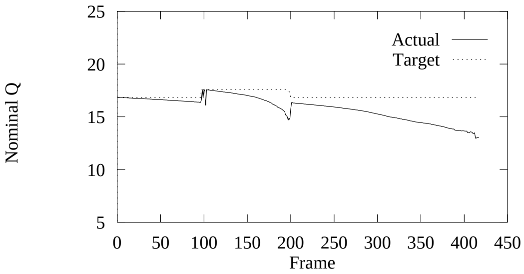
(a) Hyperbolic VBR Pass 1



(b) Hyperbolic VBR Pass 2

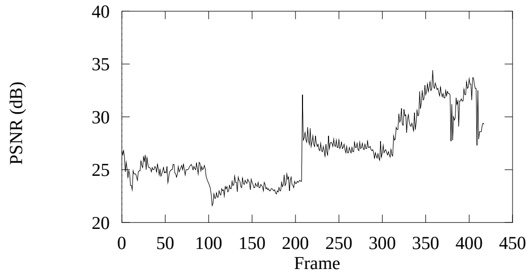


(c) Hyperbolic VBR Pass 3

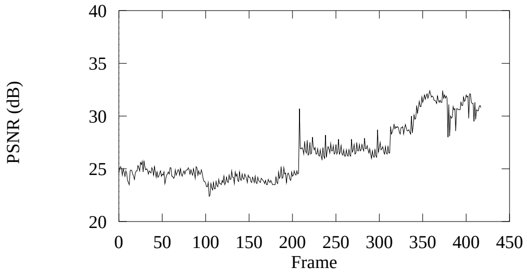


(d) Linear-Spline VBR

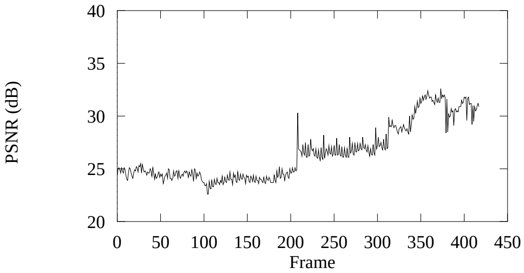
Figure 9.7: Nominal quantization scale for VBR coders.



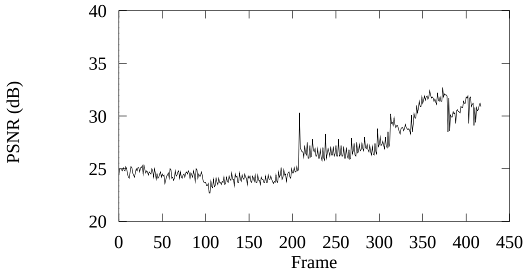
(a) TM5 CBR



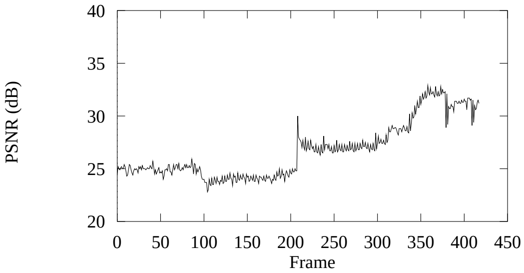
(b) Hyperbolic CBR Pass 1



(c) Hyperbolic CBR Pass 2

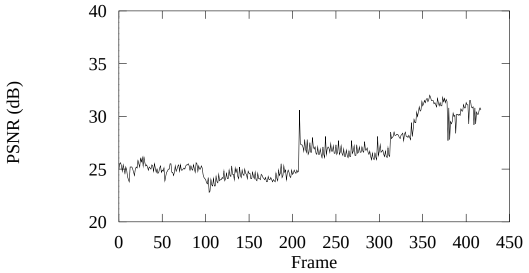


(d) Hyperbolic CBR Pass 3

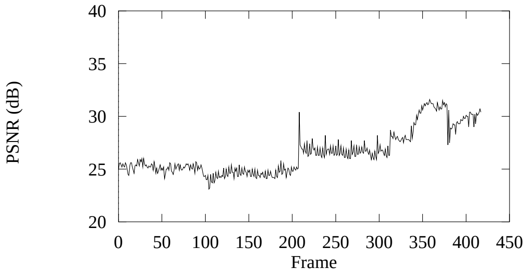


(e) Linear-Spline CBR

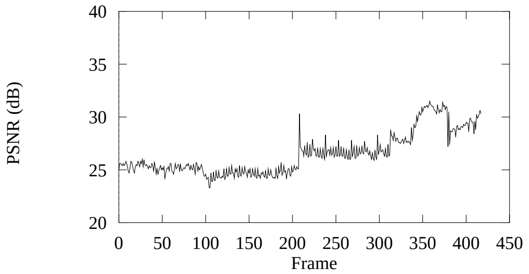
Figure 9.8: PSNR for CBR coders.



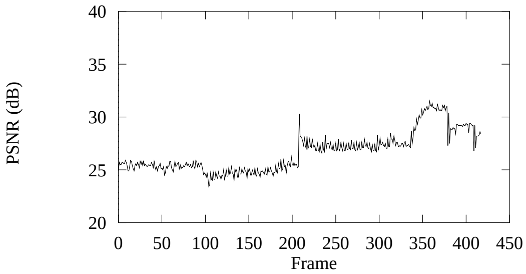
(a) Hyperbolic VBR Pass 1



(b) Hyperbolic VBR Pass 2



(c) Hyperbolic VBR Pass 3



(d) Linear-Spline VBR

Figure 9.9: PSNR for VBR coders.

Comparing hyperbolic modeling with closed-loop rate control on the one hand with linear-spline modeling with hybrid rate-control on the other, we see that the latter outperforms the former in all aspects. It is noteworthy that the hyperbolic model seems to underestimate the bit production while the linear-spline model overestimates the bit production. The result is that the actual nominal quantization scales used are higher than the target for the hyperbolic model and lower for the linear-spline model.

9.5.2 Coding a Longer Sequence

Since the video clips used in the initial experiments are short and we had to concatenate them to form somewhat artificial scene changes, we were not able to detect much perceptual difference between the different encodings. To assess the perceptual gains of lexicographically optimal bit allocation, we performed additional coding simulations using a longer video sequence with varied and dynamic content. The sequence consists of 3,660 frames of a commercial produced by the IBM Corporation to demonstrate its MPEG-2 encoding chipset. The clip starts with a fade-in to a spokeswoman standing in front of a slowing changing background. A block diagram in one corner of the picture then rotates and zooms to fill the screen. The diagram then remains static with some illumination changes before fading back to the spokeswoman. On one side of the picture, a collage of different video clips scroll up the screen. One of the clips zooms to occupy the full picture. The clips cycle through a variety of action-filled scenes from horses running to a skydiver rotating on a skateboard to a bicycle race and finally to highlights from a basketball game.

