Generalized Threshold Replenishment: An Adaptive Vector Quantization Algorithm for the Coding of Nonstationary Sources

James E. Fowler, *Member, IEEE*

***Abstract*— In this paper, we describe a new adaptive-vector-quantization (AVQ) algorithm designed for the coding of nonstationary sources. This new algorithm, generalized threshold replenishment (GTR), differs from prior AVQ algorithms in that it features an explicit, online consideration of both rate and distortion. Because of its online nature, GTR is more amenable to real-time hardware and software implementation than are many prior AVQ algorithms that rely on traditional batch training methods. Additionally, as rate-distortion cost criteria are used in both the determination of nearest-neighbor codewords and the decision to update the codebook, GTR achieves rate-distortion performance superior to that of other AVQ algorithms, particularly for low-rate coding. Results are presented that illustrate low-rate performance surpassing that of other AVQ algorithms for the coding of both an image sequence and an artificial nonstationary random process. For the image sequence, it is shown that 1) most AVQ algorithms achieve distortion much lower than that of nonadaptive VQ for the same rate (about 1.5 b/pixel), and 2) GTR achieves performance substantially superior to that of the other AVQ algorithms for low-rate coding, being the only algorithm to achieve a rate below 1.0 b/pixel.**

***Index Terms*— Adaptive vector quantization, generalized threshold replenishment, image-sequence coding.**

I. Introduction

OVER THE last 20 years, vector quantization (VQ) [1] has received significant attention as a powerful technique for data compression. VQ is theoretically attractive due to results from rate-distortion theory that show that VQ is asymptotically optimal for the coding of a data source whose statistics are stationary in time. Although VQ has been successfully applied to the coding of many types of data, including speech, audio, images, and video [1], such sources can rarely be assumed to be stationary in practice, leading to a gap between the performance predicted by theory and that actually obtained in real implementations. Indeed, the nonstationary nature of sources common in practical applications has prompted a search for more general VQ algorithms that are capable of adapting to changing source statistics as coding progresses. Such algorithms (e.g., [2]–[16]) are called *adaptive vector quantization* (AVQ).

In this paper, we first briefly outline a mathematical definition of AVQ which describes the structure of AVQ communication systems as entailed by many algorithms appearing in prior AVQ literature. We follow this background discussion with the key contribution of this paper, a new AVQ algorithm called *generalized threshold replenishment* (GTR). GTR differs from prior algorithms in that 1) it is an online algorithm that does not rely on substantial buffering or iterative processing, and 2) it employs an explicit consideration of both rate and distortion, two important quantities that measure the performance of a source-coding algorithm. Because of its online nature, GTR is more amenable to real-time hardware and software implementation than are many prior AVQ algorithms; because of its simultaneous consideration of both rate and distortion, the performance of GTR surpasses that of other AVQ algorithms, particularly for low-rate coding. We illustrate this performance advantage with a survey of experimental results wherein we investigate the rate-distortion performance of GTR not only on an artificially generated nonstationary source, but also on real data in the form of an image sequence. These experimental results include comparisons between GTR, nonadaptive VQ, other AVQ algorithms, and theoretic bounds. These results show that GTR achieves rate-distortion performance superior to that of other reported AVQ algorithms, particularly for low-rate coding.

The organization of this paper is as follows. First, in Section II, we briefly review a general framework for AVQ that parallels the traditional theory of nonadaptive VQ. As we have covered this background material elsewhere, the discussion here will be brief. Next, in Section III, we describe in detail our GTR algorithm. Then, in Section IV, we review experimental results, and finally, in Section IV, we make some brief concluding remarks.

II. Adaptive Vector Quantization

In this section, we define and describe AVQ within a general framework. This discussion summarizes detailed observations [17] of a number of previously published AVQ algorithms. We begin by proposing a mathematical definition of AVQ. To date, there has been a certain imprecise use of the word "adaptive" in VQ literature, resulting in confusion as to exactly what is entailed by an AVQ algorithm. Although a number of

Manuscript received November 30, 1996; revised July 29, 1997. This work was supported by an AT&T Ph.D. Scholarship. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Antonio Ortega.

The author was with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210 USA. He is now with the Department of Electrical and Computer Engineering, Mississippi State University, Starkville, MS 39762 USA (e-mail: fowler@ece.msstate.edu).

Publisher Item Identifier S 1057-7149(98)06861-4.

disparate VQ techniques (e.g., mean-adaption, gain-adaption, switched-codebook-adaption, and vector-excitation-coding algorithms, as well as finite-state VQ, predictive VQ, and variable-dimension VQ, all surveyed in [1]) can be considered, in one way or another to be adaptive, many are perhaps better viewed as collections of multiple nonadaptive source coders coupled with a source model that partitions the input source, as it is within this framework that the design of these systems is usually carried out. It is an idea common to many reported algorithms that AVQ properly refers to techniques that dynamically vary the contents of a VQ codebook as coding progresses. For this reason, we propose, in Section II-A, a mathematical definition that captures this codebook-updating property and establishes it as the fundamental nature of AVQ. We then proceed to Section II-B, wherein we present a communication-system model for AVQ algorithms. This model reflects the structure typical to AVQ algorithms as suggested by our observations and provides a context in which to discuss issues involved in practical implementation of these algorithms. We note that the discussion that follows in Sections II-A and II-B is intended merely to lay a foundation for the development of our new GTR algorithm in Section III. As we have devoted other publications [17]–[19] to a more complete examination of this background material, the discussion given here will be brief.

