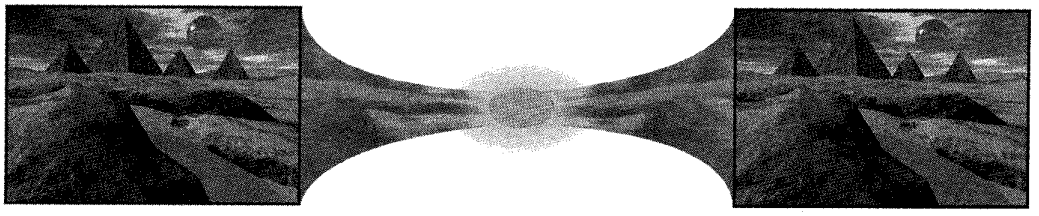
Optimal Modeling for COMPLEX SYSTEM DESIGN



Michelle Effros

Many modern engineering problems rely on some means of modeling to deal with system conditions about which there is uncertainty at design time. Data compression, speech recognition, mobile communications, and market forecasting all represent examples of problems where modeling uncertainty plays a major role. Data compression systems use algorithms designed to match the statistics of the data to be compressed; speech recognition systems incorporate models of speech patterns; mobile communications systems use models of communication channel noise characteristics; and market-forecasting programs incorporate information about market statistics.

The Lessons of Rate-Distortion Theory

In each of the above examples, statistical modeling plays a major role in the system design problem. This role is even more critical when the conditions under which the system must operate are themselves unknown or time-varying. A compression system may be used to compress a wide variety of different data types; a speech recognition system may be used to recognize speech from speakers with different voice characteristics, accents, and speech patterns; a mobile communication system may be used in a variety of physical surroundings or network traffic conditions; a financial model may be used to predict stock prices through distinct modes of market behavior; and so on. In each of these examples, and many more like them, knowledge about the range of possible working conditions is available at design time, but the actual conditions under which the system operates at any given instant are typically unavailable a priori. For example, it is not difficult to come up with an estimate of the types of images that will be sent to your printer and the relative frequencies of those data types, yet predicting the exact nature of the next image is extremely difficult since there is no natural order in which documents tend to arrive. Similarly, a reasonable model of the range of speaker characteristics observed in an automated telephone banking system may be obtained by monitoring such calls over an extended time period, but accurate prediction of the next caller's speech characteristics is much more difficult. Again, each system is characterized

1053-5888/98/$10.00©1998IEEE

by a range of possible conditions that may be reasonably understood using large quantities of past data, but each system may undergo changes that vary widely and erratically within that range of allowed behavior.

While the basic modeling problem in any of the above applications is difficult in its own right, the enormous variability resulting from changing system conditions makes the modeling task even more difficult. One approach to modeling for such complex environments is to treat model design as a two-level problem. The high-level problem is to model the range of possible system conditions. The low-level problem is to describe the data characteristics associated with any single set of system conditions. For example, in a data compression system, the high-level model might address the range of different images that may be observed within a single compression system while the low-level model might cover the modeling inherent in the compression system designed for any particular image modality. This approach effectively breaks the complex system design problem into a collection of simpler design problems by building the overall system model from a collection of simpler models.

Rate-distortion theory bounds the achievable performance of data compression systems.

The main topic of this article is the design and use of collections of models. While the low-level modeling problem is application-dependent, the optimal techniques for designing and using collections of models apply across an enormous variety of applications, and thus a single approach suffices for addressing the high-level modeling problem. This work includes a description of the multiple-model-design algorithm and discussions of the application of that technique both for the low-level data compression system design applications from which it derives and for high-level design both within data compression systems and beyond.

This article begins with a brief introduction to the theory describing optimal data compression systems and their performance. Following that introduction is a brief outline of a representative algorithm that employs these lessons for optimal data compression system design. The implications of rate-distortion theory for practical data compression system design is then described, followed by a description of the tensions between theoretical optimality and system practicality and a discussion of common tools used in current algorithms to resolve these tensions. Next, the generalization of rate-distortion principles to the design of optimal collections of models is presented. The discussion focuses initially on data compression systems, but later widens to describe how rate-distortion theory principles generalize to model design for a wide variety of modeling applications. The article ends with a discussion of the performance benefits to be achieved using the multiple-model design algorithm.

While this work focuses primarily on basic concepts rather than specific examples, a small collection of examples appears in the boxes that accompany the article. These boxes are included to give a brief look at implementation issues in data compression systems and to demonstrate that the lessons of rate-distortion theory can be applied not only to design good simple source models but also to design good quantization matrices for JPEG-style data compression systems, design transform codes with performance that far exceeds that of JPEG, and in fact to design good collections of models for a wide variety of applications where the data modeled by a system may vary over a wide range of possibilities. (The JPEG (Joint Photographic Expert Group) algorithm [1] is an image-compression standard. A brief description of JPEG appears in Box 3.)

The brief collection of examples cited in the boxes and elsewhere throughout this work should by no means be considered a general survey of the data compression literature. The limited nature and scope of this work necessitates the omission of a wide variety of works playing extremely significant roles in the fields here touched upon. The author apologizes for those omissions and encourages interested readers to seek out the many books and papers providing general literature surveys for more complete and balanced looks at these fields.

Data Compression: An Introduction

Data compression algorithms provide efficient data representations for information storage or transmission. Rate-distortion theory bounds the achievable performance of data compression systems.

Data compression, also known as source coding (e.g., image or video coding), is the science of information representation. The data compression problem may be posed in two ways. Describe data to the greatest accuracy possible within a given file size (or within a required time across a noisy communication system); or, describe data using the smallest possible file size (or transmission time) while maintaining a desired level of reproduction accuracy. In either expression of the problem, the fundamental trade-off is the same. The system trades description length or *rate* (often measured in bits per symbol from the original source alphabet; e.g., bits per pixel) for reproduction fidelity or low reproduction *distortion* (typically measured by some simple distance measure such as squared error, which is assumed throughout this work unless indicated otherwise).

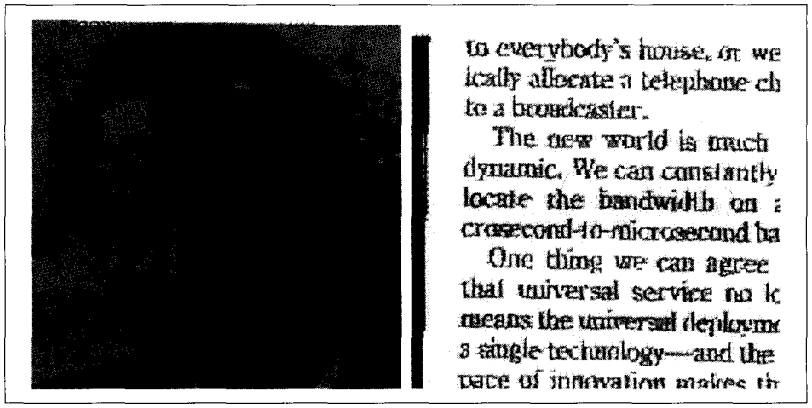
On an intuitive level, any data compression system could be imagined, as illustrated in Fig. 1, as a long list with three columns. The first column contains the row indices for book-keeping purposes. The second column contains a list of possible reproduction sequences. The

third column contains the binary descriptions used to specify a particular reproduction. In the example in Fig. 1, $i \in \{1, \ldots, 39\}$ designates the row indices, $\{\beta(i)\}$ denotes the collection of reproduction sequences, and $\{\gamma(i)\}$ specifies the associated binary descriptions. Each reproduction sequence in the second column may be either a single value (scalar) or a collection of values (vector) from the reproduction alphabet. For example, the second column in Fig. 1 contains a list of 64-dimensional vectors arranged in 8×8 squares. If each reproduction sequence is equal in dimension to a single image, then the collection of images that can be represented using the given code is identical to the collection of reproduction sequences. Typically, though, each reproduction sequence is smaller in dimension than a single image. In this case, the reproduction sequences may be strung together in any possible order to create a wide variety of different images. Two examples of images created by stringing together reproduction vectors from Fig. 1 appear in Fig. 2.

The overall compression system maps incoming data sequences into reproductions by way of their associated binary descriptions. The list of binary descriptions is chosen to be "uniquely decodable" (see, for example, [2]) so that there is no uncertainty in parsing an incoming bit stream into separated block descriptions or determining to which reproduction a given binary description must correspond. Intuitively, one may think of a uniquely decodable code as a binary tree with left branches labeled by zeros and right branches labeled by ones, as shown in Fig. 3. In this tree, each binary description represents a path starting at the root and ending at a leaf (rather than an internal node) of the binary tree. Thus, each leaf is labeled by the unique index i to which the path description $\gamma(i)$ corresponds. A binary string of coded data may be parsed by tracing the corresponding path down the binary tree. The tracing process begins at the root of the tree and follows the left branch if the next bit is a zero and the right branch if the next bit is a one. This process continues until a leaf of the tree is reached, at which point the end of a single block description is realized. The process continues at the top of the tree using the remaining bit stream. Using such a diagram, it becomes obvious that one cannot design a uniquely decodable code such that all indices have short binary descriptions. To give short binary descriptions to some indices one must give long binary descriptions to other indices in order to keep the number of leaves in the tree constant. (While it is not true that every uniquely decodable code can be described in the above manner, it is true that for any uniquely decodable code there exists a code with the same description lengths

i	$\beta(i)$	$\gamma(i)$
1		111100
2		1110110
3		111110111
4		111111011
5		11011010
6		111110010
7		11111110
8		1111111111
9		1101010
10		11110111
11		11011100
12		11101111
13		11011110
14		111111010
15		111010
16		11010111
17		11010110
18		111000
19		0
20		11110111
21		1110011
22		111111110
23		1111010
24		111110110
25		111110011
26		1101100
27		110100
28		1111111110
29		11111000
30		11111010
31		10
32		11011101
33		11111100
34		1110010
35		1100
36		110110110
37		110110111
38		11101110
39		11011111

▲ *1. A data-compression codebook. The first column contains the reproduction index i. The second column contains the (8 × 8) reproduction vector $\beta(i)$. The third column contains the binary description $\gamma(i)$.*



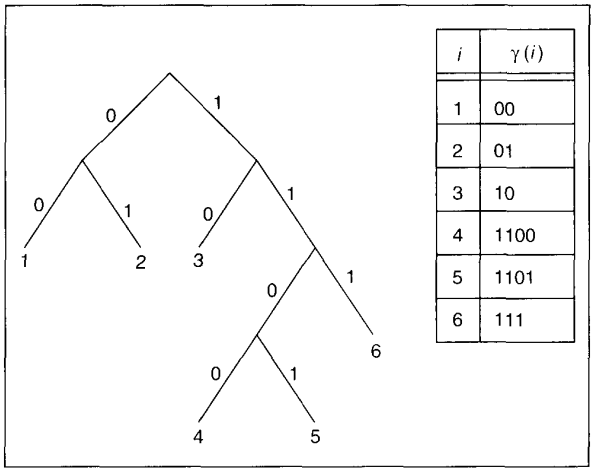
▲ *2. Two images created using the codebook in Fig. 1. (Note that the scale in these images is very different from the scale of the reproduction vectors of Fig. 1. Each codeword in the codebook is an 8 × 8 pixel block. Each of the above images has dimension 512×512 pixels.) The average rate used to describe the sequence of indices leading to the image on the left is 4.35 bits per vector or 0.07 bits per symbol. The average rate used to describe the sequence of indices leading to the image on the right is 3.92 bits per vector or 0.06 bits per symbol.*

that fits the binary-tree format described above. Thus, the general conclusion that short binary descriptions for some indices necessitate long binary descriptions for others applies to all uniquely decodable codes.)

Given the list structure of Fig. 1, compression may be achieved in two ways. First, when the list of possible reproduction sequences is shorter than the list of possible data sequences, the number of bits needed to uniquely describe a member of the reproduction "codebook" is smaller than the number of bits needed to uniquely describe all possible input sequences. For example, there exists a fixed-rate code such that any one of the $39 < 2^6$ 64-dimensional reproduction vectors in Fig. 1 may be described at a rate of 6 bits per 64-dimensional vector or $6/64 = 0.09$ bits per pixel. In contrast, describing one of the $256^{64} = 2^{512}$ 64-dimensional data vectors from reproduction alphabet $\{0,\ldots,255\}$ with a fixed-rate code requires 512 bits per 64-dimensional vector or 8 bits per pixel. Notice, however, that the savings in description length resulting from the restriction of the collection of allowed reproduction sequences comes at the expense of a decrease in reproduction accuracy. For any fixed dimension, the smaller the codebook, the smaller the collection of possible images that can be created by stringing together reproduction vectors from that codebook. This restriction or "quantization" of the space of possible reproductions may cause a loss in reproduction fidelity in describing an arbitrary input image using the given codebook.