The video sequence is in NTSC CCIR-601 format. We coded the sequence in CBR mode at 3 Mbits/sec and in VBR mode at 3 Mbits/sec average and 4.5 Mbits/sec peak. The encoding parameters used are shown in Table 9.3. The VBV buffer size is 1,835,008 bits. We used linear-spline interpolation models in conjunction with the hybrid rate control strategy.

Some encoding statistics are listed in Table 9.4. The buffer fullness, nominal Q, and PSNR plots are shown in Figures 9.10, 9.11, and 9.12, respectively. The differences between the different coders are much more pronounced with these simulations than with the previous ones. The lexicographic CBR coder is able to control the quantization to a narrower range than the TM5 coder, with a resulting increase in PSNR. The lexicographic VBR coder sacrifices quality in earlier pictures in order to code better the later more complex pictures. The result is that the nominal quantization is nearly constant and the PSNR plot is more even.

Visually, the lexicographic VBR coder produced near constant-quality video with few noticeable coding artifacts. In contrast, both CBR coders produced noticeable blocking artifacts in scenes with high motion, especially in the basketball scene. However, the lexicographic CBR coder fared noticeably better than TM5 at maintaining constant quality through scene changes and reducing artifacts during complex scenes of short duration.

9.6 Limiting Lookahead

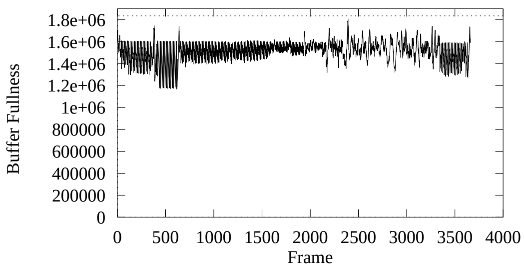
The above rate control algorithms compute an allocation for the entire video sequence. This may not be feasible when the sequence consists of many pictures, as in a feature-length movie, for

Value	Description
3660	number of frames
0	number of first frame
15	N (# of frames in GOP)
3	M (I/P frame distance)
0	ISO/IEC 11172-2 stream
0	0:frame pictures, 1:field pictures
720	horizontal_size
480	vertical_size
2	aspect_ratio_information 1=square pel, 2=4:3, 3=16:9, 4=2.11:1
4	frame_rate_code 1=23.976, 2=24, 3=25, 4=29.97, 5=30 frames/sec
3000000.0	bit_rate (bits/sec)
112	vbv_buffer_size (in multiples of 16 kbit)
0	low_delay
0	constrained_parameters_flag
4	Profile ID: Simple = 5, Main = 4, SNR = 3, Spatial = 2, High = 1
8	Level ID: Low = 10, Main = 8, High 1440 = 6, High = 4
0	progressive_sequence
1	chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4
2	video_format: 0=comp., 1=PAL, 2=NTSC, 3=SECAM, 4=MAC, 5=unspec.
5	color_primaries
5	transfer_characteristics
4	matrix_coefficients
720	display_horizontal_size
480	display_vertical_size
0	intra_dc_precision (0: 8 bit, 1: 9 bit, 2: 10 bit, 3: 11 bit
1	top_field_first
0 0 0	frame_pred_frame_dct (I P B)
0 0 0	concealment_motion_vectors (I P B)
1 1 1	q_scale_type (I P B)
1 0 0	intra_vlc_format (I P B)
0 0 0	alternate_scan (I P B)
0	repeat_first_field
0	progressive_frame
0	P distance between complete intra slice refresh
0	rate control: r (reaction parameter)
0	rate control: avg_act (initial average activity)
0	rate control: Xi (initial I frame global complexity measure)
0	rate control: Xp (initial P frame global complexity measure)
0	rate control: Xb (initial B frame global complexity measure)
0	rate control: d0i (initial I frame virtual buffer fullness)
0	rate control: d0p (initial P frame virtual buffer fullness)
0	rate control: d0b (initial B frame virtual buffer fullness)
4 4 63 63	P: forw_hor_f_code forw_vert_f_code search_width/height
2 2 15 15	B1: forw_hor_f_code forw_vert_f_code search_width/height
3 3 31 31	B1: back_hor_f_code back_vert_f_code search_width/height
3 3 31 31	B2: forw_hor_f_code forw_vert_f_code search_width/height
2 2 15 15	B2: back_hor_f_code back_vert_f_code search_width/height

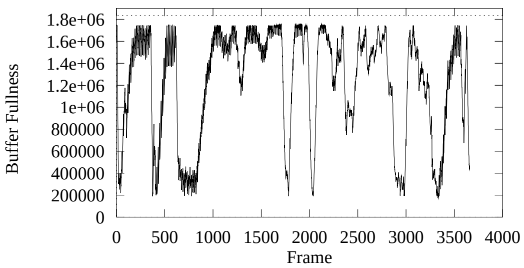
Table 9.3: Parameters for MPEG-2 Simulation Group software encoder used to encode the IBM commercial.

Method	Average PSNR (dB)	Std. Dev. of PSNR	Average Nom. Q	Std. Dev. of Nom. Q	Maximum Nom. Q	Minimum Nom. Q
TM5 CBR	33.34	4.95	14.00	9.88	48.86	1.95
Linear-Spline CBR, Hybrid	33.45	4.77	13.01	7.97	29.62	2.61
Linear-Spline VBR, Hybrid	33.08	2.58	11.80	2.07	17.68	8.00

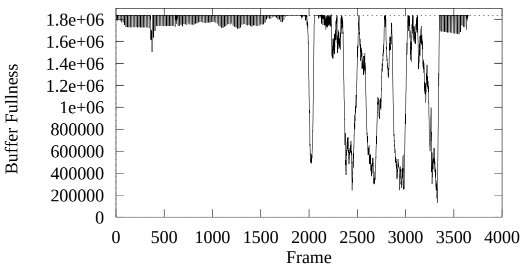
Table 9.4: Summary of coding simulations with IBM Commercial.