A. Mathematical Definition of AVQ

In this section, we define mathematically the concept of an adaptive vector quantizer. We begin by reviewing the theory of nonadaptive VQ as our development of AVQ parallels it. We note that nonadaptive VQ has been covered extensively elsewhere, most prominently in the comprehensive book by Gersho and Gray [1].

VQ is the generalization of scalar quantization to higher dimensions [1]. Briefly, nonadaptive VQ consists of a vector quantizer, Q, that maps vectors from N-dimensional space to a fixed, finite set $\mathcal{C}$ of N-dimensional vectors; i.e.,

$$Q\colon \Re^N \rightarrow \mathcal{C}. \tag{1}$$

Set $\mathcal{C}$ is called the *codebook* of the vector quantizer.

Rate-distortion theory [20] states that, for a stationary, ergodic random process, there exists a rate-distortion function, $R(D)$, such that, for a given distortion D, $R(D)$ is the lower bound on the minimum average rate achievable by any coding method. In essence, the theory shows the existence of a vector quantizer that achieves this bound as the dimension of the quantizer becomes infinitely large [20].

This theoretic asymptotic optimality of VQ has inspired its use in many applications. However, most sources of practical interest are, in reality, nonstationary. A number of AVQ algorithms (e.g., [2]–[16]) have been introduced to provide more efficient coding in these applications. These AVQ algorithms compensate for the changing source statistics associated with nonstationary sources by periodically updating the VQ codebook.

We have developed a mathematical definition to describe the operation of these AVQ algorithms. A brief summary of this mathematical definition follows; a more comprehensive discussion is given elsewhere. [17], [18]. Assume that we have an N-dimensional random-vector process, $\mathbf{X}_t$. We define *adaptive vector quantizer*, Q_t, as follows. Let $\mathcal{C}^*$ denote a large *universal codebook*, $\mathcal{C} \subseteq \Re^N$, that is fixed for all time t. We define a sequence of *local codebooks*, $\mathcal{C}_t$, such that

$$\mathcal{C}_t \subset \mathcal{C}^* \tag{2}$$

at each time t. We restrict each set $\mathcal{C}_t$ to be finite. Adaptive vector quantizer Q_t is a time-variant mapping from N-dimensional Euclidean space to the local codebook for time t; i.e.,

$$Q_t\colon \Re^N \rightarrow \mathcal{C}_t. \tag{3}$$

The output of the adaptive vector quantizer is another random-vector process

$$\hat{X}_t = Q_t(\mathbf{X}_t). \tag{4}$$

The definition of AVQ given above is in some respects reminiscent of the theory of universal source coding [21]. Briefly, given a class of sources, a source coder is said to be *universal* if it can optimally code each of the sources of the class without knowing beforehand which source is being coded [21]. Some approaches to universal source coding use universal codebooks; for example, Neuhoff *et al.* [21] describe a large universal codebook that contains all the optimal block source codes for each of the individual sources of the class. Despite the fact that lossy universal source-coding algorithms have been mainly of theoretical interest [22], the theory of universal source coding is an important inspiration behind the development of several AVQ algorithms. Indeed, if an AVQ algorithm can be proven to achieve rate-distortion optimality for a particular class of sources, it can be considered to be a universal algorithm for that class. Such universal results have, in fact, been shown for certain AVQ algorithms (e.g., [22]). An example of a practical design procedure for lossy universal source coding was proposed by Chou *et al.* [23] who implement a *weighted universal vector quantizer* (WUVQ), a type of switched-codebook-adaption VQ [1].

Despite the fact that AVQ is a possible approach to lossy universal source coding [22], the goal of universal source coding is fundamentally different from that of AVQ: a universal source code attempts to learn the statistics (which are unknown but usually assumed to be stationary) of a single source over a long period of time, while AVQ attempts to compensate for source statistics that change over time. Additionally, universal coders such as WUVQ [23] possess a structure sufficiently different from that of most AVQ algorithms as to make a meaningful experimental comparison between them difficult. For these reasons, a comprehensive comparison between universal source coding and AVQ techniques is beyond the scope of this paper.

The mathematical definition that we have described in this section provides a general representation of AVQ; however, this definition alone offers little insight into the structure necessary for the implementation of AVQ in practice. We investigate a suitable practical structure for AVQ next.