The second possible means of achieving compression involves the careful choice of the uniquely decodable binary label for each reproduction vector. For example, using shorter binary descriptions for more probable reproduction sequences (e.g., $i = 19$) at the expense (within the uniquely decodable structure) of longer binary descriptions for seldom used reproduction sequences (e.g., $i = 28$) yields an expected description length of fewer than 0.09 bits per pixel. This process, called lossless compression, results in no loss in reproduction quality.

Viewed as in Fig. 1, the compression system is effectively a collection of very simple models. Here, each model is a single reproduction value or "codeword." Each model in the collection has associated with it a cost, where the cost of a particular model is the rate necessary to describe that model to the decoder. For most applications, it is unreasonable to assume that any *single* model will be good for all possible data sequences. However, use of a jointly designed *collection* of models and a careful mechanism for choosing among those models, can yield good average performance in representing a wide variety of possible data sequences that may be observed in the given compression system. The more models used within a given compression system, the better these models can cover the space and thus the greater the reproduction accuracy. At the same time, the more models used within a given compression system, the higher the cost in rate of using the system because the use of more models generally requires the use of more bits (on average) to describe the chosen model; hence, the fundamental trade-off between rate and distortion.



i	$\gamma(i)$
1	00
2	01
3	10
4	1100
5	1101
6	111

▲ *3. A simple uniquely decodable code, for which each binary description $\gamma(i)$ corresponds to a single leaf in the tree. Using such a code tree, a binary string may be parsed into a unique list of codeword indices by beginning at the top of the tree, sequentially traveling left for each 0 and right for each 1 until encountering a leaf, outputting the given leaf index, and then traveling back to the root of the tree and repeating the process for the remaining bit stream. For example, the binary string 011010111... corresponds to the sequence 2,3,3,6,... of indices.*

By describing, for each source distribution, the optimal trade-off between the two performance measures from which it gets its name, rate-distortion theory provides performance goals for practical compression algorithms. Yet, knowing the bounds on the performance of a data compression algorithm is not enough. The design of good codes requires knowledge of how to achieve (or at least approximate) with practical codes the performance promised by these bounds.

Ideally, rate-distortion theory would describe a design algorithm for achieving the optimal practical data compression system for any possible application. Unfortunately, rate-distortion theory falls short of this ideal since the arguments used in finding the theoretical bounds prove the existence of good codes without demonstrating how to find them or worrying about their practicality. Nonetheless, many of the lessons offered by rate-distortion theoretic arguments are invaluable in designing practical codes. In particular, careful examination of the proofs of the optimal coding bounds yields important information about the properties of optimal codes. Many of these properties are applicable in practical system designs. Application of these results yields a wide variety of algorithms for designing good collections of models to fill the intuitive list in Fig. 1.

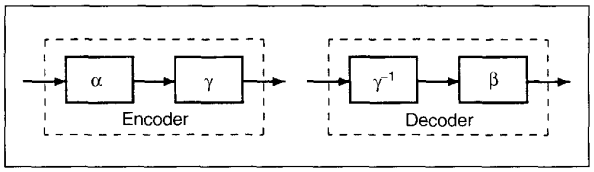
Rate-Distortion Theory: Basic Lessons

Rate-distortion theoretic proofs describe limits on the performance achievable by source coding algorithms and demonstrate the existence of codes that achieve those bounds. Careful examination of both parts of this argument yields a variety of conclusions regarding the properties of optimal codes.

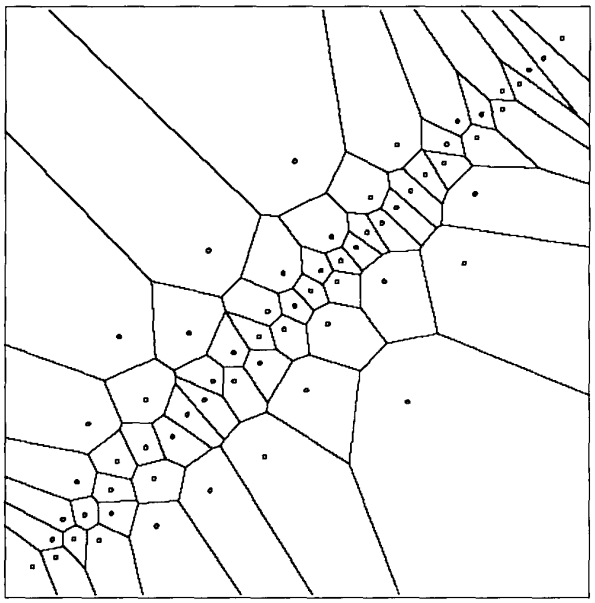
While most data compression systems are not *implemented* in the form shown in Fig. 1, most if not all practical data compression algorithms can be *described* in this format. (See Boxes 1, 2, and 3 for some simple examples of popular data compression implementations.) Use of the table in Fig. 1 requires the definition of some function that takes as its input any possible data segment and chooses for that data segment an appropriate model. The model choice is specified by its corresponding row index. Define the mapping α such that for any x^n in a fixed source alphabet $\mathcal{X}^n$ (which may be either discrete or continuous), $\alpha(x^n)$ is the row index of the model chosen for x^n. For any row index i, let $\beta(i)$ and $\gamma(i)$ denote the reproduction and binary description, respectively, found in columns 2 and 3 of row i of the table in Fig. 1. Since the binary descriptions are, by assumption, uniquely decodable, the function $\gamma^{-1}(\cdot)$ here denotes the inverse mapping from a given binary description back to the row index i to which it corresponds.

A complete compression system using the above defined functions appears in Fig. 4. The system encoder $\gamma \circ \alpha$ maps each incoming data vector x^n to a binary description $\gamma(\alpha(x^n))$ by way of the corresponding row index $\alpha(x^n)$. The length $|\gamma(\alpha(x^n))|$ of the binary description denotes the cost in rate of describing vector x^n with the given encoder. The decoder $\beta \circ \gamma^{-1}$ maps the binary sequence $\gamma(\alpha(x^n))$ to its corresponding reproduction $\beta(\alpha(x^n))$ by way of the index $\gamma^{-1}(\gamma(\alpha(x^n))) = \alpha(x^n)$. Given some distortion measure $d(x, \hat{x})$, the reproduction fidelity can be measured as $d(x^n, \beta(\alpha(x^n)))$, where $d(x^n, \hat{x}^n) = \sum_{i=1}^{n} d(x_i, \hat{x}_i)$.

The innermost segment of the compression system, here denoted by γ and γ^{-1}, is a lossless code, which maps source indices to and from their binary strings. The outermost segment of the compression system, here denoted by α and β, and henceforth referred to as a *quantizer*, breaks the space of possible data sequences into disjoint subsets or *quanta*, representing each region by a single reproduction value. An example of a quantizer encoder and decoder at $n = 2$ appears in Fig. 5. In this figure, the outer square represents the n-dimensional space of possible data vectors. That space is divided into subsets, where each subset is the set of x^n values mapped to a particular row index i by the encoder α. Each region contains a point representing the n-dimensional reproduction vec-



▲ *4. The functional data-compression system.*



▲ *5. Sample α and β functions. The encoder α carves the space of possible data vectors into subsets, each of which carries a unique index. The decoder β chooses a representative value for each region. The line segments in the above drawing mark the region boundaries. The dot in each region shows the associated representative value.*

tor $\beta(i)$ (described in the second column of Fig. 1) associated with the given region.

While rate-distortion theory does not provide an optimal joint design algorithm for α, β, and γ nor describe how to implement these functions for practical applications, it does yield a variety of lessons about the optimal system. Discussions of a few of those lessons appear in the subsections that follow.

Lesson 1: Bigger is Better: The High-Dimension Advantage

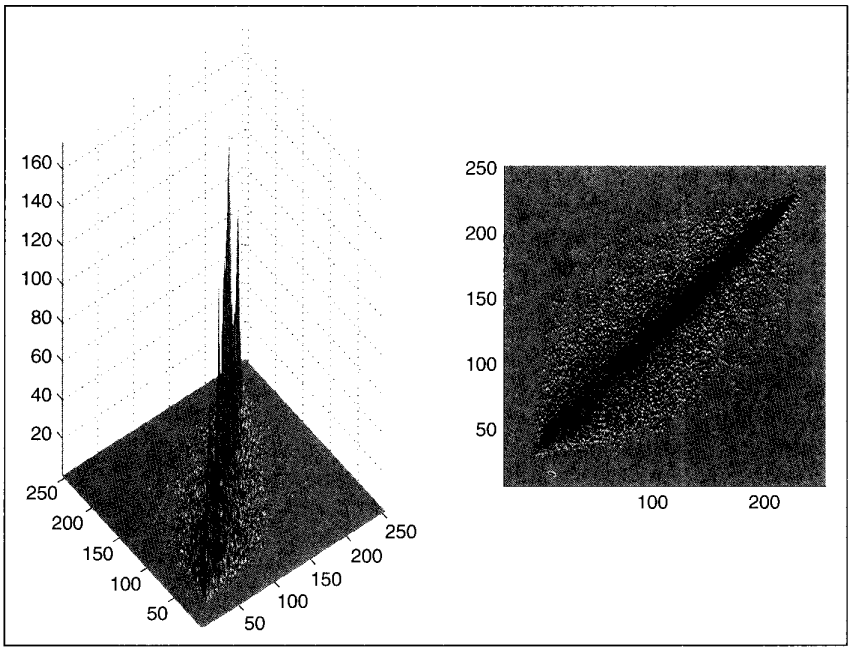
"It is simpler to describe an elephant and a chicken with one description than to describe each alone." —Cover and Thomas [2]

Perhaps the most pervasive lesson of rate-distortion theory is that joint descriptions are more efficient than separate descriptions, even for uncorrelated random variables. In the terminology of the earlier defined system, this means that the performance achievable at a higher dimension n exceeds the performance achievable at lower dimensions. Two main factors, described next, contribute to this effect [3].

The first and perhaps easier to understand is that a compression system that deals with data in larger "chunks" (or vectors) can take better advantage of correlation between samples than can a compression system that deals with data in smaller pieces. To illustrate this effect, consider a typical natural image. The given image, a photograph, is broken into vectors of dimension two (that is, each pixel is paired with a single neighbor). Figure 6 shows, from two different angles, the histogram of two-dimensional vectors from that image. The vast majority of the pixel values in this example line up along the $x_1 = x_2$ line. Many varieties of images incorporate extremely high correlation between neighboring pixel values like that seen in Fig. 6. In this case, knowing the value of one of the two pixels predicts, with very high accuracy, the value of the neighboring pixel. A compression system that treats each pixel separately cannot take advantage of this correlation.

The second key advantage of higher dimensional codes results from the economies that come with scale. Even in cases of independent experiments it is far easier to predict the typical behavior in a long sequence of experiments(x^n for large values of n) than it is to predict the outcome of any single experiment. For example, consider n independent tosses of a coin for which the probability of heads is some known constant $0 < p < 0.5$. For n small, prediction of the outcome of the experiment is difficult. For example, consider the prediction of the outcome for any single experiment ($n = 1$). Since the probability p of a head is given to be less than 0.5, the best possible guess is that the outcome will be tails. This guess is best every time the game is played even though tails might be only slightly more probable than heads and, for any $p > 0$, the event that *every* toss yields tails is unlikely. To quantify this poor performance, note that this best guess is expected to be wrong proportion p of the time. For n large, however, more accurate predictions about the expected outcomes of the n coin toss experiment are possible. In this case, (by the weak law of large numbers) roughly proportion p of the tosses are expected to give heads most of the time. Using this as a guess for the experimental outcome yields an expected probability of error that shrinks to zero as n grows without bound. The predictability of long sequences of experiments yields efficient coding schemes that give short descriptions to all possible strings with roughly proportion p of heads at the expense of long descriptions for strings with any other proportion of heads. The result is a lossless coding scheme that gives good performance for high-dimensional data (see, for example, [2]).

The above example illustrates the high-dimensional advantage in lossless coding. A similar effect can be seen in quantization since the encoding cells defined by α pack better in higher dimensions than in lower dimensions [4]. This observation is illustrated by Fig. 7. The figure contains two-dimensional drawings of the optimal cell shape for coding uniformly distributed data at $n = 1$ and the optimal cell shape for coding uniformly distributed data at $n = 2$. The drawing on the left shows the optimal cell shape associated with $n = 1$. In this case, the encoder is forced to make independent decisions in dimensions one and two.