(a) TM5 CBR

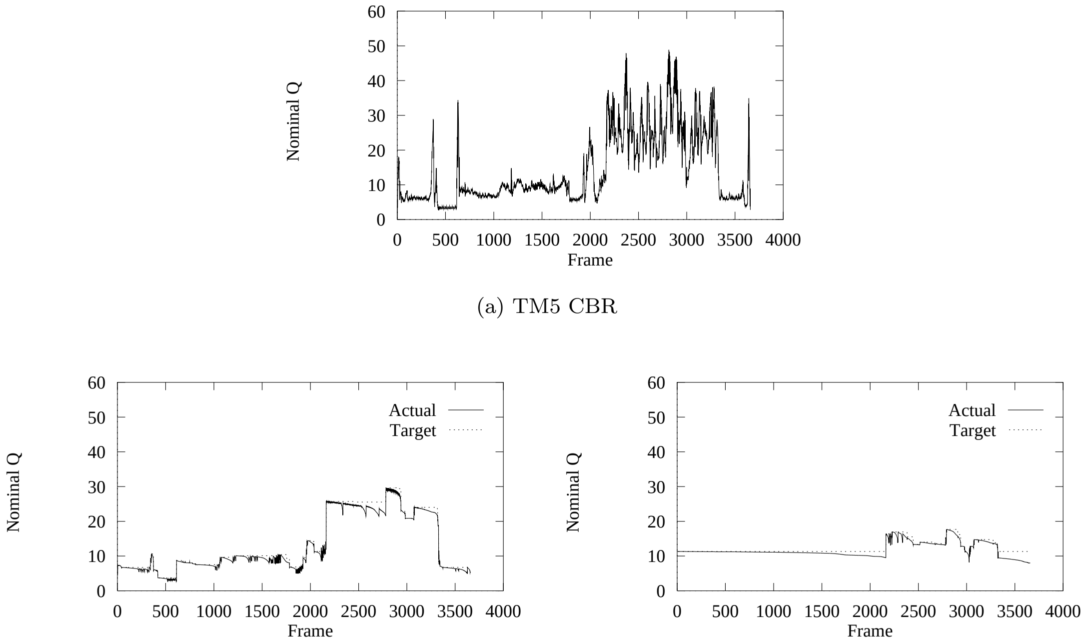


(b) Linear-Spline CBR



(c) Linear-Spline VBR

Figure 9.10: Evolution of buffer fullness for coding IBM Commercial.



(a) TM5 CBR

(b) Linear-Spline CBR

(c) Linear-Spline VBR

Figure 9.11: Nominal quantization scale for coding IBM Commercial.

example. One way to deal with this is to partition the sequence into blocks consisting of a small number of consecutive pictures. Optimal allocation can then be performed on the blocks separately. In order to do this, the starting and ending buffer fullness must be specified for each block for the CBR case. For the VBR case, the bit budget must also be specified for each block. This approach is globally suboptimal; however, it is easy to parallelize since the block allocations are independent of each other.

Another approach is to use limited lookahead in conjunction with hybrid rate control. Using a lookahead window of size W and a step size $S \leq W$, the procedure is as follows:

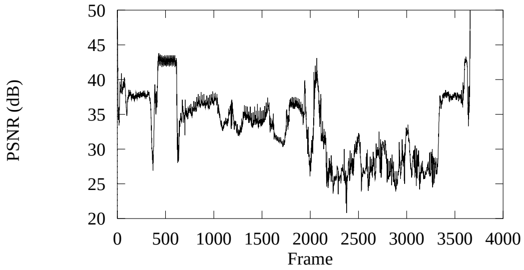
1. Compute a bit allocation for the next W pictures not yet coded by computing a reverse dynamic programming table.

2. Code the next S pictures using hybrid rate control, using the dynamic programming table to recover from model errors.

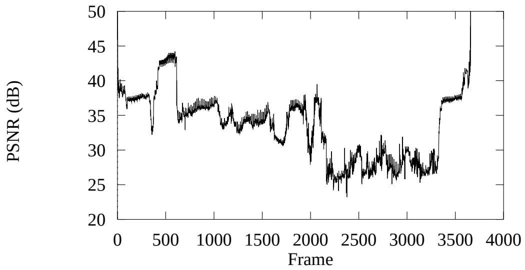
3. Repeat Step 1.

This procedure can be thought of as performing lookahead with a sliding window.

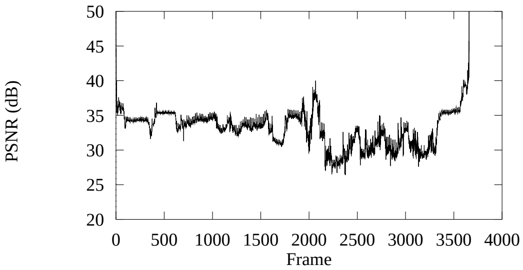
Another approach similar to the hybrid rate control method is to use the allocation computed from a given model and only recompute the allocation when the buffer fullness breach preset buffer boundaries, such as 10% and 90% of buffer fullness. As with hybrid rate control, reverse dynamic programming can be used to speed up the reallocation.



(a) TM5 CBR



(b) Linear-Spline CBR



(c) Linear-Spline VBR

Figure 9.12: PSNR for coding IBM Commercial.

9.7 Related Work

A suite of heuristic methods is proposed in [79] to reduce the complexity as compared to an optimal bit allocation based on the Viterbi algorithm. A Lagrangian optimization technique is applied to recompute an allocation incrementally for each picture, similar to technique described in Section 9.3.2. In addition, the Lagrangian optimization is performed with a finite window size. In essence, this method implements limited lookahead with a sliding window, similar to the technique described in Section 9.6. The authors also describe the heuristic of recomputing a allocation only when the buffer reaches predefined threshold levels.

In this chapter, we have considered a simple hyperbolic model and a linear-spline model of bit production. In [10], a bit-production model is derived for block-transform coders based on rate-distortion theory and assuming a stationary Gaussian process. The model is applied for VBR coding with motion JPEG and H.261 coders. In [12], an adaptive tree-structured piecewise linear bit-production model is proposed and applied to MPEG video coding using a one-pass encoding strategy. A cubic-spline model of rate and distortion is proposed in [64, 63] for use with a gradient-based rate-control algorithm [61, 62] that attempts to minimize MSE. The model takes into account the temporal dependencies introduced by predictive coding.

9.8 Discussion

In this chapter, we have described an implementation of the bit allocation algorithms of Chapters 7 and 8 within an MPEG-2 software encoder. In addition to evaluating two types of bit-production models, we have developed robust techniques for recovering from errors in the bit-production models. Since we can view coding dependencies as contributing to errors in the (independent) bit-production models, these error-recovery techniques effectively allow us to apply the bit-allocation framework to predictive video coders.

Chapter 10

Extensions of the Lexicographic Framework

In Chapters 6 through 8, we laid a theoretical foundation for lexicographic bit allocation, and in Chapter 9 we demonstrated that the framework works well in practice. In this chapter, we provide evidence that the framework is also flexible and general by showing how it can be readily applied to other domains, extended to perform statistical multiplexing, and used in a discrete optimization setting.

10.1 Applicability to Other Coding Domains

While the lexicographic bit allocation framework was originally motivated and formulated for MPEG video coding, it can be applied equally well in other lossy coding domains for which buffer-constrained bit allocation is a valid problem and where a perceptual distortion measure needs to be equalized among coding units. Obvious examples include lossy image coding (such as specified by the JPEG standard [83]), where the coding unit would logically be a block of pixels, and audio coding, where a coding unit might correspond to half a second of sampled sound.