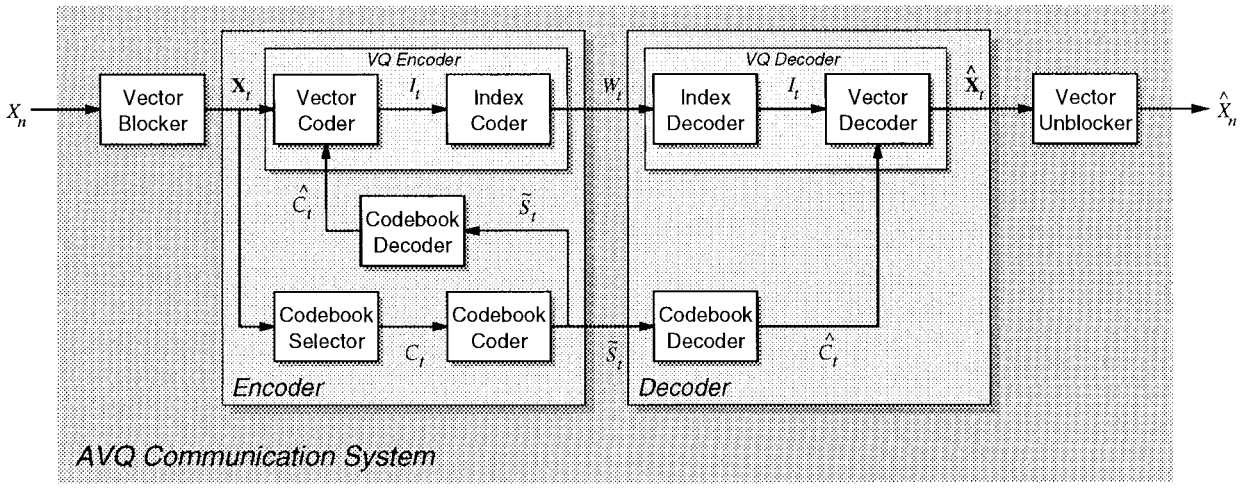


Fig. 1. Model of a communication system using AVQ.

B. Model of Communication Systems Using AVQ

In this section, we address the issues involved in the practical implementation of AVQ algorithms by reviewing a model for communication systems using AVQ. This model reflects structure that is, from our observations, common to practical implementations of AVQ algorithms appearing in prior literature. The brevity of this discussion requires the omission of many relevant details; a more complete development can be found in [17] and [19]. Fig. 1 depicts our AVQ communication-system model consisting of time-varying versions of the VQ encoder and VQ decoder used in nonadaptive VQ systems with the addition of certain components to make the system adaptive; we briefly discuss each of these components below. We concentrate on the encoder here as the operation of the decoder follows naturally from that of the encoder.

The vector coder maps input vectors to codewords of the current approximation, $\hat{\mathcal{C}}_t$, of the local codebook. For most AVQ algorithms, the vector coder is simply a nearest-neighbor mapping with respect to some distortion measure, although a few algorithms [7], [10]–[12] use a more sophisticated mapping based on both rate and distortion. The output of the vector coder, index process I_t, is passed to the index coder. The index coder typically uses some form of entropy coding to produce a coding approaching the first-order entropy of I_t.

The codebook selector identifies a sequence of local codebooks, $\mathcal{C}_t$, given the universal codebook $\mathcal{C}^*$. Often, the codebook selector implements a modified version of traditional VQ-training algorithms such as the generalized Lloyd algorithm [24], Kohonen learning [25], or entropy-constrained VQ (ECVQ) [26]. Finally, the codebook coder describes the contents of each local codebook to the decoder via a coding, $\tilde{S}_t$. This coding is often called "side information" [1], [5], [8] as it is considered information "sent on the side" of the main channel transmitting VQ indices.

In our AVQ communication system, the codebook coder treats $\mathcal{C}_t$ as a random source which it codes to produce output process $\tilde{S}_t$, following the model proposed by Zeger *et al.* [27]. If universal codebook $\mathcal{C}^*$ is a finite set, it is possible for the codebook coder to identify each local codebook as a subset of $\mathcal{C}^*$. In this case, the local codebooks used by the decoder, $\hat{\mathcal{C}}_t$, can be the same as those generated by the codebook selector, $\mathcal{C}_t$. However, lossless description of local codebooks is not possible in the case that $\mathcal{C}^*$ is an infinite set; in this case, the codebook coder must necessarily introduce some distortion so that, in general, $\hat{\mathcal{C}}_t \neq \mathcal{C}_t$. Naturally, the overall distortion performance of the AVQ system will depend on distortion introduced by the vector coder as well as a distortion (the "codebook-mismatch" distortion) due to the fact that both the encoder and the decoder use only an approximation, $\hat{\mathcal{C}}_t$, to the local codebook, $\mathcal{C}_t$, designed by the codebook selector.

This codebook-mismatch effect has been analyzed under the assumptions of a stationary source, a high-resolution vector coder, and an index coder implementing a fixed-length code by Zeger *et al.* [27]. They have determined that, under these assumptions, the overall distortion incurred by an AVQ system can be separated into a distortion associated with the vector coder and local codebook plus the codebook-mismatch distortion. In this case, the design of an AVQ system may be partitioned so that the codebook selector and vector coder are designed independently from the codebook coder.