▲ 6. *Histogram of two-dimensional pixel vector values for a photographic image.*

The resulting cells are square. Making optimal decisions in two-dimensional space yields hexagonal encoding regions. These regions are superior to the square encoding regions found on the left in the sense that uniformly distributed data over a hexagonal cell yields lower expected distortion with respect to the cell center than does uniformly distributed data over a square cell of the same area with respect to its cell center. This result is intuitively clear because the hexagon is more similar to a circle than is the square. Going to higher and higher dimensional spaces yields cell shapes that are more and more "spherical." That is, higher-dimensional spaces admit more efficient packings than do lower-dimensional spaces.

Lesson 2: The Optimal System has Optimal Parts

A source code contains three components, the quantizer encoder α, the quantizer decoder β, and the lossless code γ. The goal of data compression system design is to choose α, β, and γ to minimize the expected distortion subject to a constraint on the expected rate (or vice versa) in coding samples from a *known source distribution*. For practical applications, knowledge about the source distribution is typically embodied in a single, fixed training set of data. Thus, the optimal system is the system (of a given dimension) that achieves the best possible performance on that training set. To avoid overtraining problems, systems are typically trained and tested on nonoverlapping data sets.

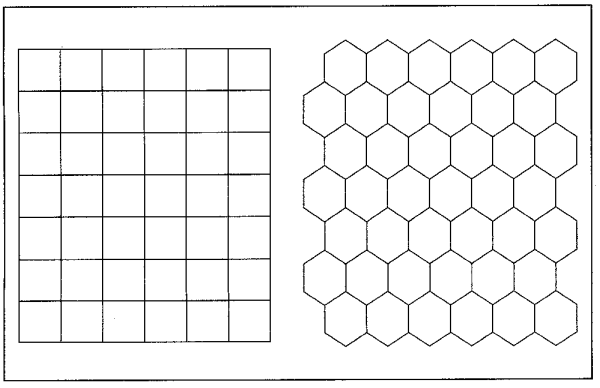
For the system as a whole to be optimal, it is necessary (though not sufficient) that α, β, and γ themselves be optimal. While rate-distortion theory does not provide a single algorithm for simultaneously optimizing the triplet (α, β, γ), it does describe the optimal α for a given β and γ, the optimal β for a given α and γ, and the optimal γ for a given α and β.

Lesson 2.1: The Optimal Quantizer Encoder α

Given β and γ functions, the optimal quantizer encoder α is the quantizer encoder that minimizes the system's expected distortion subject to a constraint on the expected rate (or equivalently minimizes the expected rate subject to a constraint on the expected distortion). This minimization is achieved by mapping every input vector x^n to the reproduction index i that yields the best performance for that particular input. Thus, denoting the optimal value of α by α^* and using a Lagrangian minimization [5] with non-negative Lagrangian constant λ,

$$\alpha^*(x^n) = \arg\min_i [d(x^n, \beta(i)) + \lambda |\gamma(i)|].$$

The Lagrangian approach minimizes the expected distortion subject to a constraint on the expected rate. The Lagrangian constant λ, which may be thought of as a descriptor of the relative importance of minimizing the expected rate versus minimizing the expected distortion, is chosen as a function of the system's target rate [5].



▲ 7. Optimal one-dimensional and two-dimensional packing.

Perhaps the most pervasive lesson of rate-distortion theory is that joint descriptions are more efficient than separate descriptions.

Lesson 2.2: The Optimal Quantizer Decoder β

Given α and γ functions, the optimal quantizer decoder β is the decoder that achieves the best possible expected distortion. This optimal performance is achieved through independent optimization of the conditional expected distortions conditioned on every possible value of $\alpha(x^n)$. That is, for each index i,

$$\begin{aligned}\beta^*(i) &= \arg\min_{\hat{x}^n} E[d(X^n, \hat{x}^n) + \lambda|\gamma(i)| \,|\, \alpha(X^n) = i] \\ &= \arg\min_{\hat{x}^n} E[d(X^n, \hat{x}^n) \,|\, \alpha(X^n) = i].\end{aligned}$$

For example, use of the squared-error distortion measure $d(x, \hat{x}) = (x - \hat{x})^2$ results in optimal decoder models (or codewords) $\{\beta(i)\}$ at the centroids of their encoding regions. That is, $\beta^*(i) = E[X^n | \alpha(X^n) = i]$ for the squared-error distortion measure.

Lesson 2.3: The Optimal Lossless Code γ

Results on the optimal lossless codes appear in the information theory literature. In particular, an optimal lossless code satisfies

$$|\gamma^*(i)| = -\log \Pr(\alpha(X^n) = i)$$

for each i. Intuitively, one can understand this result as follows. Given an alphabet with K equiprobable symbols, the best that one could hope to do in describing an outcome from this alphabet would be to describe that alphabet with $\log K$ bits, since $\log K$ bits suffice for describing $2^{\log K}$ different outcomes (throughout, $\log(x) = \log_2(x)$). There is nothing to be gained in this scenario from giving

shorter descriptions to some symbols than to others since all symbols are, by assumption, equiprobable. Now consider a source alphabet $\mathcal{X}$ with symbol probabilities $\{p(x): x \in \mathcal{X}\}$ such that $p(x)$ is *not* necessarily equal to a constant for all x. For any $x \in \mathcal{X}$, symbol x effectively looks like one of $1 / p(x)$ equiprobable outcomes. For example, a symbol with probability 1/8 in some sense looks like one of eight equiprobable outcomes. Since $\log 8 = 3$ bits is the number of bits required to describe any one out of eight equiprobable outcomes, the expectation is that $\log(1 / p(x)) = -\log(p(x))$ bits should again suffice for describing x in this scenario, even though the number of source symbols in $\mathcal{X}$ may be larger or smaller than eight and the symbols are not necessarily equiprobable.

Notice that the above formula often cannot be achieved. For example, $-\log \Pr(\alpha(X^n) = i)$ may not be an integer. Fortunately, for large values of n (see Lesson 1), the difference between the "optimal" per symbol description length $-(1/n)\log \Pr(\alpha(X^n) = i)$ and the best achievable per symbol description length differ by very little. In particular, for any distribution $\Pr(\cdot)$, there exists a uniquely decodable code with description lengths $|\gamma^*(i)| \le \lceil -\log \Pr(\alpha(X^n) = i) \rceil$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x (see, for example, [2]). The rate cost of using the above code rather than the "optimal" code is at most one bit per x^n vector or $1/n$ bits per symbol, which is arbitrarily small for large enough n.

Lesson 3: Optimal Parts Do Not Guarantee an Optimal System

The conditions described in the above lessons are necessary but not sufficient for optimality.

While Lesson 1 gives some insight into the choice of the optimal vector dimension and Lesson 2 describes the optimal α for a given β and γ, the optimal β for a given α and γ, and the optimal γ for a given α and β, realization of an optimal system is still far from attained. First, the above conditions are necessary for optimality but are generally not sufficient. Second, Lesson 1 implies that bigger is better when it comes to coding dimension. Thus, for any dimension n, the potential for improvement by going to a larger value of n always exists, resulting in a push for higher and higher dimensions (and, as discussed later, less and less practical codes). Third, given two optimal system parts, Lesson 2 describes techniques for designing the third element to be jointly optimal with the other two, but where should the two other parts come from originally? This is the global design problem treated in the next section. The question of coding dimension is considered in the section titled "Trouble in Asymptopia."

Get It Together: Lessons In Global Design

The above rate-distortion theory lessons go a long way towards describing the optimal code. The question that remains is how to jointly design the system components to simultaneously achieve all of the above conditions.

While a wide variety of (implementation dependent) code-design algorithms exist in the literature, only the simplest and most direct method for applying the results of Lesson 2 are discussed in this section. The resulting design algorithm for fixed-rate codes (codes for which $|\gamma(i)|$ is forced to equal a constant value for all i) appears in [6]. A further generalized variation for variable-rate codes comes from [5]. Both are instances of the generalized Lloyd algorithm.

The generalized Lloyd algorithm is a code-design algorithm typically run on a training set of data determined by the system designers to be representative of the data to be coded by the desired system. The design procedure, performed off-line during system design, generates a quantizer α, β and lossless code γ to match the training data. The design begins by choosing an arbitrary initial codebook of a desired dimension and maximal codebook size and an initial lossless code. For example, a codebook with index set $\{1, \ldots, 256\}$ to code four-dimensional vectors for a grey-scale image database may be initialized at random or through a variety of simple splitting techniques (see, for example, [7]). The lossless coder is typically initialized using a fixed-rate code. That is, all binary descriptions $\{\gamma(i)\}$ in the given collection are initialized to the natural fixed-length binary representations of the associated integer indices $\{i\}$. This initialization is followed by an iterative design procedure with the following three steps.

1. Optimize the quantizer encoder α for the given quantizer decoder β and lossless code γ. This step is accomplished using the optimal encoder design of Lesson 2.1.

2. Optimize the quantizer decoder β for the given quantizer encoder α and lossless code γ. This step is accomplished using the optimal decoder design of Lesson 2.2.

3. Optimize the lossless code γ for the given quantizer encoder α and quantizer decoder β. For a fixed-rate code, this step requires no change. For a variable-rate code, this step is accomplished using a lossless code-design algorithm that achieves the rates given in Lesson 2.3 (to within one bit per vector).

The process iterates until convergence.

At each step of the above iterative design procedure, the system's expected Lagrangian distortion $E[d(X^n, \beta(\alpha(X^n))) + \lambda|\gamma(\alpha(X^n))|]$ either decreases or remains constant. Since the Lagrangian distortion is bounded below by zero, the iterative procedure is guaranteed to converge. Running the above procedure to convergence typically yields a system that simultaneously satisfies all three optimal code properties. (Strictly speaking, while convergence of the Lagrangian distortion is guaranteed, convergence of the functions α, β, and γ is not. This observation is of little consequence, though, since two codes with the same Lagrangian distortion are functionally equivalent from a rate-distortion perspective.) Unfortunately, the conditions described are necessary but not sufficient for system optimality.

Rate-distortion theory is not the theory of optimal *practical* code design.

Nonetheless, since the above procedure finds, at each step, a globally optimal solution to its subsystem design problem, the entire process guarantees convergence to a locally optimal solution.

Notice that the generalized Lloyd algorithm solves a number of very interesting and important problems. In particular, the procedure answers three questions.

▲ **How many codewords are needed?**

The number of codewords needed in a fixed-rate code is trivially determined; a fixed-rate-R (bits per symbol) source code with dimension n, requires 2^{nR} n-dimensional codewords.

The same question is more interesting in the case of variable-rate codes. In this case, the number of codewords needed could, conceivably, be arbitrarily large. The question, then, is how many codewords are needed to optimally cover the space of data vectors given a target per symbol bit budget.

When the optimal codebook size is finite, the question of codebook size is solved by the iterative design procedure. That is, assuming that the optimal number of codewords is finite and that the codebook is initialized with some number of codewords greater than the number needed, the convergence algorithm effectively disposes of all unnecessary codewords. This codeword removal procedure is accomplished as a natural outcome of the lossless code design. In particular, extra codewords attract fewer and fewer training samples through the iterative procedure. As a codeword's probability decreases, its binary description length increases as the negative logarithm of the codeword probability, thereby increasing the effective cost associated with choosing the given codeword. This increase in cost causes a further decrease in probability and so on. The eventual outcome of this process (neglecting the effects of local minima problems) is that the probability of unnecessary codewords goes to zero and thus their cost in rate goes to infinity, resulting in an effective removal of all "extra" codewords from the coding system.

▲ **Which codewords should be used?**

The codeword locations are given by the optimal model equations described in Lesson 2.2. For the squared-error distortion measure, these optimal codewords are easily calculated as conditional expectations with respect to the underlying distribution (or averages over appropriate subsets of the training data).

▲ **When should each codeword be used?**

The codeword choice problem is solved by the optimal encoder described in Lesson 2.1. Implementations of truly optimal encoders typically involve a full search for each incoming data vector over the collection of possible codewords. Full search codes of this type, called vector quantizers, are described in greater detail in Box 1.

Trouble In Asymptopia: The Cost of the High-Dimension Advantage

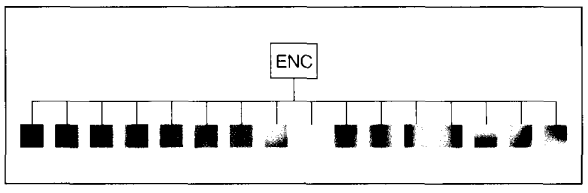
According to rate-distortion theory, the existence of the optimal code is only guaranteed asymptotically as the coding dimension goes to infinity. Yet with the rate-distortion performance gains of high-dimensional codes come a number of pitfalls that make the theoretical optimality of infinite-dimensional coding a practical impossibility.