10.2 Multiplexing VBR Streams over a CBR Channel

10.2.1 Introduction

There are many scenarios where multiple compressed video streams are to be transmitted through a common channel. Two obvious examples are networked video and digital video broadcasting. In these types of applications, the transmission channel is typically bandwidth-limited. With the available bandwidth, we would like to provide as much video programming as possible without having to sacrifice quality.

An often-cited motivation for VBR video encoding is that VBR encoding can potentially allow for the simultaneous transmission of more video streams over a common channel than CBR encoding at the same quality level. The main reasoning is provided through a concept called *statistical multiplexing*. Statistical multiplexing is based on the observation that the bit rate of constant-quality video is highly variable from frame to frame. In order to achieve image quality that is *not less* lexicographically than that of a constant-quality VBR encoding, a CBR encoding would require a bit rate that would correspond to the peak rate of the VBR encoding. Since a VBR encoding typically requires the peak rate for only a small percentage of time, it uses less bandwidth on average than a comparable CBR encoding. Furthermore, assuming that the VBR streams have independent bit-rate characteristics, we can transmit more VBR streams than CBR streams over a common channel with a low probability that the combined instantaneous bit rate would exceed the channel rate.

As an example, consider a channel with a bandwidth of 100 Mbits/sec. Suppose that for a desired level of quality, a peak rate of 10 Mbits/sec is required for coding a suite of video programming. Using CBR encoding, up to 10 sequences can be transmitted through the channel simultaneously. Since the peak rate is required only a small percentage of the time, suppose that the actual average rate is only 5 Mbits/sec. Then using VBR encoding with a peak rate of 10 Mbits/sec and average rate of 5 Mbits/sec, we can *potentially* transmit 20 simultaneous sequences. This would correspond to a *statistical multiplexing gain* of 2.

In order to transmit the 20 VBR sequences simultaneously, however, the instantaneous bit rate for the 20 sequences must not exceed the channel capacity for an extended period of time, which is determined by the amount of buffering present. Assuming that the bit rates of the different video streams are uncorrelated in time, there is a low probability that the channel capacity would be exceeded in any time interval. Quantifying and minimizing this probability are central themes of research.

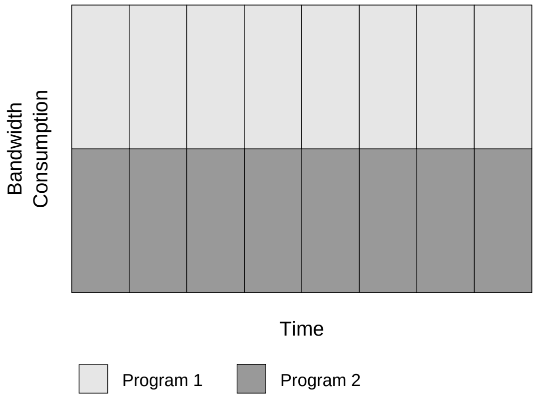
The advantage of VBR encoding over CBR is illustrated through a simple example in Figure 10.1. In this example, three VBR encodings are shown multiplexed using at most the same bandwidth required by a CBR encoding of only two of the sources.

In the remainder of this section, we will show how our basic lexicographic bit allocation framework can be readily extended to handle the multiplexing of multiple VBR bitstreams over a CBR channel. However, in contrast to typical statistical multiplexing techniques, as exemplified in Figure 10.1, our method allocates bits to the VBR bitstreams in a *deterministic* manner, making full use of all the available channel bandwidth.

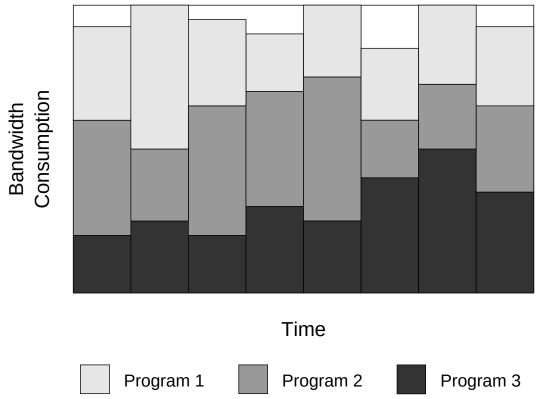
In related work, a buffered rate control scheme for multiplexing VBR sources onto a CBR channel is described in [81]. This work is based on the rate-distortion framework of [80] and [13] and uses a multiplexing model very similar to the one we are about to present. As described in the paper, the basic allocation unit is taken to be a GOP.

10.2.2 Multiplexing Model

We first elaborate a model for multiplexing multiple VBR bitstreams onto a CBR channel. Since our bit allocation framework is deterministic and uses lookahead, we assume that complete statistics of the multiple video sources are available to the bit allocation algorithm. This requirement can be met by providing a centralized encoder for the multiple sources, as depicted in Figure 10.2.



(a) Multiplexing of 2 CBR Bitstreams



(b) Multiplexing of 3 VBR Bitstreams

Figure 10.1: Example of how three VBR bitstreams can be multiplexed into the same channel as two CBR bitstreams, for a statistical multiplexing gain of 1.5.

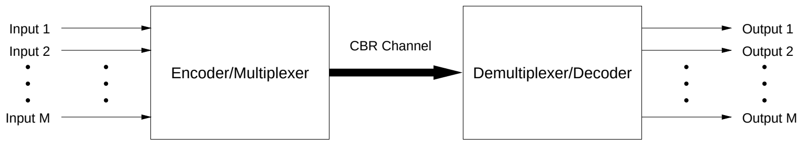


Figure 10.2: System for transmitting multiple sequences over a single channel.

In the figure, M video sources enter a encoder/multiplexer that produces a single multiplexed stream for transport over a CBR channel. On the receiving end, a demultiplexer/decoder performs demultiplexing and decoding to reproduce the M video sequences. This multiplexing model is similar to that proposed in [35].

This model is applicable to applications such as a video server where the video sequences to be multiplexed are known in advance. An especially noteworthy case is that of near-video-on-demand (NVOD), where a single sequence is to be transmitted simultaneously with different starting times. For example, 20 copies of a 2-hour long movie can be multiplexed so that the viewing of the movie can begin every six minutes.

The encoder/multiplexer block is expanded in Figure 10.3. As shown, the input video sources are encoded individually and time-division multiplexed and stored in a buffer before being output to the channel at a constant bit rate. The encoders are synchronized so that they output the encoding of a picture at the same time every T seconds. The multiplexer then concatenates the multiple encodings in order as shown in Figure 10.4.

The demultiplexer/decoder block is expanded in Figure 10.5. The demultiplexer/decoder mirrors the operation of the encoder/multiplexer. Incoming bits from the channel are stored in a decoding buffer. Every T seconds, the demultiplexer instantaneously removes from the buffer all bits needed to decode the next picture of all sequences and routes the bitstreams to the appropriate decoders, which then output the reconstructed video.