Because of the complexity associated with the design of efficient codebook coders for infinite universal codebooks, most AVQ literature has not pursued codebook-coder design in depth and has instead focused on the codebook selector. We do likewise here—we describe the codebook selector and vector coder for a new AVQ algorithm, which we compare with algorithms proposed by other authors. In these experimental comparisons, we use the same simple codebook coder for each AVQ technique under consideration and ignore any codebook-mismatch distortion. In this approach, the codebook coder implements a uniform scalar quantizer that describes the local-codebook sequence incrementally by coding each vector added to the local codebook with a fixed number of bits per vector component. Additionally, a set of indices is sent to indicate which old codewords are to be removed from the local codebook. It has been argued that rate-distortion inefficiency due to such a simple scalar-quantizer codebook coder is

negligible if side information accounts for only a small part of the total rate [17]. This simple scalar-quantizer codebook coder will be used in each AVQ algorithm considered in the experimental evaluations conducted later.

III. The Generalized Threshold Replenishment Algorithm

In this section, we describe GTR, a new AVQ algorithm. GTR is an online algorithm that does not require large amounts of batch computation, and it employs cost criteria involving both rate and distortion measures. The GTR algorithm weighs the distortion performance against the cost in rate in both the coding of the current source vector and the updating of the local codebook. In general terms, GTR incorporates a rate-distortion-based cost function similar to the one developed by Lightstone and Mitra [10], [11] into an online framework similar to the AVQ algorithm described by Paul [2] while offering substantial performance improvement over these and other approaches.

Many approaches to AVQ (e.g., [5], [6], [8]–[11]), operate in a batch manner: a large number of source vectors are buffered, and a computational intense codebook selector trains over the source buffer for one or more iterations. GTR, on the other hand, is an online algorithm. As in other online AVQ algorithms (e.g., [2], [4], [12]), the computational load of GTR is spread over time so that a small amount of computation is performed as each source vector enters the algorithm. Source vectors do not need to be buffered, while codebook updates, which can occur at any time, are based on quantities estimated dynamically. As a result, GTR is more amenable to real-time hardware and software implementation than are many other AVQ algorithms based on traditional batch training methods.

In addition, GTR differs from most other AVQ algorithms in that the vector coder and the codebook selector are based on cost criteria involving both rate and distortion measures. AVQ algorithms based on rate-distortion criteria are relatively new. The first such technique was developed by Lightstone and Mitra [10], [11] who applied ECVQ [26] to generate new codewords which update the codebook according to a Lagrangian rate-distortion-based cost criterion. A subsequent approach by Chan and Vetterli [12] invoked a similar ECVQ update technique but included mechanisms for the growing and shrinking of the codebook by adding, splitting, and deleting codewords. Contrary to these approaches, traditional AVQ algorithms (e.g., [5], [6], [8], [9], [13]–[16]) usually focus on the minimization of distortion alone, regardless of the consequences incurred in rate performance. Our GTR algorithm, like the other rate-distortion-based AVQ techniques [10]–[12], weighs distortion performance against associated cost in rate to achieve overall rate-distortion performance superior to that of traditional AVQ algorithms.

The rate-distortion-based cost criteria of the GTR algorithm operate as follows. The codebook selector chooses a codeword from the current local codebook as a potential coding of the current source vector by considering both the distortion between the two vectors and the rate needed to specify the codeword to the decoder. This rate is estimated from the current codeword probabilities, assuming that the index coder implements variable-length entropy coding. Once the winning codeword is chosen, the codebook selector evaluates a decision rule to see if a codebook update would result in a reduction in distortion outweighing the cost in rate associated with the update. If so, the codebook selector replaces a codeword in the local codebook with the current source vector.

Given: initial local codebook, $\mathcal{C}_0$
initial codeword probabilities, $p_0(i)$, for each codeword $\mathbf{c}_i \in \mathcal{C}_0$
rate-distortion parameter, λ
windowing parameter, ω
initial time, $t = 1$

Step 1: Calculate initial codeword lengths of the index coder:

$$l(\gamma_t(\mathbf{c}_i)) = -\log_2 p_{t-1}(i).$$

Step 2: Find the distortions between each codeword $\mathbf{c}_i \in \mathcal{C}_{t-1}$ and $\mathbf{X}_t$:

$$\delta(\mathbf{c}_i) = d(\mathbf{c}_i, \mathbf{X}_t).$$

Step 3: Calculate the cost function for each codeword

$$J(\mathbf{c}_i) = \delta(\mathbf{c}_i) + \lambda \cdot l(\gamma_t(\mathbf{c}_i)).$$

Step 4: Find the winning codeword:

$$\mathbf{c}^* = \arg \min_{\mathbf{c} \in \mathcal{C}_{t-1}} J(\mathbf{c}).$$

Let the index of $\mathbf{c}^*$ be denoted i^*.