Lesson 1 motivates the push for very high-dimensional codes. Yet, increasing the dimension n of an optimal vector-based source code increases the complexity, delay, and memory requirements of that code. Given a target rate of R bits per symbol, the number of reproduction vectors in the given code grows at least as quickly as 2^{nR}. Since the optimal encoder α is typically implemented by

Box 1 - Implementation Examples: VQ [5, 6]

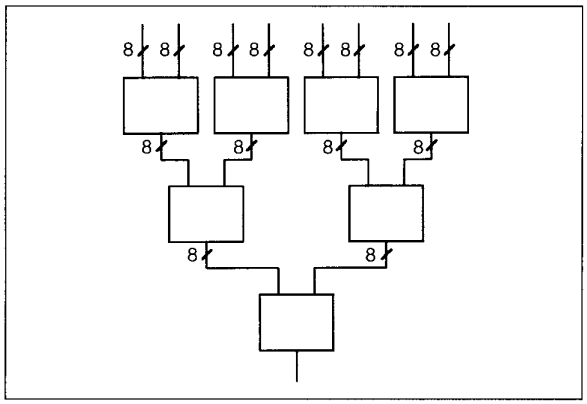
The conceptually simplest form of quantizer encoder explicitly implements the optimal encoder described in Lesson 2.1. That is, given a quantizer decoder β and lossless code γ, the encoder maps each incoming vector x^n to the index associated with the closest codeword: $\alpha^*(x^n) = \arg\min[d(x^n, \beta(i)) + \lambda|\gamma(i)|]$. An example of such an encoder is shown below for a 16 codeword system with vector dimension $n = 64$.

The encoding function is implemented using a search over all possible indices in the code's collection. That is, the encoder calculates $d(x^n, \beta(i)) + \lambda|\gamma(i)|$ for every i in the code's index set, and chooses that i corresponding to the lowest weighted distortion measure. The resulting code is called a full search vector quantizer, or simply a VQ. The VQ's decoder is a simple table lookup. The code table (more typically known as a codebook) stores one reproduction vector for each index in the code's collection.



Box 2 - Implementation Examples: Hierarchical VQ [10]

Box 1 includes a description of the traditional VQ encoder and decoder. In that case, the decoder, implemented as a simple table look-up, requires very low complexity, but the encoder, which implements a full search over all possible models in the collection, is much more computationally expensive. While high encoder complexity is acceptable for many applications (e.g., video on demand, where good system performance does not imply a requirement that the encoder be either inexpensive or fast), other applications require both the decoder and the encoder to be fast and inexpensive. For example, in videoconferencing applications, both parties in a conversation need to encode outgoing messages and decode incoming messages simultaneously and with very low delay.



In [10], Vishwanath and Chou describe a simple (suboptimal) vector quantizer that replaces the full search of the optimal VQ's encoder with a sequence of table look-ups. The observation made in this case is that for any digital system, the number of possible input vectors is finite. As a result, one possible approach for encoding data vectors is to devise a table including all possible input vectors and their corresponding optimal codeword indices. Given such a table, the encoder would require a single table look-up, which is fast and inexpensive, rather than a sequence of distortion calculations, which carry a much higher computational cost. Unfortunately, the table size required in such a system would grow exponentially with the vector dimension. As a result, Vishwanath and Chou replace the above single table look-up with a sequence of table look-ups. The resulting system contains a collection of small, simple tables, where each table takes as its inputs two 8-bit values and outputs one of 256 possible indices. The tables in the system are arranged in a simple hierarchy, shown at the left for an eight-dimensional ($n = 8$) vector quantizer.

At the top level, each table takes in the indices of two input symbols (e.g., pixel values) and outputs a single index. The indices from each neighboring pair of tables are then fed into tables at the next level, and so on until all of the pixels are represented in the final table look-up decision. While a single (enormous) table could achieve optimal encoding, the above sequence of table look-ups cannot guarantee encoder optimality. (See [10] for a discussion of hierarchical VQ table design and performance.)

way of a brute-force comparison of every incoming data vector to every possible reproduction codeword (see Box 1), exponential increases in codebook size result in exponential increases in encoder complexity. Similarly, the memory required for storing the optimal decoder $\{\beta(i)\}$ is proportional to the product of the code's size and dimension. Thus, exponential growth in source code size likewise results in exponential growth in memory requirements for storing the appropriate table of code vectors. Finally, the delay associated with block source codes of the type considered in this work grows linearly in the coding dimension. That is, since incoming data streams are described in fixed-length vectors of length n, the encoder must read in n source symbols before it can begin describing even the first symbol in that vector. While this delay may be inconsequential for a source code used in a storage system, the same delay may be prohibitive for applications such as real-time video conferencing.

The Power of Imperfection: Optimality-Performance Trade-Offs in Source Code Design

Giving up *optimal* encoders and decoders for ones that are merely *good* (and clever!) yields much of the high-dimension advantage without the computation, memory, and delay costs associated with high-dimensional codes.

Given the trade-off between rate-distortion performance and the associated cost in complexity, delay, and memory, the problem of source code design at first appears to be merely a matter of choosing the optimal coding dimension. After all, rate-distortion theory provides methods for searching for the optimal source code for any dimension, and thus it seems that the only remaining question is how to find the maximal dimension for a desired complexity (or the minimal complexity for a target performance).

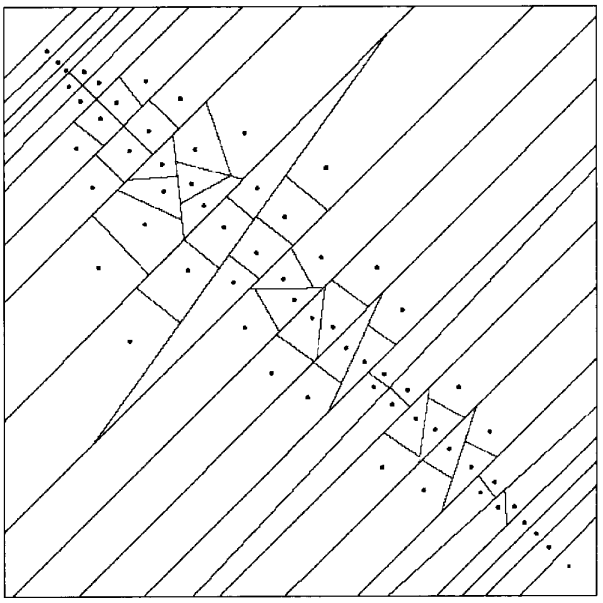
Nonetheless, a closer look reveals a tantalizing observation: "optimality" is not always the best choice. In other words, while the optimal low-dimensional code is guaranteed by definition to achieve better rate-distortion performance than any other code at that dimension, it is not guaranteed to achieve better performance than even a mediocre code at a higher dimension. The vast majority of source coding algorithms can be described as attempts at finding suboptimal (in a rate-distortion sense) high-dimensional codes that give the best possible performance for a given allowed level of complexity.

The above discussion of the power of suboptimal codes leads to one inevitable conclusion. Rate-distortion theory is not the theory of optimal *practical* code design.

In other words, the definition of optimality provided by rate-distortion theory is not the definition of optimality needed for practical coding, which requires further constraints. The true goal of system design is to minimize distortion subject to constraints on *rate and complexity and memory and delay* (or to minimize rate subject to constraints on distortion, complexity, memory, and delay, or to minimize complexity subject to constraints on rate, distortion, memory, and delay, or ...). Unfortunately, this minimization is much more difficult to solve explicitly, and thus the practical source-coding literature is full of algorithmic attempts, none of which is known to be the one true answer to the rate-distortion-complexity-memory-delay question.

While the number and variety of source-coding alternatives is almost limitless, a few basic tools represent common threads through many of these algorithms. Three of the most basic underlying strategies are considered next. Two algorithms employing these strategies are described in Boxes 2 and 3.

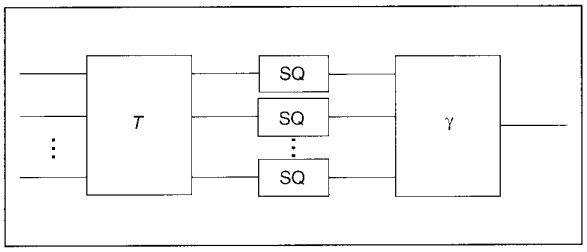
It is important to remember that the (rate-distortion) performance of a source code at any dimension n cannot be better than the performance of the optimal n-dimensional source codes described earlier. Thus, all codes designed to achieve the high-dimension advantage must themselves be high-dimensional codes. The goal here is to design high-dimensional codes that are easier to implement and store than their optimal counterparts but achieve performance that is almost as good (or at least better than the performance of optimal low-dimensional codes of the same complexity). In fact, the design of many of these suboptimal systems includes a rate-distortion optimization subject to the constraints imposed by the suboptimal coding structure (see, for example, [8] and [9]). Descriptions of three basic techniques for tackling this problem follow.



▲ *8. Tree-structured vector quantizer encoder and decoder. The above figure shows the encoding regions and corresponding decoder codewords associated with a depth-6 tree-structured vector quantizer. The given code has the same (fixed) rate and was designed on the same training set as the corresponding encoder and decoder of Fig. 5. The encoder and decoder shown above are designed to give the best possible expected distortion using a sequence of six binary comparisons (rather than the 64 separate comparisons of the full-search example) for each incoming data vector. While the resulting code gives a 23% increase in distortion relative to the full search code (on the same training set), the distortion increase is accompanied by a 90% savings in computational complexity.*

Box 3 - Implementation Examples: JPEG and Other Transform Codes [1]

Transform coding is another example of a suboptimal high-dimensional coding strategy that achieves performance far exceeding that of low-dimensional optimal codes of similar complexity. The encoder for a transform code is shown below. In transform coding, a high-dimensional vector x^n is sent through a transform prior to coding. (In the JPEG image coding standard, the vector dimension $n = 64$ and the transform, indicated below by the symbol T, is a discrete cosine transform (DCT).) The function of the transform is to decorrelate the data prior to scalar encoding. The resulting transform coefficients are then quantized using an appropriate bit allocation and a collection of uniform scalar quantizers (here denoted SQ). The bit allocation for the JPEG algorithm is specified in the form of a "quantization matrix," which is described in the file header and thus may be changed from image to image. Many implementations of JPEG, however, fix the quantization matrix to be constant over all images, and simply scale that matrix up and down to achieve a variety of rates. The transform decoder reverses the operations of the encoder, first reversing the lossless code, then reconstructing the transform coefficients using the appropriate set of scalar quantizers, and finally performing an inverse transform on the data set.



The overall effect of the system is illustrated in Fig. 9, where decorrelating the data prior to coding yields a system that approximates the optimal quantizer while maintaining very low complexity.

▲ Fast Search

The complexity requirements of the high-dimensional source codes described earlier in this work and in Box 1 are primarily borne by the quantizer encoder α, which finds the closest codeword to any incoming data vector by computing the distance to all candidates and then choosing the one with the best performance. The search for the "closest" codeword can be simplified (at the cost of optimality) using a variety of fast search techniques. In tree-structured codes (see, for example, [8]), a sequence of binary searches takes the place of the search over all possible codewords. The result is a suboptimal partitioning of the space of possible data vectors (see Fig. 8 for an example) in exchange for a search complexity that grows linearly rather than exponentially in the coding dimension. Another fast-search technique, used in the hierarchical table-lookup vector quantizer of [10], replaces all codeword comparisons with a sequence of table look-ups for extremely fast searches. This technique is described in greater detail in Box 2.

▲ Transformation

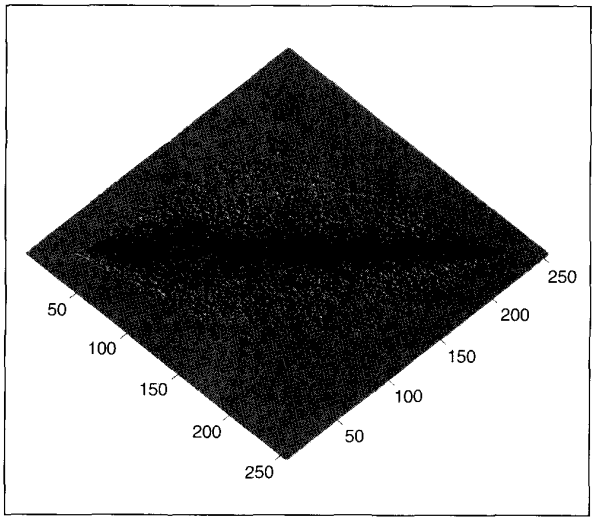
As discussed in Lesson 1, one key factor contributing to the high-dimension advantage is the ability of high-dimensional codes to recognize and thus capitalize on sample correlations. The quantizer illustrated in Fig. 5 was designed on the data whose histogram appears in Fig. 6. Clearly, sending each sample through a separate scalar quantizer, which must necessarily lead to a square grid of reproduction values and encoding cells in two-dimensional space, would result in performance inferior to the performance of the code in Fig. 5, which places most of its codewords on and around the $x_1 = x_2$ line.