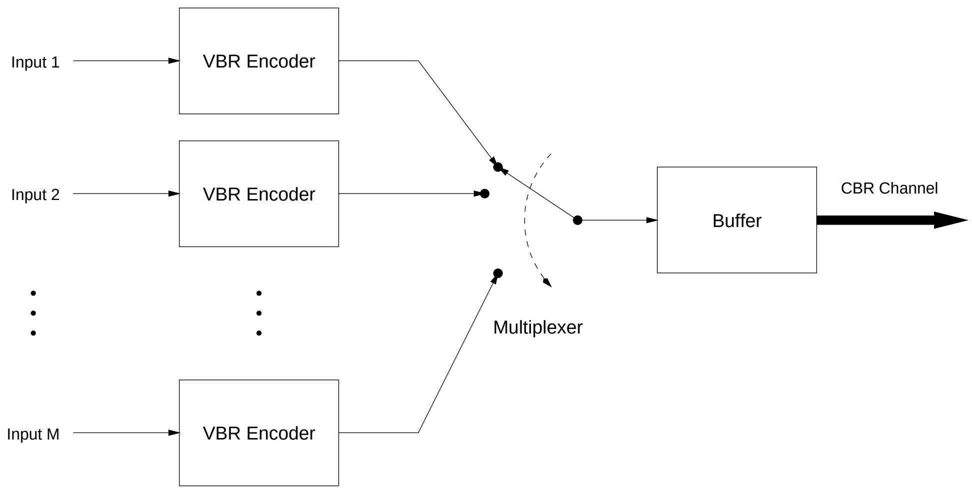


Figure 10.3: Block diagram of encoder/multiplexer.

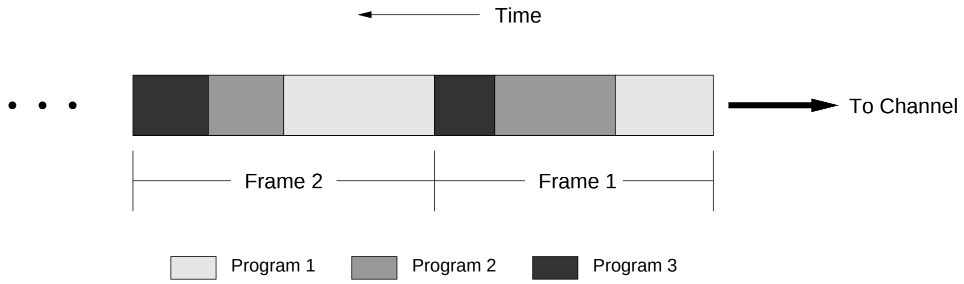


Figure 10.4: Operation of multiplexer.

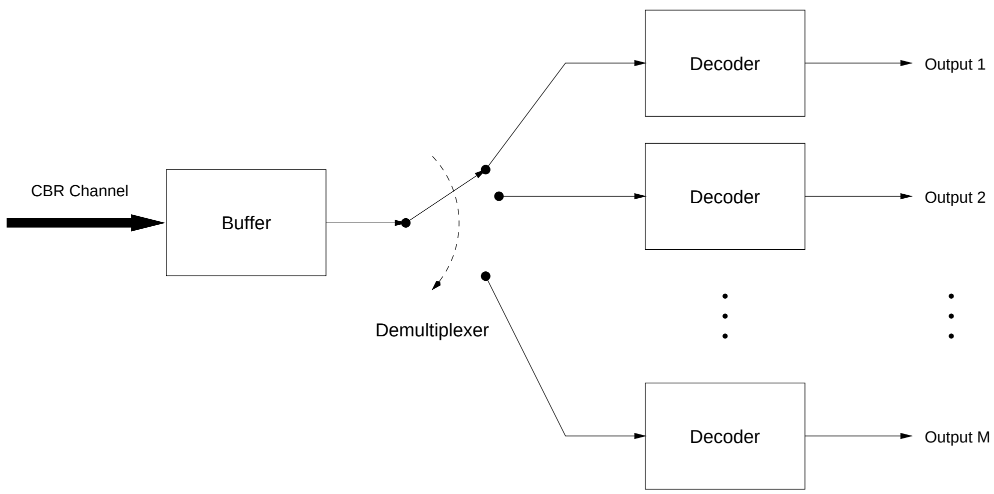


Figure 10.5: Block diagram of demultiplexer/decoder.

The multiplexing model described above resembles the operation of the single-stream encoder and decoder system implied by the MPEG Video Buffering Verifier. If we view the different input sources as providing "slices" of the same picture, the resemblance would be very close indeed. This construction is intentional and allows us to apply the lexicographic framework to allocate bits optimally to the multiple VBR bitstreams.

10.2.3 Lexicographic Criterion

Before we can apply our lexicographic bit allocation framework, we need to define an optimality criterion. Since there are multiple sources, a lexicographical criterion based on the encoding of a single video sequence is certainly not appropriate. We need to consider the quality of all the sequences. A simple way to do this is to consider the concatenation of all the sequences and define a lexicographic criterion on the concatenated video stream. By doing this, we are putting equal weight to each picture of each video sequence. We can also consider extending the lexicographic criterion to compare vectors of length M instead of scalar quantities. However, as we will see, this is equivalent to just considering the concatenation of the M sequences.

10.2.4 Equivalence to CBR Bit Allocation

In this section, we show in the multiplexing model put forth above that the problem of optimal bit allocation for multiple VBR streams reduces to a CBR bit allocation problem for a single stream.

The multiplexing model follows the operation of the MPEG Video Buffering Verifier (VBV). We can view the buffer, the demultiplexer, and the bank of M decoders in Figure 10.5 as comprising a single VBV. The lexicographic framework of Chapter 6 can then be applied. Since the transmission channel operates at a constant bit rate, the CBR constraints of Chapter 7 would apply.

For display interval i, we need to consider the quantization scales used to code picture i of each of the M video sequences. Given a fixed bit budget for coding picture i of each video sequence, it is easy to show that the same quantization scale must be used to code picture i of all sequences to achieve lexicographic optimality; if the quantization scales differ, we can always shift bits around to reduce the highest quantization scale by increasing a lower quantization scale. This result also holds if we formulate the lexicographic criterion using vectors of quantization scales.

By using a combined bit-production model that is the sum of the bit-production models for the individual sequences, we can then allocate bits jointly to the sequences using the CBR algorithm of Chapter 7.

While the above technique guarantees that the buffer in Figure 10.5 does not overflow or underflow, it should be noted that doing so *does not* guarantee MPEG VBV compliance for the individual sequences, except when the individual VBV buffers are at least the size of the buffer in Figure 10.5. A multiplexing model that explicitly includes individual decoder buffers is certainly possible. However, analysis of this situation is not as straightforward as the above model and remains an open problem.