Step 5: Calculate the distortion improvement and rate cost of a codebook update, as well as the update cost function:

$$\Delta d = -\delta(\mathbf{c}^*), \qquad \Delta r = l(\mathbf{X}_t),$$
$$\Delta J = \Delta d + \lambda \cdot \Delta r.$$

Step 6: Set $\mathcal{C}_t = \mathcal{C}_{t-1}$. If $\Delta J < 0$, go to Step 6a. Else, go to Step 6b.
Step 6a: Set $\mathbf{c}^* = \mathbf{X}_t$ in $\mathcal{C}_t$. Send to the decoder $\mathbf{X}_t$, entropy-coded index i^*, and a flag indicating a codebook update. Go to Step 7.
Step 6b: Send the entropy-coded index i^* and a flag indicating no codebook update.

Step 7: Estimate the new codeword probabilities:

$$p_t(i) = \begin{cases} [\omega p_{t-1}(i)]/(\omega+1), & i \neq i^*, \\ [\omega p_{t-1}(i)+1]/(\omega+1), & i = i^*. \end{cases}$$

Step 8: Set $t = t + 1$ and go to Step 1.

Fig. 2. Basic GTR algorithm.

To simplify the discussion of this section, we present two versions of the GTR algorithm. In Section III-A, we describe the basic algorithm which lays a foundation for the move-to-front variant of the algorithm to follow in Section III-B. It has been observed that the move-to-front variant has a slight performance advantage over the basic algorithm while being only marginally more computationally complex [17].

A. The Basic Algorithm

The basic variant of the GTR algorithm is outlined in detail in Fig. 2. This algorithm operates as follows. The first sequence of steps determines the codeword "closest" to the current source vector in a rate-distortion sense. This rate-distortion-based nearest-neighbor mapping operates as follows. First, the codebook selector estimates how many bits the index coder would use to code each index as

$$l(\gamma_t(\mathbf{c}_i)) = -\log_2 p_{t-1}(i) \tag{5}$$

where $\gamma_t(\cdot)$ represents the index coder and $p_{t-1}(i)$ are the estimated probabilities of the codewords $\mathbf{c}_i \in \mathcal{C}_{t-1}$. Thus, $l(\gamma_t(\mathbf{c}_i))$ estimates the rate needed to transmit index i to the decoder if codeword $\mathbf{c}_i$ were to be chosen by the vector coder. Then, the distortion, $\delta(\mathbf{c}_i)$, between the current source vector, $\mathbf{X}_t$, and each codeword, $\mathbf{c}_i \in \mathcal{C}_{t-1}$, is calculated. The codebook selector combines these distortions with the previous rate estimates into a cost function using a rate-distortion parameter, λ; i.e., the cost function, $J(\mathbf{c}_i)$, for each $\mathbf{c}_i \in \mathcal{C}_{t-1}$ is

$$J(\mathbf{c}_i) = \delta(\mathbf{c}_i) + \lambda \cdot l(\gamma_t(\mathbf{c}_i)). \tag{6}$$

The codebook selector chooses the codeword, $\mathbf{c}^*$, with the lowest cost function as a potential update vector for the local codebook. We denote the index of $\mathbf{c}^*$ as i^*. One should note that the steps to this point implement the modified nearest-neighbor rule of the well known ECVQ algorithm [26].

In the next steps, the codebook selector decides whether the local codebook needs to be updated by first estimating the improvement in distortion to be gained by a codebook update. If the codebook is updated, the current source vector will be coded with zero distortion. However, if the codebook is not updated, the current source vector will be coded with a distortion of $\delta(\mathbf{c}^*)$. Thus, the estimated distortion improvement due to a codebook update is

$$\Delta d \triangleq -\delta(\mathbf{c}^*). \tag{7}$$

The cost in rate of the codebook update is the amount of side information needed to send $\mathbf{X}_t$ to the decoder

$$\Delta r \triangleq l(\mathbf{X}_t), \tag{8}$$

where $l(\mathbf{X}_t)$ is the number of bits of side information which would be sent to the decoder in the case of a codebook update.

The codebook selector uses the expected distortion improvement and the expected rate cost in a rate-distortion-based update cost function, defined as

$$\Delta J \triangleq \Delta d + \lambda \cdot \Delta r \tag{9}$$

where λ is the rate-distortion parameter given to the GTR algorithm. If $\Delta J < 0$, then the expected improvement in distortion outweighs the expected cost in rate, and the codebook selector inserts $\mathbf{X}_t$ into the local codebook by replacing $\mathbf{c}^*$ with $\mathbf{X}_t$. The codebook coder transmits to the decoder a flag indicating whether the codebook was updated. Additionally, if the codebook was updated, the codebook coder sends $\mathbf{X}_t$ as side information.