Imagine instead rotating the given data set 45° prior to coding, as shown in Fig. 9. In this case, while a square grid could not perfectly replicate the performance of the two-dimensional code, the best possible square grid of reproduction values and encoding cells would achieve far superior performance to that achieved prior to rotation. This is precisely the justification for transform codes (see Box 3), which use reversible decorrelating data transformations followed by low-dimensional source coding of the decorrelated samples. The tool of bit allocation enhances the benefit reaped through the transformation step.

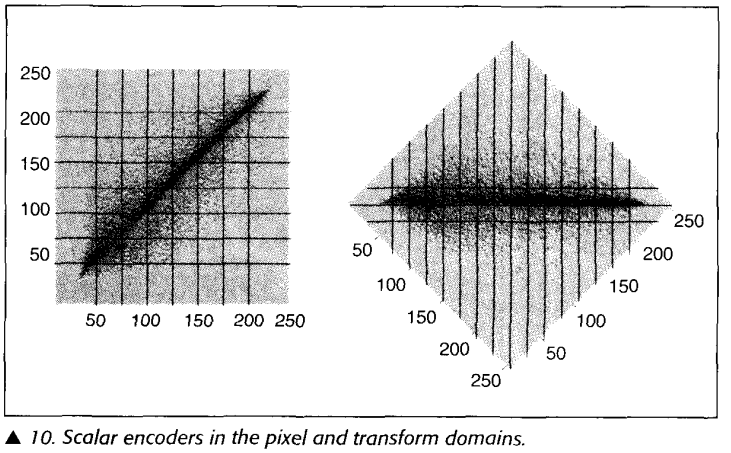
▲ Bit allocation

While there is no reason to expect energy variation as a function of pixel location in an image source (e.g., there is no reason to believe that $E[(X_1)^2]$ should exceed $E[(X_2)^2]$ or vice versa in the two-dimensional data source of Figure 6), the same cannot be said about transform coefficients. For example, rotating the two-dimensional space X^2 of Fig. 6 by 45° clockwise as shown in Fig. 9 yields one coefficient (on the new horizontal axis) representing the average value of the pixels (a "low frequency" coefficient) and one coefficient (on the new vertical axis) representing the difference between the two pixels (a "high-frequency" coefficient). Since a wide variety of images contain primarily low-frequency information, it is reasonable to expect differing amounts of energy in the two coefficients in this frequency-like representation. Clearly this expectation is met by the example in Fig. 9. Thus, while a square grid in the new coordinates is expected to yield superior performance to a square grid in the old coordinates (see Fig. 10), the optimal pair of scalar quantizers should dedicate more reproduction values and higher rate to the first dimension than to the second. The process of allocating differing number of bits to different coefficients is known as bit allocation and is a tool common to most transform-based coding schemes, as described briefly in Box 3.

Many source coding algorithms achieve poor performance on inhomogeneous data sets.



▲ *9. Rotated histogram of samples from Fig. 6.*



▲ *10. Scalar encoders in the pixel and transform domains.*

Take It To The Next Level: Source-Independent Source Coding Systems

Many source coding algorithms achieve poor performance on inhomogeneous data sets. This problem can be addressed through high-level application of the rate-distortion theory lessons discussed earlier.

The basic *techniques* of data compression show surprisingly little variation from data type to data type and application to application. In fact, the vast majority of modern compression algorithms can be described as combinations and variations on the themes described in the preceding sections. These basic tools, then, are extremely broad in their applicability. Nonetheless, special care must be taken in the use of these techniques when designing systems for which the data characteristics are not homogeneous. (A short discussion on homogeneous and inhomogeneous data characteristics appears in Box 4.)

All of the codes described in the previous sections are source dependent. For example, the optimal quantizer β and lossless code γ of Lesson 2 depend explicitly on the source distribution. (In contrast, the optimal quantizer encoder α depends on the source statistics only through its dependence on the quantizer decoder and the lossless code.) Designing a source code to match the statistics of a

Box 4 - Source Statistics: Issues and Ideas

Data compression system design is effectively a statistical modeling problem. The goal is to come up with a good collection of models to cover the space of possible observations. A good model of the source data accurately describes its statistical properties—specifying which possible "events" are more probable and which are less probable. While picturing this distribution might be difficult, an intuitive understanding of its existence is not. For example, there are many characteristics shared by x-rays. While you might have trouble describing those characteristics in the form of a distribution, you likely would not have trouble distinguishing x-rays from photographs or text.

The most common assumption in source modeling is the assumption of "stationarity." Roughly speaking, a source is stationary if *before seeing any part of that data set* the probabilistic description of likely and unlikely events is space invariant. Given a data set in which the left edge of each image is typically brighter than the right edge, then the class of images is not stationary. If no such space-dependent statement can be made a priori, then stationarity is a reasonable modeling assumption.

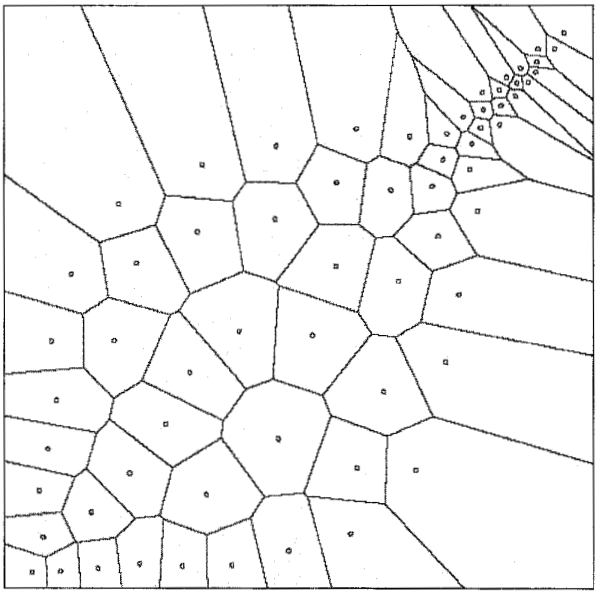
The "memory" of a distribution refers to the dependencies between individual pixels—that is, how your prediction for the future depends on your knowledge of what happened in the past. Source memory can account for the substantial differences in the characteristics of different subsets of an image. Due to the "spatially varying" look of such images, the concept of source memory is often mistakenly treated as a symptom of source nonstationarity. While both properties may be exploited in source coding, the two concepts should not be confused. Nonstationarity refers to an *a priori* spatial dependence of your expectations about an image. Source memory accounts for variation in your expectations regarding a portion of the image that you haven't seen as a function of the portion that you have seen.

For many applications, the source statistics seem to arise from a variety of distinct modes. For example, an image might be either an x-ray or a photograph or a schematic. In this case, finding a single distribution that models all possible outcomes is difficult. Instead, it is easier to think of data sets of this type as arising from a family of distributions rather than any single distribution. The family itself is governed by a top level distribution, which describes the probability of each its members. For example, it might be difficult, in designing a compression system for a home entertainment system, to come up with a *single* distribution that models faces, landscapes, text, computer graphics, animation, and so on. On the other hand, coming up with a model just for text is less complex. In this case, the larger modeling problem may be broken into a collection of smaller problems that replaces the single difficult design problem with a collection of simpler design problems.

Given this idea of a collection of distributions, one could imagine two types of sources. In one type, all samples are drawn from a single distribution. That is, every image has the same characteristics as every other image, and thus a single model suffices to describe any image in the system. Roughly speaking, this is the definition of an *ergodic source*. A data set drawn from an ergodic distribution may be considered a *homogeneous* data set. The second type of distribution combines a collection of possible distributions and some sort of switching process. For example, imagine the following scenario. At the beginning of time, the goddess of chance picks some distribution at random from a family of possible distributions; for the rest of time, samples are drawn according to the chosen distribution. That is, if the goddess of chance chooses the distribution matched to x-rays, all of the samples look like x-ray samples; if she chooses the distribution matched to text, all of the samples look like text; and so on. This scenario, requiring a collection of distributions and a distribution over that collection, roughly describes the class of *nonergodic sources* (under conditions where the ergodic decomposition holds [19]). A data set coming from such a distribution may be thought of as an *inhomogeneous* data set.

The assumption of source ergodicity is appropriate when the data appears to be drawn from a single source; that is, when the data can be well-modeled by a single distribution. The assumption of source nonergodicity is appropriate in the case where the data statistics vary over a wider range of possibilities.

The beauty of the two-stage coding process described in the section titled "Take it to the Next Level" is that it achieves system models analogous to the multidistribution source models for stationary nonergodic sources *without* requiring a priori knowledge of the number or types of classes to be considered. That is, in our earlier example of compression for a home entertainment system, the system design procedure automatically divides the set of possible image types into the optimal categories (e.g., faces, landscapes, etc.), which need not even be understood by the system designer.

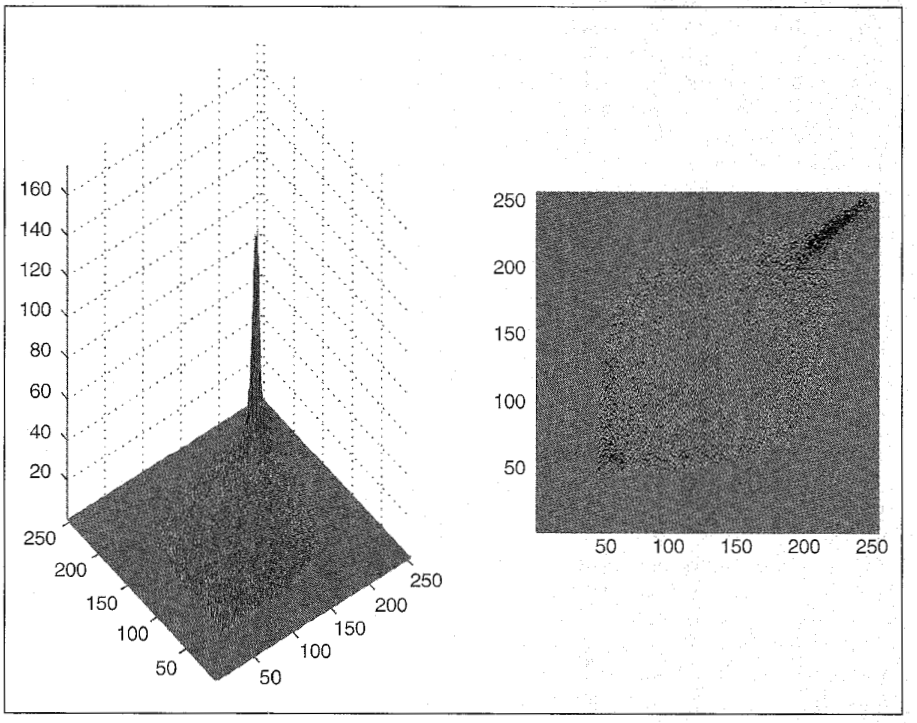


▲ *11. Sample α and β functions. The encoder α carves the space of possible data vectors into subsets, each of which is given a unique index. The decoder β chooses a representative value for each of the regions chosen by α. The given code was designed for an image including both photos and text. The two-dimensional statistics of the training image are illustrated by the histogram shown in Fig. 12. The above code suggests a much lower degree of correlation between samples x_1 and x_2 than suggested by the code in Fig. 5. This observation is consistent with the histogram differences between Figs. 12 and 6.*

particular source yields good performance on that source, but may result in poor performance when the same code is used for any other type of data. For example, the image statistics of Fig. 6 are well matched to the quantizer described in Fig. 5, and this code achieves good compression performance in coding that source. (In fact, the statistics shown in Fig. 6 are precisely the statistics for which the code in Fig. 5 was designed.) The same data, however, are not well-compressed (a 400% increase in distortion at the given rate) using the quantizer shown in Fig. 11, which was designed for an image scanned from a magazine page containing both photographic images and text. The two-dimensional statistics of the second image appear in Fig. 12.

While the above example illustrates the source dependency problem in the optimal coding paradigm presented earlier, the source dependency problem pervades nonoptimal source codes as well. For example, the best transforms for use in transform coding are data specific. The same rotation that aligns one set of data statistics with the axes for better scalar quantization performance makes another set of data statistics less suitable for scalar quantization and bit allocation. Even the use of standard transforms like the DCT (used, for example, in the JPEG image coding standard discussed briefly in Box 3) belies implicit assumptions about the expected source statistics. The DCT is a good decorrelating transform for smooth, slowly varying source statistics but achieves poor performance on sources characterized by sharp edges or abrupt changes. Thus, inherent in all of the codes discussed are explicit or implicit assumptions about the types of data to be compressed.