10.3 Bit Allocation with a Discrete Set of Quantizers

One of the assumptions made in Chapter 6 is that there is a continuous relationship between quantization (distortion) and rate. As shown in Chapters 7 and 8, this assumption facilitates rigorous analysis of the buffer-constrained bit allocation problem under the lexicographic optimality criterion and results in an elegant characterization of the optimal solution. In order to apply directly the results of the analysis, we need to construct a continuous model of the relationship between quantization and rate. As demonstrated in Chapter 9, this can be done by gathering statistics during multiple encoding passes and fitting these to a chosen functional form. Because of the inevitable error in the modeling, some form of error recovery is needed, such as the scheme proposed in Chapter 9.

In most practical coders, however, both the set of available quantizers and the number of bits produced are discrete and finite. The problem of buffer-constrained bit allocation under these conditions have been examined by Ortega, Ramchandran, and Vetterli [80]. They provide a dynamic programming algorithm to find a CBR allocation that minimizes a sum-distortion metric. In this section, we briefly describe their algorithm and show how it can be readily extended to perform lexicographic minimization.

10.3.1 Dynamic Programming

The dynamic programming algorithm described in [80] is based on the Viterbi algorithm outlined in Section 3.3.3a for solving the budget-constrained bit allocation problem. To handle the additional buffer constraints, the buffer fullness is recorded at each state instead of the total number of bits used so far; for CBR coding, the number of bits used can be determined from the buffer fullness. We can use the recurrence equations in Section 6.4.1 to update the buffer fullness and create a trellis. Instead of pruning states that exceed a given bit budget, we instead prune states that overflow or underflow the buffer. At each stage in the construction of the trellis, we compare the current sum distortion associated with edges that enter a new state and record the minimum distortion along with a pointer to the source state. At the last stage of trellis construction, we identify the state with the minimum sum distortion and backtrack through the stored pointers to recover an optimal bit allocation. Since an integral number of bits is generated, the maximum number of states that can be generated at each stage is equal to the size of the buffer. Therefore, with M quantizers, N pictures, and a buffer of size B, the dynamic programming algorithm of [80] requires $O(MBN)$ time to compute an optimal bit allocation.

10.3.2 Lexicographic Extension

It is straightforward to modify the dynamic programming algorithm of [80] to perform lexicographic minimization. Instead of keeping track of a minimum sum distortion value, a scalar, we keep track of a lexicographic minimum, a vector. A naive implementation would store a vector of length k for a state at the kth stage in the trellis, where the vector records the quantizers used for coding the first k pictures. However, since the set of quantizers is finite and we are only concerned with the number of times a given quantizer is used and not with the order in which the quantizers are used, we only need to store M values at each state, where M is the number of quantizers. Each of these M values count the number of times a given quantizer has been used to code the first k pictures in an optimal path ending at the given state. Given two vectors of quantizer counts, a lexicographic comparison can be performed in $O(M)$ time. With this modification, we can find a lexicographically optimal bit allocation in $O(M^2BN)$ time.

Bibliography

[1] V. R. Algazi, Y. Kato, M. Miyahara, and K. Kotani. Comparison of image coding techniques with a picture quality scale. In *Proceedings of SPIE, Applications of Digital Image Processing XV*, pages 396–405, San Diego, CA, July 1992.

[2] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[3] T. C. Bell and I. H. Witten. Relationship between greedy parsing and symbolwise text compression. *Journal of the ACM*, 41(4):708–724, July 1994.

[4] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards*. Kluwer Academic Publishers, Boston, MA, 1995.

[5] M. Bierling. Displacement estimation by hierarchical blockmatching. *SPIE Vol. 1001 Visual Communications and Image Processing*, pages 942–951, 1988.

[6] CCIR Recommendation 601. Encoding parameters of digital television for studios, 1982.

[7] CCITT. Description of reference model 8 (RM8), June 1989. Study Group XV—Document 525.

[8] CCITT. Video codec for audiovisual services at $p \times 64$ kbit/s, August 1990. Study Group XV—Report R 37.

[9] M. H. Chan, Y. B. Yu, and A. G. Constantinides. Variable size block matching motion compensation with applications to video coding. *IEE proceedings*, 137(4):205–212, 1990.

[10] J.-J. Chen and H.-M. Hang. A transform video coder source model and its application. In *Proceedings ICIP'94*, volume 2, pages 967–971, 1994.

[11] M. C. Chen and Jr. A. N. Willson. Rate-distortion optimal motion estimation algorithm for video coding. In *Proceedings 1996 International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages 2096–2099, Atlanta, GA, May 1996.

[12] J.-B. Cheng and H.-M. Hang. Adaptive piecewise linear bits estimation model for MPEG based video coding. In *Proceedings ICIP'95*, volume 2, pages 551–554, 1995.

[13] J. Choi and D. Park. A stable feedback control of the buffer state using the controlled lagrange multiplier method. *IEEE Transactions on Image Processing*, 3(5):546–557, September 1994.

[14] P. A. Chou, T. Lookabaugh, and R. M. Gray. Entropy-constrained vector quantization. *IEEE Transactions on Signal Processing*, 37(1):31–42, January 1989.

[15] K. W. Chun, K. W. Lim, H.D. Cho, and J. B. Ra. An adaptive perceptual quantization algorithm for video coding. *IEEE Transactions on Consumer Electronics*, 39(3):555–558, August 1993.

[16] T.-Y. Chung, K.-H. Jung, Y.-N. Oh, and D.-H. Shin. Quantization control for improvement of image quality compatible with MPEG2. *IEEE Transactions on Consumer Electronics*, 40(4):821–825, November 1994.

[17] W. C. Chung, F. Kossentini, and M. J. T. Smith. A new approach to scalable video coding. In *Proceedings 1995 Data Compression Conference*, pages 381–390, Snowbird, UT, March 1995. IEEE Computer Society Press.

[18] G. Cicalini, L Favalli, and A. Mecocci. Dynamic psychovisual bit allocation for improved quality bit rate in MPEG-2 transmission over ATM links. *Electronic Letters*, 32(4):370–371, February 1996.

[19] J. G. Cleary and I. H. Witten. A comparison of enumerative and adaptive codes. *IEEE Transactions on Information Theory*, IT–30(2):306–315, March 1984.

[20] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communication*, COMM–32(4):396–402, April 1984.

[21] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 1991.

[22] W. Ding. Joint encoder and channel rate control of VBR video over ATM networks, April 1996. preprint.

[23] W. Ding and B. Liu. Rate control of MPEG video coding and recording by rate-quantization modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(1):12–20, February 1996.

[24] F. Dufaux and M. Kunt. Multigrid block matching motion estimation with an adaptive local mesh refinement. *SPIE Vol. 1818 Visual Communications and Image Processing*, pages 97–109, 1992.

[25] H. Everett. Generalized langrange multiplier method for solving problems of optimum allocation of resources. *Operation Research*, 11:399–417, 1963.

[26] R. M. Fano. The transmission of information. Technical Report 65, Research Laboratory of Electronics, 1949.