The final step of the algorithm is to estimate the new codeword probabilities using a time average. Our method of estimation assumes that the source possesses some degree of local stationarity; i.e., the statistics over a window of ω source vectors are approximately stationary. The algorithm estimates how many of the past ω source vectors have mapped to codeword $\mathbf{c}_i \in \mathcal{C}_{t-1}$ as

$$n_{t-1}(i) = \omega \cdot p_{t-1}(i). \tag{10}$$

When the codebook selector has determined the winning codeword, $\mathbf{c}^*$, for the current source vector, the new counts are calculated as

$$n_t(i) = \begin{cases} n_{t-1}(i), & i \neq i^* \\ n_{t-1}(i) + 1, & i = i^*. \end{cases} \tag{11}$$

The algorithm estimates the new partition probabilities using the new codeword counts; i.e.,

$$p_t(i) = \frac{n_t(i)}{\sum_{\mathbf{c_j} \in \mathcal{C}_t} n_t(j)} \tag{12}$$

for each $\mathbf{c}_i \in \mathcal{C}_t$. Plugging (11) into (12) yields

$$p_t(i) = \begin{cases} [\omega p_{t-1}(i)]/(\omega+1), & i \neq i^* \\ [\omega p_{t-1}(i) + 1]/(\omega+1), & i = i^*. \end{cases} \tag{13}$$

After the codeword probabilities are updated, the index i^* is entropy coded and transmitted to the decoder. The algorithm repeats for the next source vector.

Before continuing to the move-to-front variant, several comments on the parameters of the basic GTR algorithm are in order. The parameter λ given to the algorithm is called the *rate-distortion parameter*, as it controls the tradeoff between rate and distortion within the algorithm. The value of λ affects not only the nearest-neighbor mapping to the current source vector but also the codebook-update decision. Consequently, λ ultimately determines the performance of the algorithm, in terms of rate and distortion. Indeed, varying λ traces out the rate-distortion performance curve of the algorithm. Larger values of λ will focus efforts on minimizing rate over distortion, whereas smaller values of λ will result in performance with a lower distortion and a higher rate.

The *windowing parameter*, ω, controls the relative weighting of the past versus the present in the time-average estimates of the current codeword probabilities. As we have determined that the value of ω is noncritical in the performance of the algorithm [17], we will use $\omega = 100$ throughout the experimental results presented later.

Finally, since the codebook selector uses the current source vector itself to replace codewords in the current local codebook, the universal codebook for the GTR algorithm is the N-dimensional source alphabet, $\mathcal{X}^N$. In the experimental results presented later in Section IV, we consider two types of sources: an artificial nonstationary random process and real image-sequence data. The artificial random process is continuously distributed. Thus, the universal codebook is infinite in this case, and the codebook coder must implement lossy coding of the source vector for a codebook update. For this codebook coder, we simply scalar-quantize each component of the updating source vector to the 9 b. These 9 b are chosen, in advance, to be the nine most significant bits of the dynamic range of the (finite-length) process currently being encoded. Thus, for the artificial random process, the length of the representation of $\mathbf{X}_t$ sent to the decoder is $l(\mathbf{X}_t) = 9N$ bits. In the case of the image-sequence data, which is originally represented with 256 gray levels, the universal codebook is a finite set of 256^N vectors. Thus, the codebook coder is capable of losslessly transmitting the sequence of local codebooks to

Step 6: Set $\mathcal{C}_t = \mathcal{C}_{t-1}$. If $\Delta J < 0$, go to Step 6a. Else, go to Step 6b.
Step 6a: Insert $\mathbf{X}_t$ in the front of $\mathcal{C}_t$. Increment the indices of all the other codewords. Delete the codeword with the highest index. Send to the decoder $\mathbf{X}_t$ and a flag indicating a codebook update. Go to Step 7.
Step 6b: Send the entropy-coded index i^* and a flag indicating no codebook update. Move $\mathbf{c}^*$ to the front of $\mathcal{C}_t$.
Step 7: Estimate the new codeword probabilities. Let K be the size of the local codebook; i.e., $K = |\mathcal{C}_t|$. If $\Delta J \geq 0$, or $i^* = K$, go to Step 7a. Else, go to Step 7b.
Step 7a: Estimate the new codeword probabilities as:

$$p_t(i) = \begin{cases} [\omega p_{t-1}(i)]/(\omega+1), & i \neq i^*, \\ [\omega p_{t-1}(i)+1]/(\omega+1), & i = i^*. \end{cases}$$

Rearrange the indices of the probabilities to match the move-to-front rearrangement of $\mathcal{C}_t$; i.e., move $p_t(i^*)$ to index 1, shifting all the other probabilities. Go to Step 8.
Step 7b: Form the new codeword counts as:

$$n_t(i) = \begin{cases} \omega p_{t-1}(i), & i \neq i^*, K \\ [\omega p_{t-1}(i)+1]/2, & i = i^* \\ n_t(i^*), & i = K. \end{cases}$$

Calculate the new codeword probabilities as:

$$p_t(i) = \frac{n_t(i)}{\sum_{1 \leq j \leq K} n_t(j)}.$$

Rearrange the indices of the probabilities to match the move-to-front rearrangement of $\mathcal{C}_t$; i.e., move $p_t(K)$ to index 1, shifting all the other probabilities.

Fig. 3. Move-to-front GTR algorithm. These steps replace the corresponding steps in the basic GTR algorithm of Fig. 2.

the decoder. There exist several possible implementations of such a lossless codebook coder [11], [17], [27]; the simple technique we use here is to send each updated vector directly. That is, $\mathbf{X}_t$ has $\log_2 256 = 8$ b per vector component, and the length of the representation of $\mathbf{X}_t$ sent to the decoder is $l(\mathbf{X}_t) = 8N$ b. More complicated coding schemes for lossy and lossless codebook coders are discussed in [11], [17], and [27].