Any of the above-described codes can be optimized to match the statistics of a given data source. An optimal source code (vector quantizer) can be trained to obtain the optimal quantizer encoder and decoder and the optimal lossless code for the given set of data statistics; a transform can be designed to achieve the best possible data decorrelation; the optimal bit allocation depends on the transform coefficient statistics; the tables of a hierarchical code can be optimized for a given training set of data; and so on. Thus, one way to treat the problem of source variability is to combine all of the data into a single training set and design the best possible code for that training set. Using this approach, source-dependent com-



▲ *12. Histogram of two-dimensional pixel vector values for an image including both photos and text.*

ponents of a compression system are designed to do well *on average* across the training set but may not achieve, on any particular member of that set, performance as good as the performance achievable with a code designed specifically for that data. For example, the DCT transform used in JPEG yields a transform code that does well on average across a wide variety of images, but does not do as well on computer graphics as a transform code using a transform designed specifically for computer graphics.

Another approach for dealing with the problem of source variability in data compression systems is the adaptive approach. An adaptive technique involves a changing codebook that is continually redesigned in an attempt to match changing source statistics (see, for example, [7]). The advantage of adaptive algorithms is that (when they work very well), they track changing source statistics—providing continuous, matched coverage for every source encountered. Typically, adaptive codes work best on statistics that vary slowly over time or space and for which source changes can either be tracked independently by or communicated efficiently to the system decoder. The potential pitfalls of such approaches lie in the complexity, rate, and distortion trade-offs. Designing algorithms that adapt quickly enough to track changing source statistics while maintaining reasonable complexity, low code description rates, and general system stability is clearly a nontrivial problem. Further, many adaptive techniques fail to take advantage of the modal nature of data types such as images, where a particular image might share less in common with nearby images in the data stream and more in common with images elsewhere in the data set.

A third approach to the problem of source variability is to apply the lessons of rate-distortion theory at a higher level in the system in an attempt to design a single code that does as well on all possible source statistics as if it were specifically designed for the source in operation at any given time. As described in previous sections, rate-distortion theory treats the problem of designing small collections of very simple models (codewords) to cover the space of possible data samples. The same basic techniques may be applied to the higher-level problem of designing small collections of more sophisticated models (e.g., compression systems) to cover the space of possible compression systems. The focus of the remainder of this section is on this approach.

Intuitively, the design and use of any single code to do well on average across a collection of possible data types is equivalent to a coarse, high-level quantization. To understand this idea, imagine an abstract space of possible codes such that for each data type in the collection, there exists a single code designed specifically for that data type. Choosing one representative code to be used on all incoming data is equivalent to performing a rate-zero quantization on the space of possible codes. The single code chosen acts as an approximation to the optimal code for the data type in operation. Since only one code has been chosen, no rate need be spent in describing the chosen code to the decoder.

For many applications, multiple-model systems yield significant performance improvements over their single-model counterparts.

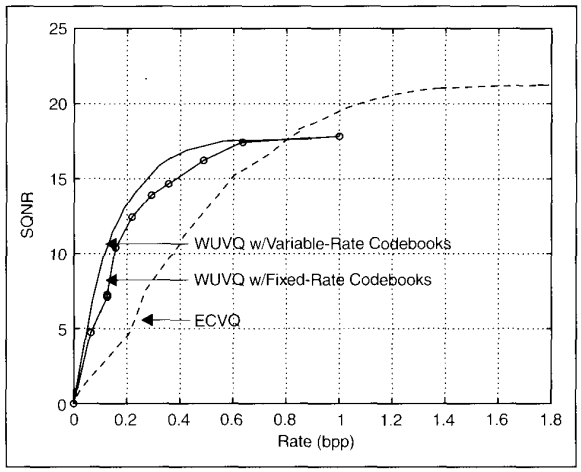
The trade-off between the rate spent on describing codewords and the distortion achieved by those codewords has a parallel in the trade-off between the rate spent in describing a code and the performance of the chosen code. A single code used to compress every source in the space of possible sources requires no rate for code description but achieves in return poor average performance in coding source samples from the given family of distributions. Replacing the single code with a large collection of codes (a "codebook of codebooks") increases (from zero) the amount of rate required to describe the code, but simultaneously improves the rate-distortion performance of the code used to describe the data. Systems of this type are called *two-stage* coding algorithms, since data descriptions are sent in two stages: the first-stage description specifies a code in the given collection of codes; the second-stage description contains the data encoded using the code described in the first stage.

The quantization interpretation leads to a variety of interesting rate-distortion theoretic results [11] in addition to an optimal design strategy [11, 12]. A brief look at the theory underlying the two-stage approach to source coding appears in the next section. The following section contains a description of the optimal design algorithm for a wide variety of two-stage source codes. A discussion of possible applications of this algorithm to other modeling problems follows. Before tackling these descriptions, it is worthwhile to pause to establish a bit of historical perspective.

The benefits of combining collections of codes into a single coding strategy appear both in rate-distortion theory and in practical code design. For example, basic proofs of the rate-distortion theorem for stationary ergodic sources with memory (see Box 4 for an intuitive look at the concept of ergodicity) use collections of codes to achieve good performance on the ergodic modes of an n-dimensional vector source created by combining nonoverlapping n-dimensional sequences of samples into vectors (see [13]). (For sources with memory, the ergodicity of the original source does not guarantee the ergodicity of an n-dimensional vector source for arbitrary values of n.) The derivation of the rate-distortion function for stationary nonergodic sources on Polish alphabets [14] arises from a similar ergodic decomposition argument.

Box 5 - Two-Stage Optimal Source Coding: The WUVQ Algorithm [11, 12]

The weighted universal vector quantization (WUVQ) algorithm is a two-stage code consisting of a collection of vector quantizers. The design of such a code is accomplished using the algorithm described in the section titled "Through the Looking Glass," where the codebook optimization step is itself accomplished with the generalized Lloyd algorithm on each of the codebooks in the collection. The result is a nested generalized Lloyd algorithm.



The graph appearing to the left shows the results of a variable-rate WUVQ on a sequence of medical images. The system, trained on a collection of 20 256×256 magnetic resonance (MR) images and tested on a disjoint set of five MR images of the same size, uses vector dimension $n = 4$ and makes new code decisions every $k = 4$ four-dimensional vectors. The following graph contains the rate vs. signal-to-quantization noise ratio (SQNR) results. Rate is reported in terms of entropy and SQNR is calculated as $-10 \log(D / D_0)$, where D is the expected distortion at the given rate and D_0 is the expected distortion at rate 0. Codebook sizes for first- and second-stage codebooks in the case of variable-rate coding were initialized to 256 and 4 respectively. Thus we have a large collection of very simple models rather than the single, more complex model of the variable-rate VQ (ECVQ), which uses up to 256 four-dimensional codewords. More than 7 dB performance improvement results from this choice.

The use of the generalized Lloyd algorithm with another optimization strategy inside the quantizer decoder optimization step appears in a variety of practical system-design algorithms. A few of these algorithms are described in detail in upcoming sections and in Boxes 5, 6, and 7. Other examples include use of the generalized Lloyd algorithm and the Levinson algorithm for designing linear predictors for speech [15], the generalized Lloyd algorithm and the Baum-Welch algorithm for designing hidden Markov models for speech recognition [16], the generalized Lloyd algorithm and the Huffman algorithm for designing entropy codes for subband coding [17], the generalized Lloyd algorithm and conditional expectation for designing probability mass functions for classification tree design [12], and the generalized Lloyd algorithm and the generalized Lloyd algorithm for designing individual vector quantizers for a residual vector quantizer [18]. Unfortunately, many of these algorithms fail to incorporate the first-stage coding rates (or their equivalents) in their design strategies. A brief discussion of the importance of including the first-stage coding performance measures in the overall optimization procedure accompanies the discussion of two-stage algorithms.

Codebooks of Codebooks: Two-Stage Universal Source Coding

By "covering" the space of possible source codes, optimal two-stage codes yield performance that asymptotically approaches the best possible performance for every source in some broad class of possible sources.

Conceptually, the one-stage compression systems and the two-stage compression systems described earlier are very similar. In each case, the system contains a list of possible models and their associated binary descriptions. The main difference between the systems is the type of model included in the list. In the one-stage quantizer, each model is a single codeword of some fixed dimension n, as shown in Fig. 1. Here choosing a model is equivalent to choosing a reproduction vector or codeword. In the two-stage quantizer, each model is itself a distinct n-dimensional compression system or codebook, as shown in Fig. 13. Here choosing a model is equivalent to choosing an algorithm for encoding the given data; e.g., choosing a particular vector quantizer from a collection of available vector quantizers. The one-stage code's encoder individually maps each n-dimensional source vector to a reproduction vector. The two-stage code's encoder maps each supervector of k n-dimensional vectors to a single n-dimensional source code. (The incoming sequence of samples is here called a supervector to indicate that the dimension kn of the incoming data vector may differ from the dimension n of the codes contained in the two-stage code's collection.) All k n-vectors are then reproduced using the chosen source code (but not necessarily the same reproduction). In going from a single quantizer to a collection of quantizers, one must either increase the overall complexity of the system or decrease the complexity of each quantizer in the collection.

Rate-distortion theory, in particular the theory of universal source codes, treats the case where the two-stage code is a collection of optimal vector quantizers like the ones described earlier. In this case, two-stage coding

yields algorithms that are, in some sense, source-independent. That is, the resulting "universal" codes achieve performance that asymptotically approaches the optimal performance for every source in a given class of possible sources. Roughly speaking, the argument proceeds as follows. Consider a two-stage code with first-stage coding dimension k and second-stage coding dimension n. The two-stage code contains a collection of n-dimensional optimal vector quantizers. The incoming data sequence is divided into kn-dimensional supervectors, each of which comprises k n-dimensional vectors. In describing a particular supervector, the encoder first describes the single n-dimensional vector quantizer in the code's collection that gives the best performance on average across those k vectors. The encoder then describes each n-vector using the chosen code. As the dimensions k and n grow without bound, the cost (in rate) of describing a particular quantizer in the collection of quantizers is amortized over more and more symbols. As a result, the optimal number of quantizers to use in the collection likewise grows without bound, thereby filling more and more densely the space of possible quantizers and achieving performance closer and closer to the optimal performance on each source in the given class.

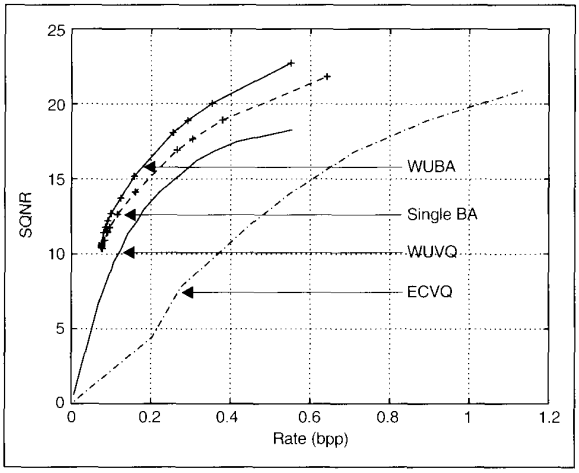
Since all binary descriptions are assumed to be uniquely decodable, the concatenated sequence of binary descriptions is likewise uniquely decodable.

Unfortunately, as the above argument indicates, universal performance is achieved only in the limit as the vector dimension grows without bound. Thus, universal coding theory suffers from the same curse of dimensionality suffered by the earlier described optimal vector quantizers. Getting the best possible performance requires use of the optimal universal code at the highest possible dimension. Nonetheless, as the argument of the "Power of Imperfection" section suggests, the "optimal" code in a rate-distortion sense is not necessarily optimal in

Box 6 - Getting the Most Bang for Your JPEG Buck

As mentioned in Box 3, the header of a JPEG compressed data file contains a description of the quantization matrix (bit allocation) used in compressing the given file. As a result, this quantization matrix may be changed from image to image, yielding good performance on a wide variety of different image types.

Unfortunately, many commercial implementations of JPEG fail to take full advantage of the flexibility afforded by this feature, using only a single quantization matrix (scaled up and down to achieve a variety of rates). The price to be paid for this omission is illustrated in the following graph from [20, 22], which compares a scheme using a single bit allocation (labeled "single BA") to the weighted universal bit-allocation algorithm (WUBA), a two-stage coding scheme with an optimal collection of bit allocations. Both the single bit- allocation scheme and WUBA use input vector dimension $n = 64$ as does JPEG. The single bit allocation algorithm uses the same quantization matrix for all image blocks. The WUBA uses, on each 64-dimensional vector, the bit allocation (from the code's collection) that best matches the given data vector ($k = 1$). (Notice that the WUBA takes the notion of quantization matrix variation beyond that allowed in the baseline JPEG algorithm since the WUBA allows a change of bit-allocation scheme at each vector rather than requiring the same bit allocation throughout a single image.) The following graph compares systems optimized with respect to the same training set using the squared-error distortion measure. This distortion measure may be easily replaced with perceptual distortion measures such as the one described in [20].