[27] E. R. Fiala and D. H. Greene. Data compression with finite windows. *Communications of the ACM*, 32(4):490–505, April 1989.

[28] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, March 1973.

[29] J. E. Fowler and S. C. Ahalt. Differential vector quantization of real-time video. In *Proceedings 1994 Data Compression Conference*, pages 205–214, Snowbird, UT, March 1994. IEEE Computer Society Press.

[30] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Press, 1992.

[31] J. B. Ghosh. Siting facilities along a line when equity of service is desirable. *Journal of Operation Research Society*, 47(3):435–445, March 1996.

[32] P. C. Gutmann and T. C. Bell. A hybrid approach to data compression. In *Proceedings 1994 Data Compression Conference*, pages 225–233, Snowbird, UT, March 1994. IEEE Computer Society Press.

[33] A. Hartman and M. Rodeh. Optimal parsing of strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, pages 155–167. Springer-Verlag, 1985.

[34] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG–2*. Chapman & Hall, New York, NY, 1997.

[35] B. G. Haskell and A. R. Reibman. Multiplexing of variable rate encoded streams. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(4):417–424, August 1994.

[36] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[37] D. T. Hoang, P. M. Long, and J. S. Vitter. Explicit bit-minimization for motion-compensated video coding. In *Proceedings 1994 Data Compression Conference*, pages 175–184, Snowbird, UT, March 1994. IEEE Computer Society Press.

[38] D. T. Hoang, P. M. Long, and J. S. Vitter. Rate-distortion optimizations for motion estimation in low-bitrate video coding. Technical Report CS-1995-16, Duke University, Dept. of Computer Science, 1995.

[39] D. T. Hoang, P. M. Long, and J. S. Vitter. Efficient cost measures for motion compensation at low bit rates. In *Proceedings 1996 Data Compression Conference*, pages 102–111, Snowbird, Utah, March 1996.

[40] D. T. Hoang, P. M. Long, and J. S. Vitter. Rate-distortion optimizations for motion estimation in low-bit-rate video coding. In V. Bhaskaran, F. Sijstermans, and S. Panchanathan, editors, *Digital Video Compression: Algorithms and Technologies 1996*, pages 18–27, 1996. Proc. SPIE 2668.

[41] P. G. Howard and J. S. Vitter. Practical implementations of arithmetic coding. In J. A. Storer, editor, *Images and Text Compression*, pages 85–112. Kluwer Academic Publishers, Norwell, MA, 1992.

[42] P. G. Howard and J. S. Vitter. Design and analysis of fast text compression based on quasi-arithmetic coding. *Journal of Information Processing and Management*, 30(6):777–790, 1994. A shortened earlier version appears in *Proceedings 1993 Data Compression Conference*, Snowbird, UT, April 1993.

[43] C.-Y. Hsu, A. Ortega, and A. R. Reibman. Joint selection of source and channel rate for VBR video transmission under ATM policing constraints. *IEEE Journal on Selected Areas in Communications*, 1997. To appear.

[44] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098–1101, 1952.

[45] A. C. Hung. PVRG-p64 codec 1.1, 1993. URL: `ftp://havefun.stanford.edu/pub/p64`.

[46] T. Ibaraki and N. Katoh. *Resource Allocation Problems*. MIT Press, Cambridge, MA, 1988.

[47] ISO. Cd11172-2: Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbits/s, November 1991.

[48] ISO-IEC/JTC1/SC29/WG11/N0400. Test model 5, April 1993. Document AVC-491b, Version 2.

[49] ISO-IEC/JTC1/SC29/WG11/N0802. Generic coding of moving pictures and associated audio information: Video, November 1994. MPEG Draft Recommendation ITU-T H.262, ISO/IEC 13818-2.

[50] J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, 1981.

[51] D. Jamison and K. Jamison. A note on the entropy of partially-known languages. *Information and Control*, 12:164–167, 1968.

[52] N. S. Jayant, J. Johnson, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81:1385–1422, October 1993.

[53] R. S. Klein, H. Luss, and D. R. Smith. Lexicographic minimax algorithm for multiperiod resource allocation. *Mathematical Programming*, 55(2):213–234, June 1992.

[54] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[55] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion-compensated interframe coding for video conferencing. In *Proceedings IEEE National Telecommunication Conference*, volume 4, pages G5.3.1–G5.3.5, November 1981.

[56] G. G. Langdon. A note on the ziv-lempel model for compressing individual sequences. *IEEE Transactions on Information Theory*, IT–29:284–287, March 1983.

[57] D. J. LeGall. MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[58] H. Li, A. Lundmark, and R. Forchheimer. Image sequence coding at very low bitrates: A review. *IEEE Transactions on Image Processing*, 3(5):589–609, September 1994.

[59] D. W. Lin and J.-J. Chen. Efficient bit allocation under multiple constraints on cumulated rates for delayed video coding. In J. Biemond and E. J. Delp, editors, *Visual Communications and Image Processing '97*, pages 1370–1381, February 1997. Proc. SPIE 3024.

[60] F.-H. Lin and R. M. Mersereau. An optimization of MPEG to maximize subjective quality. In *Proceedings ICIP'95*, volume 2, pages 547–550, 1995.

[61] L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Gradient-based buffer control techniques for MPEG. In *Proceedings VCIP'95*, Taipei, Taiwan, May 1995.

[62] L.-J. Lin, A. Ortega, and C.-C. J. Kuo. A gradient-based rate control algorithm with applications to MPEG video. In *Proceedings ICIP'95*, Washington, D.C., October 1995.

[63] L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Cubic spline approximation of rate and distortion functions for MPEG video. In V. Bhaskaran, F. Sijstermans, and S. Panchanathan, editors, *Digital Video Compression: Algorithms and Technologies 1996*, pages 169–180, 1996. Proc. SPIE 2668.

[64] L.-J. Lin, A. Ortega, and C.-C. J. Kuo. Rate control using spline-interpolated R-D characteristics. In *Proceedings VCIP'96*, 1996.

[65] M. Liou. Overview of the $p \times 64$ kbit/s video coding standard. *Communications of the ACM*, 34(4):60–63, April 1991.

[66] M. Luptacik and F. Turnovec. Lexicographic geometric programming. *European Journal of Operational Research*, 51(2):259–269, March 1991.

[67] H. Luss and S. K. Gupta. Allocation of effort resources among competitive activities. *Operations Research*, 23:360–366, 1975.

[68] E. Marchi and J. A. Oviedo. Lexicographic optimality in the multiple objective linear programming. The nucleolar solution. *European Journal of Operational Research*, 57(3):355–359, March 1992.

[69] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and its Applications*. Academic Press, 1979.

[70] J. L. Mitchell and W. B. Pennebaker. Optimal hardware and software arithmetic coding procedures for the Q-Coder. *IBM Journal of Research and Development*, 32:727–736, November 1988.