B. The Move-to-Front Variant of the Algorithm

In the case of a codebook update in the basic GTR algorithm presented above, the codebook selector adds the current source vector to the local codebook by replacing the codeword $\mathbf{c}^*$ that was determined to be the closest in a rate-distortion sense. In this section, we describe the *move-to-front* GTR algorithm. In this variant of the algorithm, codewords are added to the local codebook in a move-to-front fashion, effectively replacing least-recently-used (LRU) codewords. The move-to-front GTR algorithm replaces steps 6 and 7 of the basic algorithm (Fig. 2) with the steps shown in Fig. 3 and operates as follows.

If the codebook is not updated, the codebook selector simply moves $\mathbf{c}^*$ to the front (index number 1) of the codebook. However, if the codebook is updated, i.e., if $\Delta J < 0$, the codebook selector places current source vector $\mathbf{X}_t$ in the front of $\mathcal{C}_t$. In this case, all the other codewords are shifted to the next highest index, and the one with the highest index (the LRU codeword) is deleted.

As long as the codebook is not updated, the estimation of the new codeword probabilities is the same as in the basic algorithm. However, in the case of codebook update, the move-to-front insertion of $\mathbf{X}_t$ necessitates a modified calculation of the codeword probabilities. In this case, the codebook selector "splits" the probability of partition i^* between the new codeword, $\mathbf{X}_t$, and $\mathbf{c}^*$. Fig. 3 gives the details of how this probability "split" is accomplished. After the new codeword probabilities are calculated, they are rearranged so as to match the move-to-front rearrangement of $\mathcal{C}_t$.

The move-to-front technique is a heuristic that has been shown to aid lossless coding techniques in achieving a lower rate [28]; other similar lossless techniques include [29]. Move-to-front codebook reordering has also been proposed to implement LRU replacement in other AVQ algorithms [3], [4]. The additional computational cost of implementing a move-to-front technique in AVQ algorithms is negligible as the move-to-front process involves mainly simple index shuffling. However, the LRU replacement strategy of a move-to-front AVQ algorithm has the advantage that the index of the LRU codeword does not need to be transmitted to the decoder. In the case of GTR, move-to-front replacement has been observed to contribute a slight improvement in rate-distortion performance over that of the basic algorithm [17]. Consequently, we will use only the move-to-front variant of the GTR algorithm in the experimental evaluations presented in the next section.

IV. Experimental Results

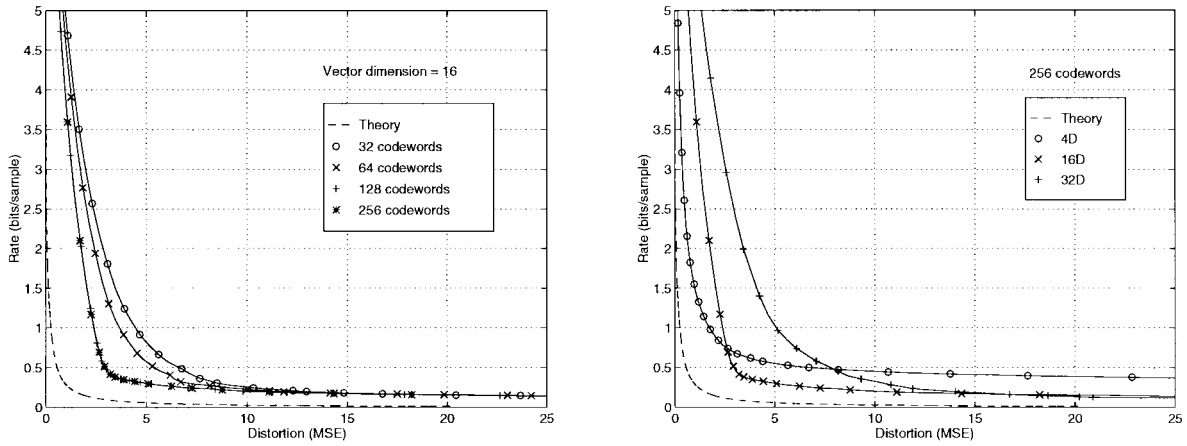
We now survey experimental results obtained for the GTR algorithm. We investigate the rate-distortion performance of GTR on an artificial nonstationary random process as well as on a real image sequence. We make comparisons between GTR, nonadaptive VQ, and other previously published AVQ algorithms.

First, in Section IV-A, we investigate the rate-distortion performance of the GTR algorithm on a Wiener process, one of the few nonstationary random processes with a known rate-distortion function (see [30]). Next, in Section IV-B, we compare the rate-distortion performance of the GTR algorithm against that of the nonadaptive VQ for an image sequence. Finally, in Section IV-C, we compare the performance of GTR to that of other AVQ algorithms on both the Wiener-process and image-sequence data described in the previous two sections.