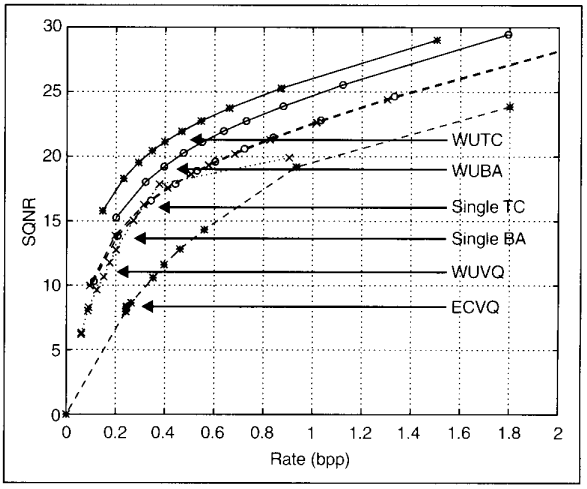


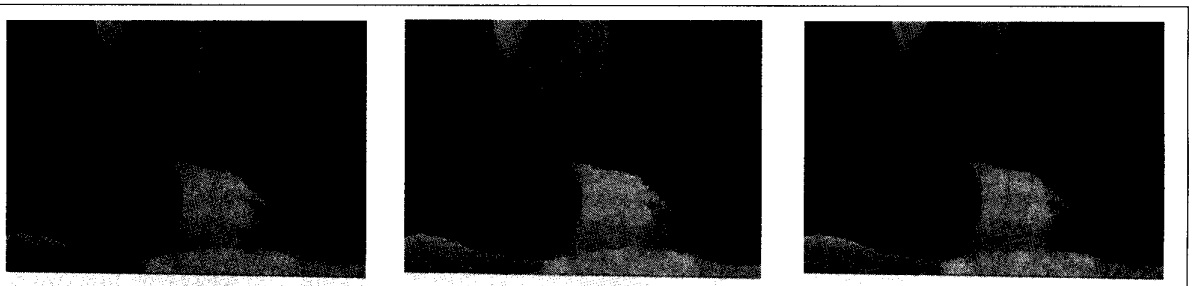
The graph to the left illustrates the 2-3 dB gains achievable on the data set described in Box 5 using a collection of bit allocations rather than just a single bit allocation. These gains are somewhat surprising given the homogeneity of the data set (a sequence of MR brain scans). Given the above improvements and the fact that JPEG does allow some quantization matrix variation [1], the fixed quantization matrix strategy of many JPEG implementations seems poorly motivated. This system design choice results not from any argument regarding the rate-distortion benefit of a single quantization matrix choice but instead from a perspective of computational expense. In particular, designing the optimal quantization matrix for each incoming image is computationally expensive. Two-stage codes represent an alternative to this approach that achieves most of the performance benefits at a fraction of the (run-time) computational expense.

Box 7 - Beyond JPEG: Optimal Two-Stage Transform Codes [20, 26]

While the WUBA algorithm described in Box 6 achieves impressive gains over the (optimal) single bit-allocation version of JPEG, it also leaves room for performance improvement. In particular, the DCT, inherited from JPEG, is not an optimal decorrelating transform for all data types. In fact, the optimal transform is itself data-dependent. The graph shown to the right gives the performance of a weighted universal transform code (WUTC) as compared to a variety of the other codes discussed in previous boxes. The weighted universal transform code is a two-stage source code containing a collection of transform codes. Each transform code uses an optimal decorrelating transform (the Karhunen-Loeve transform) in addition to an optimal bit allocation scheme. In each code, both the transform and the bit allocation are matched to the statistics of the data to which the particular code is applied. The performance of an optimal single transform code (labeled "single TC") is also included. The following experiments used a combined text and image data set for both training and testing. The training and test sets do not overlap.

The three images included below show the results of the above approach. The image on the far left is the original. The image in the center results from optimal transform coding using a single transform and bit allocation at rate 0.20 bits per pixel. The image on the right results from weighted universal transform coding using up to 64 transform codes at rate 0.23 bits per pixel.





a rate-distortion-complexity sense. Thus, suboptimal variations on the two-stage approach to universal coding theory may achieve performance at a given level of complexity that far exceeds that of the rate-distortion optimal code of the same complexity. Design of optimal and suboptimal codes based on the two-stage approach to source coding is considered in the next section.

Through the Looking Glass: Generalizing the Generalized Lloyd Algorithm

Two-stage code design is accomplished using the iterative design algorithm described earlier, which is modified only in its definitions of "codewords" and its measurements of rate and distortion.

Almost any algorithm can be built into a two-stage code. In particular, any of the codes discussed up to this point can be considered as potential candidates. For example, one could devise a two-stage code using a collection of vector quantizers (see Box 5), a two-stage code using a collection of JPEG quantization matrices (see Box 6), a two-stage code using a collection of optimal transform codes (see Box 7), or even a hybrid two-stage code containing some combination of codes of different varieties. In each case, the general design strategy is the same. A description of this strategy follows.

The earlier description of a basic compression system as an indexed list of possible reproduction vectors and their corresponding binary descriptions (shown in Fig. 1) parallels the similar illustration of a two-stage code in Fig. 13. The first column contains the row index i, the second

column contains the code $\beta(i)$, and the third column contains the uniquely decodable binary description $\gamma(i)$. An encoding function α plays the role of interface between a given data set and the above described list. For any $x^{kn} = (x_1^n, x_2^n, \ldots, x_k^n) \in \mathcal{X}^{kn}$, $\alpha(x^{kn})$ equals the row index of the n-dimensional code chosen for describing the k n-vectors $x_1^n, x_2^n, \ldots, x_k^n$. The choice of a code index i is typically made independently for each kn-vector [20]. (For a discussion on the use of higher-order statistics in two-stage coding, see [21].) The description of a single kn-vector x^{kn} using a two-stage code is the concatenation of the binary description $\gamma(\alpha(x^{kn}))$ of the chosen code index followed by the binary descriptions of $x_1^n, \ldots, x_k^n$ using the described code $\beta(\alpha(x^{kn}))$. This process is reversed at the decoder. That is, the given binary sequence is decoded by first decoding the index of the chosen code, and then using that code's decoder to interpret the binary descriptions of the vectors $x_1^n, \ldots, x_k^n$. Since all binary descriptions are assumed to be uniquely decodable, the concatenated sequence of binary descriptions is likewise uniquely decodable.

Given an algorithm for designing a single good code of a particular type (e.g., an optimal vector quantizer or a good JPEG quantization matrix to match the statistics of a given training set), the questions under consideration here are how to design the best *collection* $\{\beta(i)\}$ of codes of that type, how to design the best binary descriptions $\{\gamma(i)\}$ for those codes, and how to design the encoder α to best choose among the codes in the collection.

Given a collection of codes, the optimal encoder α has the property that for each $x^{kn} \in \mathcal{X}^{kn}$, $\alpha(x^{kn})$ is the index of the code that achieves the best Lagrangian performance in coding x^{kn}. The Lagrangian performance measure equals the weighted sum of the rate and distortion associated with a given code choice. The rate associated with a particular code choice equals the sum of the rate necessary for describing the chosen code (using a lossless code matched to the code probabilities) and the rate necessary to code the data using that code. The distortion associated with a given code is the total distortion achieved in coding the k n-dimensional vectors $x_1^n, \ldots, x_k^n$ using that code. The optimal encoder for a two-stage code is implemented using a full search analogous to the full search of the optimal encoder of a vector quantizer (see Box 1). Suboptimal first-stage coders are also possible. For example, the code choice could be accomplished using a tree search, table look-up, or any other variety of fast search technique.

The optimal first-stage lossless code is a uniquely decodable lossless code that represents each index i by a bi-

i	$\beta(i)$	$\gamma(i)$	i	$\beta(i)$	$\gamma(i)$	i	$\beta(i)$	$\gamma(i)$
1	Code 1	111100	14	Code 14	111111010	27	Code 27	110100
2	Code 2	1110110	15	Code 15	111010	28	Code 28	1111111110
3	Code 3	111110111	16	Code 16	11010111	29	Code 29	11111000
4	Code 4	111111011	17	Code 17	11010110	30	Code 30	11111010
5	Code 5	11011010	18	Code 18	111000	31	Code 31	10
6	Code 6	111110010	19	Code 19	0	32	Code 32	11011101
7	Code 7	11111110	20	Code 20	11110111	33	Code 33	11111100
8	Code 8	1111111111	21	Code 21	1110011	34	Code 34	1110010
9	Code 9	1101010	22	Code 22	111111110	35	Code 35	1100
10	Code 10	11110111	23	Code 23	1111010	36	Code 36	110110110
11	Code 11	11011100	24	Code 24	111110110	37	Code 37	110110111
12	Code 12	11101111	25	Code 25	111110011	38	Code 38	11101110
13	Code 13	11011110	26	Code 26	1101100	39	Code 39	11011111

▲ *13. A two-stage source-coding codebook. The first column contains the code index i. The second column contains the n-dimensional source code* $\beta(i)$. *The third column contains the binary description* $\gamma(i)$.

The generalized Lloyd algorithm solves a number of very interesting and important problems.

nary description of length approximately equal to the negative logarithm of the probability of that index. For example, if the code with index $i = 1$ is used to code proportion $p(1)$ of the kn-dimensional supervectors, then the optimal description length for this code is $(-\log p(1))$ bits. Thus, no matter how complex the codes in a given collection, the rate needed to describe any member of that collection is a function merely of the probability of that model.

Given an encoder α and binary code γ, the optimal collection $\{\beta(i)\}$ of codes is the collection of codes that achieves the best possible Lagrangian performance. In particular, for each i, $\beta(i)$ should be the code that achieves the optimal performance possible for the data $\{x^{kn} \in \mathcal{X}^{kn} : \alpha(x^{kn}) = i\}$ that maps to it. For any fixed coding dimension n, the rate-distortion optimal code $\beta(i)$ is the optimal n-dimensional vector quantizer matched to the source statistics of the data that mapped to the given code. In a (rate-distortion) suboptimal code, each code in the two-stage code's collection may itself be a suboptimal code; e.g., a transform code. For a given suboptimal code type, the optimal collection of codes must similarly have the property that each code in the collection is optimal (subject to the given code structure) for the data that maps to it. For example, in a two-stage code containing a collection of transform codes, each transform code in the collection should be optimized for the data that maps to it.

A further generalized version of the generalized Lloyd algorithm, or more correctly its entropy-constrained variation, may be used to optimize two-stage codes of any variety. The process initiates with an arbitrary collection $\{\beta(i)\}$ of models—in this case a collection of codes of a desired type or types—and their uniquely decodable binary descriptions $\{\gamma(i)\}$. The algorithm is again an iterative descent technique, which is run to convergence. Each iteration requires three steps.

Step 1: Optimize the encoder α for the given collection $\{\beta(i)\}$ of codes and their binary descriptions $\{\gamma(i)\}$.

This step is optimally accomplished using a full search over all of the codes in the collection. That is, each kn-dimensional supervector is encoded using each of the codes in the collection and the Lagrangian performance of each such code on the given data vector is calculated. Each supervector x^{kn} is mapped to the index i of the code yielding the best Lagrangian performance on that supervector.

Step 2: Optimize the collection of codes $\{\beta(i)\}$ for the given encoder α and lossless code γ.

If the codes in the given collection are vector quantizers, this step is accomplished using the generalized Lloyd algorithm to optimize each vector quantizer for the data that mapped to it. That is, codebook $\beta(i)$ is redesigned using the generalized Lloyd algorithm on the subset $\{x^{kn} : \alpha(x^{kn}) = i\}$ of the original training set. If the given code collection contains other varieties of codes, then the optimal design algorithms for those code types are used in the place of the generalized Lloyd algorithm.

Step 3: Optimize the binary descriptions $\{\gamma(i)\}$ for the given encoder α and decoder β.

This step is accomplished by matching an entropy code to the probabilities of the codes in the given collection. The model probabilities are estimated on the training set. That is γ is designed such that $|\gamma(i)| \approx -\log \Pr(\alpha(X^n) = i)$ for each i.