[71] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, editors. *MPEG Video Compression Standard*. Chapman & Hall, New York, NY, 1997.

[72] A. M. Moffat. Implementing the ppm data compression scheme. *IEEE Transactions on Communication*, COMM–38:1917–1921, November 1990.

[73] D. R. Morrison. PATRICIA—A practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15:514–534, 1968.

[74] MPEG Software Simulation Group. MPEG-2 encoder/decoder version 1.1a, July 4, 1994. URL: `http://www.mpeg.org/MSSG`.

[75] Y. Nakano, H. Yahagi, Y. Okada, and S. Yoshida. Highly efficient universal coding with classifying to subdictionaries for text compression. In *Proceedings 1994 Data Compression Conference*, pages 234–243, Snowbird, UT, March 1994. IEEE Computer Society Press.

[76] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation, Compression, and Standards*. Plenum Press, second edition, 1995.

[77] W. Ogryczak. On the lexicographic minimax approach to location–allocation problems. Technical Report IS - MG 94/22, Université Libre de Bruxelles, December 1994.

[78] L. A. Olzak and J. P. Thomas. Seeing spatial patterns. In K. Boff, L. Kaufman, and J. Thomas, editors, *Handbook of Perception and Human Performance*. Wiley, 1986.

[79] A. Ortega, K. Ramchandran, and M. Vetterli. Optimal buffer-constrained source quantization and fast approximations. In *Proceedings 1992 International Symposium on Circuits and Systems*, pages 192–195, San Diego, CA, May 1992.

[80] A. Ortega, K. Ramchandran, and M. Vetterli. Optimal trellis-based buffered compression and fast approximations. *IEEE Transactions on Image Processing*, 3(1):26–40, January 1994.

[81] D. Park and K. Kim. Buffered rate-distortion control of MPEG compressed video channel for DBS applications. In *Proceedings IEEE Internation Conference on Communications*, volume 3, pages 1751–1755, 1995.

[82] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps. An overview of the basic principles of the q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726, November 1988.

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

[84] M. R. Pickering and J. F. Arnold. A perceptually efficient VBR rate control algorithm. *IEEE Transactions on Image Processing*, 3(5):527–532, September 1994.

[85] A. Premoli and W. Ukovich. Piecewise lexicographic programming. A new model for practical decision problems. *Journal of Optimization Theory and Applications*, 72(1):113–142, January 1992.

[86] A. Puri and R. Aravind. Motion-compensated video coding with adaptive perceptual quantization. *IEEE Transactions on Circuits and Systems for Video Technology*, 1(4):351–361, December 1991.

[87] A. Puri and H.-M. Hang. Adaptive schemes for motion-compensated coding. *SPIE Vol. 1001 Visual Communications and Image Processing*, pages 925–935, 1988.

[88] A. Puri, H.-M. Hang, and D. L. Schilling. An efficient block-matching algorithm for motion compensated coding. In *Proceedings 1987 International Conference on Acoustics, Speech and Signal Processing*, pages 25.4.1–25.4.4, 1987.

[89] K. Ramchandran, A. Ortega, and M. Vetterli. Bit allocation for dependent quantization with applications to MPEG video coders. In *Proceedings 1993 International Conference on Acoustics, Speech and Signal Processing*, volume 5, pages 381–384, 1993.

[90] K. Ramchandran, A. Ortega, and M. Vetterli. Bit allocation for dependent quantization with applications to multiresolution and MPEG video coders. *IEEE Transactions on Image Processing*, 3(5):533–545, September 1994.

[91] A. R. Reibman and B. G. Haskell. Constraints on variable bit-rate video for ATM networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2(4):361–372, December 1992.

[92] J. Ribas-Corbera and D. L. Neuhoff. Optimal bit allocations for lossless video coders: Motion vectors vs. difference frames. In *Proceedings ICIP'95*, volume 3, pages 180–183, 1995.

[93] J. Rissanen and G. G. Langdon. Universal modeling and coding. *IEEE Transactions on Information Theory*, IT–27:12–23, January 1981.

[94] J. D. Salehi, Z.-L. Zhang, J. F. Kurose, and D. Towsley. Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing. In *Proceedings 1996 ACM International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS)*, May 1996.

[95] G. M. Schuster and A. K. Katsaggelos. *Rate-Distortion Based Video Compression: Optimal Video Frame Compression and Object Boundary Encoding*. Kluwer Academic Publishers, Boston, MA, 1997.

[96] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[97] C. E. Shannon. Prediction and entropy of printed English. *Bell System Technical Journal*, 30:50–64, January 1951.

[98] Y. Shoham and A. Gersho. Efficient bit allocation for an arbitrary set of quantizers. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(9):1445–1453, September 1988.

[99] A M. Slyz. Image compression using a ziv-lempel type coder. Master's thesis, University of Michigan School of Engineering, 1991.

[100] R. Srinivasan and K. R. Rao. Predictive coding based on efficient motion estimation. In *Proceedings International Conference on Communications*, volume 1, pages 521–526, 1988.

[101] J. A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, New York, NY, 1988.

[102] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[103] K. M. Uz, J. M. Shapiro, and M. Czigler. Optimal bit allocation in the presence of quantizer feedback. In *Proceedings 1993 International Conference on Acoustics, Speech and Signal Processing*, volume 5, pages 385–388, 1993.

[104] E. Viscito and C. Gonzales. A video compression algorithm with adaptive bit allocation and quantization. In *SPIE Proceedings: Visual Communications and Image Processing*, volume 1605, pages 58–72, November 1991.

[105] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding.* McGraw-Hill, 1979.

[106] X. Wang, S. M. Shende, and K. Sayood. Online compression of video sequences using adaptive vq codebooks. In *Proceedings 1994 Data Compression Conference*, pages 185–194, Snowbird, UT, March 1994. IEEE Computer Society Press.

[107] T.A. Welch. A technique for high performance data compression. *Computer*, pages 8–19, 1984.

[108] S. J. P. Westen, R. L. Lagendijk, and J. Biemond. Perceptual optimization of image coding algorithms. In *Proceedings ICIP'95*, volume 2, pages 69–72, 1995.

[109] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, volume 4, pages 96–104, 1960.

[110] T. Wiegand, M. Lightstone, D. Mukherjee, T. G. Campbell, and S. K. Mitra. Rate-distortion optimized mode selection for very low bit rate video coding and the emerging H.263 standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(2):182–190, April 1996.

[111] I. H. Witten and T. C. Bell. The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, IT–37:1085–1094, July 1991.

[112] X. Zhang, M. C. Cavenor, and J. F. Arnold. Adaptive quadtree coding of motion-compensated image sequences for use on the broadband ISDN. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(3):222–229, 1993.

[113] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT–23(3):337–343, 1977.

[114] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT–24:530–536, September 1978.