One key decision in designing a collection of codes is the choice of the number of codes. As the arguments of previous section suggest, the theoretically optimal number of codebooks in a two-stage code is a function of the space of possible sources (a more homogeneous data set can be covered with fewer codes than can a less homogeneous data set) as well as the coding dimension. In practice, the number of codes used in a two-stage code is more commonly the combined result of complexity constraints and design outcomes. By including the rate used to describe the model in the rate calculation, the above system inherits from its predecessor a codebook size optimization mechanism. That is, the number of models used in the system can itself be optimized through the iterative descent procedure. Often, the number of codes is also controlled by concerns about complexity. (It is interesting to note that the performance of a collection of very simple codes often exceeds the performance of much more complex single-code algorithms.) In practice, two-stage compression algorithms are typically initialized with the largest collection $\{\beta(i)\}$ of source codes that is computationally feasible. Through the iterative descent design process, the probabilities of any "extra" codes tend to zero, their associated rates approach infinity, and the codes are effectively removed from the system.

If $k = 1$, a two-stage system using a collection of n-dimensional codes can be redrawn as a one-stage system of the same dimension. For any $k > 1$, however, the same is not the case. Further, even at $k = 1$, the existence of an equivalent one-stage code does not imply the existence of a low-complexity implementation corresponding to that code. For example, using $k = 1$ and a collection of transform codes yields a code that is not easily implemented directly.

A variety of practical codes can be built using this approach (see [20] for a summary). The resulting systems yield vast performance improvements, even on data sets that seem quite homogeneous. For example, a two-stage code containing a collection of vector quantizers yields a 10 dB performance improvement over a single vector quantizer when both systems are trained on a collection of 20 sagittal magnetic resonance brain scans and tested on a (nonoverlapping) collection of five scans of the same type. Examples of two-stage source codes designed using the above technique appear in Boxes 5, 6, and 7. A discussion of the design and use of multiple-model sys-

tems for applications other than source coding appears in the next section.

Beyond Source Coding: A Look at the World Through $R(D)$ Colored Glasses

Given appropriate definitions of "rate" and "distortion," a wide variety of modeling problems may be viewed through the lens of rate-distortion theory. This nontraditional perspective lends valuable insight into the solution of numerous engineering problems in which modeling uncertainty plays a major role.

At first glance, rate-distortion theory appears applicable only to the compression problems from which it arises. *Yet the idea that better performance may be achieved by using a collection of simple models rather than a single all-encompassing model is a fundamental lesson of rate-distortion theory that is applicable to an enormous variety of problems in which uncertainty plays a critical role.* The design and use of multiple-model systems as a general tool for uncertainty management are explored in this section.

As the compression example discussed throughout this work demonstrates, the underlying principle behind multiple model systems is extremely simple. In applications where uncertainty plays a major role, there are often performance benefits to be gained by separating sources of uncertainty and designing a collection of simple models rather than a single, more complex model. In speech recognition, that might mean designing different recognition systems for men and women and optimizing systems for different regional accents; for mobile communications, that might mean designing a collection of different channel codes for use under varying channel characteristics; and so on. The questions that arise in considering the use of a multiple model system for any of these applications are the same as the questions considered for multiple-model compression systems:

- How many models should be designed?
- Which models should be used?
- What are the price and payoff of multiple-model systems?

While it is not difficult to come up with ad-hoc answers to some of these questions, experimental results [20,23,24] indicate that there are enormous gains to be achieved by using optimal collections of models rather than collections that are merely intuitively satisfying.

Under the conditions where it applies, the same theory used to provide optimal answers to the above questions for the source coding problem may be used to tackle these questions for other applications. The system requirements necessary for successful application of that theory are enumerated below.

1. There must exist a performance criterion (analogous to the Lagrangian performance of a rate-distortion problem) such that:

(a) the goal of system design is the minimization of the given criterion (The restriction to problems described as minimizations rather than maximizations is included for convenience. No loss of generality results from this restriction since any maximization becomes a minimization upon negation of the associated performance criterion.);

(b) the value of the performance criterion is bounded below.

2. There must exist a model redesign algorithm such that given a pre-existing model and a set of data, the given algorithm redesigns the model such that the new model gives performance at least as good as the performance of the original model on the given data set.

3. There must exist a testing procedure such that given a collection of models and any possible scenario, some means of determining which model yields the best performance in the given scenario is available.

The lessons of rate-distortion theory lead to optimal methods for designing collections of very simple source models.

The above requirements are necessary and sufficient for application and convergence of the multiple model design algorithm described in the previous section. The algorithm is not described in detail again here, but a rough outline follows. First, separate the training set into a collection of disjoint subsets. The initial subsets may be arbitrary, or they may be chosen to match the designer's intuition. For example, the data in a speech training set may be separated by regional accent, speaker age, or speaker gender. The number of initial subsets chosen should equal the maximal number of models acceptable (from a computational standpoint) in the system. Design an initial model for each subset of the training data. Then reclassify the data in the training set, grouping each member of the training set with the model that achieves the best performance on that data. Calculate the proportion of data that maps to each model in the system and use that proportion as an estimate of the model probability. Iterate the procedure—repeatedly designing each model to match the data that mapped to it and then remapping the data until convergence of the performance criterion.

The requirements described in conditions 1-3 are precisely the conditions needed to implement the above procedure and guarantee its convergence. The requirement in 1(a) provides a means for evaluating system performance and an explicit design goal. The requirement in 1(b) is necessary to guarantee convergence of the multiple model design algorithm. The design algorithm in 2 is necessary for redesigning each model for the data that mapped to it, and the requirement of system improvement is necessary for the algorithm's convergence. If the design algorithm guarantees not only a good solution but a globally optimal solution (or that algorithm may itself be described in a sequence of steps such that each step guarantees a globally optimal solution), then the multi-

ple-model-design algorithm is guaranteed not only to converge but to converge to a locally optimal solution. The requirement in 3 is necessary for designing a method for choosing among the models in a given collection.

A final characteristic of most systems employing the multiple-model approach is a dependence of the performance criterion on the model probabilities. For example, in two-stage source codes, the Lagrangian performance includes the first-stage source description, which is roughly equal to the negative logarithm of the probability of the chosen model. Inclusion of this probability in the performance criterion in a manner that favors high probability models over models with low probabilities yields multiple-model systems in which the optimal number of models at a particular dimension is bounded.

As mentioned at the end of the section titled "Take it to the Next Level," the multiple-model approach has been applied in a variety of applications. Applications incorporating the use of first-stage coding rate (or its equivalent) include image compression [20], communications over randomly varying channels [25], and speaker- and context-independent continuous speech recognition [23, 24]. In each of these applications, the performance improvements garnered by going from single- to multiple-model systems are striking:

▲ up to 10 dB compression performance improvement may be observed between one- and two-stage codes of the same type [20];

▲ up to 9 dB performance improvement of multiple-model systems over single-model systems may be observed in joint source and channel coding performance for a system with variable channel characteristics [25];

▲ a 50% decrease in word-error probability may be observed using collections of HMMs rather than a single HMM in speaker-independent continuous speech recognition [23, 24];

Conclusions

Rate-distortion theory quantifies the optimal trade-off between modeling cost and modeling resources in data compression systems. While the theory does not provide the optimal system for achieving the bounds it describes, rate-distortion theory does highlight a number of important aspects of optimal source coding systems, including the performance benefits to be gained from high-dimensional coding and the properties of optimal quantizers and lossless codes. The generalized Lloyd algorithm combines the lessons of rate-distortion theory into an iterative descent technique for designing (locally) optimal data compression systems from a rate-distortion perspective.

It is interesting to note that the data compression system that achieves the optimal performance at a given complexity is often a suboptimal code from a pure rate-distortion perspective. This seeming anomaly arises from the fact that rate-distortion theory largely ignores issues of computational complexity. As a result, performance gains may be achieved by giving up optimal encoding and decoding in exchange for the ability to code at higher dimensions. The goal in designing such suboptimal codes is to achieve the best possible rate-distortion trade-off subject to the constraints imposed by the coding structure.

The lessons of rate-distortion theory lead to optimal methods for designing collections of very simple source models. Applying these same techniques to more general data models yields an optimal algorithm for designing collections of more general data models for use in a wide variety of applications. Thus, rate-distortion theory provides a theoretical framework for deciding how best to break any single, complex modeling problem into a collection of simpler modeling problems. The resulting systems are computationally inexpensive to use since all of the design is done off-line rather than during the coding process (as in adaptive systems). Further, the design requires no expert intervention. The algorithm finds the optimal way to divide a problem automatically. For many applications, such as image compression, speech recognition, and joint source and channel coding, the resulting multiple-model systems yield significant performance improvements over their single-model counterparts.

Acknowledgments

This material is based upon work partially supported by NSF CAREER Award MIP-9501977, a grant from the Charles Lee Powell Foundation, and a donation through the Intel 2000 program.

M. Effros is with the Department of Electrical Engineering of the California Institute of Technology in Pasadena, California, USA (e-mail:effros@z.caltech.edu).

References

1. W.B. Pennebaker and J.L. Mitchell, *JPEG Still Image Compression Standard*. Van Nostrand Reinhold, New York, 1993.

2. T.M. Cover and J.A. Thomas, *Elements of Information Theory*. Wiley, 1991.

3. T.D. Lookabaugh and R.M. Gray, High resolution quantization theory and the vector quantization advantage. *IEEE Transactions on Information Theory*, IT-35(5):1020-1033, September 1989.

4. A. Gersho, Asymptotically optimal block quantization. *IEEE Transactions on Information Theory*, 25(4):373-380, July 1979.

5. P.A. Chou, T. Lookabaugh, and R.M. Gray, Entropy-constrained vector quantization. *IEEE Transactions on Acoustics Speech and Signal Processing*, 37(1):31-42, January 1989.

6. Y. Linde, A. Buzo, and R.M. Gray, An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84-95, January 1980.

7. A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, 1992.

8. P.A. Chou, T. Lookabaugh, and R.M. Gray, Optimal pruning with applications to tree structured source coding and modeling. *IEEE Transactions on Information Theory*, IT-35(2):299-315, March 1989.

9. K. Ramchandran and M. Vetterli, Best wavelet packet bases in a rate-distortion sense. *IEEE Transactions on Image Processing*, 2(2):160-175, April 1993.

10. M. Vishwanath and P.A. Chou, An efficient algorithm for hierarchical compression of video. In *Proceedings of the IEEE International Conference on Image Processing*, volume 3, pages 275-279, Austin, TX, November 1994.

11. P.A. Chou, M. Effros, and R.M. Gray, A vector quantization approach to universal noiseless coding and quantization. *IEEE Transactions on Information Theory*, IT-42(4):1109-1138, July 1996.

12. P.A. Chou, Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4), April 1991.

13. T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

14. M. Effros, P.A. Chou, and R.M. Gray, Variable-rate source coding theorems for stationary nonergodic sources. *IEEE Transactions on Information Theory*, IT-40(6):1920-1925, November 1994.

15. A. Buzo, A.H. Gray Jr., R.M. Gray, and J.D. Markel, Speech coding based upon vector quantization. *IEEE Transactions on Acoustics Speech and Signal Processing*, 28:562-574, October 1980.

16. L.R. Rabiner, J.G. Wilpon, and B.-H. Juang, A segmental *K*-means training procedure for connected word recognition. *AT&T Technical Journal*, 64(3):21-40, May 1986.

17. R.J. Safranek and J.D. Johnston, A perceptually tuned sub-band image coder with image dependent quantization and post-quantization data compression. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1945-1948, Glasgow, May 1989.

18. W.-Y. Chan and A. Gersho, Constrained-storage quantization of multiple vector sources by codebook sharing. *IEEE Transactions on Communications*, COM-39(1):11-13, January 1991.

19. R.M. Gray, *Probability, Random Processes, and Ergodic Properties*. Springer-Verlag, New York, 1988.

20. M. Effros, P.A. Chou, and R.M. Gray, Universal image compression. 1996. Submitted to the *IEEE Transactions on Image Processing,* December 18, 1996. In review.

21. M. Effros, Conditional weighted universal source codes: second order statistics in universal coding. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2733-2736, Munich, Germany, April 1997.

22. M. Effros and P.A. Chou. Weighted universal bit allocation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2343-2346, Detroit,MI, May 1995.

23. J.C. Fang, *Speaker independent continuous speech recognition using multiple hidden Markov models*. Senior Thesis, Dept. of Electrical Engineering, California Institute of Technology, Pasadena, CA, 1998.

24. J.C. Fang and M. Effros, Speaker and context independent continuous speech recognition using multiple hidden Markov models. In preparation for submission to ICASSP-99 and *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

25. M. Effros, Robustness to channel variation in source coding for transmission across noisy channels. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2961-2964, Munich, Germany, April 1997.

26. M. Effros and P.A. Chou, Weighted universal transform coding: universal image compression with the Karhunen-Loeve transform. In *Proceedings of the IEEE International Conference on Image Processing*, Washington, D.C., October 1995.