A. Wiener-Process Results

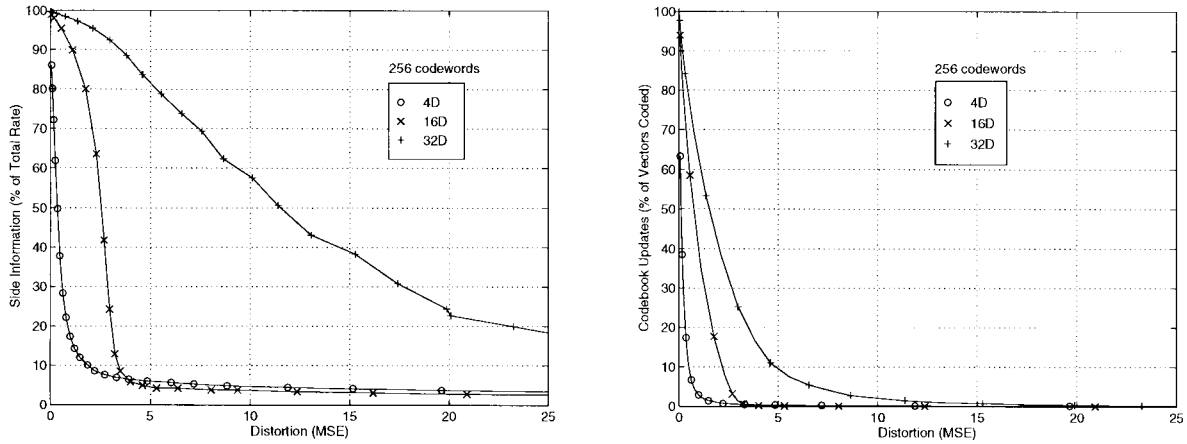
In this section, we investigate the rate-distortion performance of GTR on a Wiener process. Berger [30] has shown that the rate-distortion function of this nonstationary process exists and has obtained an easily evaluated closed-form expression for it. In the results presented here, we evaluate the rate-distortion performance of the GTR algorithm against the theoretic bound given by this expression.

In this section, we consider a Wiener process with variance $\sigma^2 = 1$. All results are averaged over five trials; i.e., we use a testing data set consisting of five different instances of the Wiener process. Each instance is 80 000 samples long. We use a sixth instance of the Wiener process to train an initial local codebook using the generalized Lloyd algorithm (see [24]). We use a uniform scalar quantizer for the codebook



(a) (b)

Fig. 4. Rate-distortion performance of the GTR algorithm on the Wiener process. Each curve is generated from left to right using an increasing, geometric sequence of λ values. For each curve, $\omega = 100$. The theoretic rate-distortion curve for the Wiener process is indicated by the dashed line. (a) Sixteen-dimensional vectors and various local-codebook sizes. (b) Local codebook of 256 codewords and various vector dimensions.



(a) (b)

Fig. 5. Side information for the GTR algorithm using various codeword dimensions. In each plot, we use a local codebook of 256 codewords and a windowing parameter of $\omega = 100$. (a) The amount of side information due to codebook updates expressed as a percentage of the total rate reported in Fig. 4(b) (does not include side information due to flags). (b) The frequency of codebook updates expressed as a percentage of the number of vectors coded.

coder; i.e., for each vector transmitted as side information, the codebook coder sends 9 b per vector component. The total rate reported accounts for this update-vector side information, the bits used as update flags, and the first-order entropy of the index process output from the vector coder; the rate is expressed as the average number of bits per original source symbol. We present results only for the move-to-front variant of the GTR algorithm, and we fix the windowing parameter ω at 100.

In Fig. 4(a), we show the rate-distortion performance of GTR for a fixed vector dimension and various local-codebook sizes. We see from this figure that increasing the local-codebook size yields performance closer to the theoretic rate-distortion curve, particularly in the low-rate, low-distortion area of the curve (the so-called "knee" of the curve). However, the fact that the 128- and 256-codeword curves are coincident suggests that there is a limit to the increase in performance obtainable by increasing the codebook size. We use a local-codebook size of 256 vectors for the remaining results of this section.

In Fig. 4(b), we plot the rate-distortion performance of GTR for a fixed local-codebook size and various vector dimensions. From rate-distortion theory, we would expect that increasing the vector dimension would yield performance increasingly close to the rate-distortion function. However, for practical implementations of AVQ, such as GTR, the burden in rate due to side information varies with vector dimension. As illustrated in Fig. 4(b), when the distortion is high, increasing the vector dimension yields performance increasingly close to the theoretic rate-distortion curve as expected. However, we see from the same figure that, when the distortion is low, the opposite is true; that is, using smaller vector dimensions yields performance closer to the rate-distortion curve. This effect is explained in Fig. 5. Fig. 5(a) plots the amount of side information due to codebook updates as a percentage of the total rate for the corresponding data in Fig. 4(b). In

(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 6. Original image sequence. (a)–(d) Four frames of the Miss America sequence. (e)–(h) Four frames of the garden sequence.

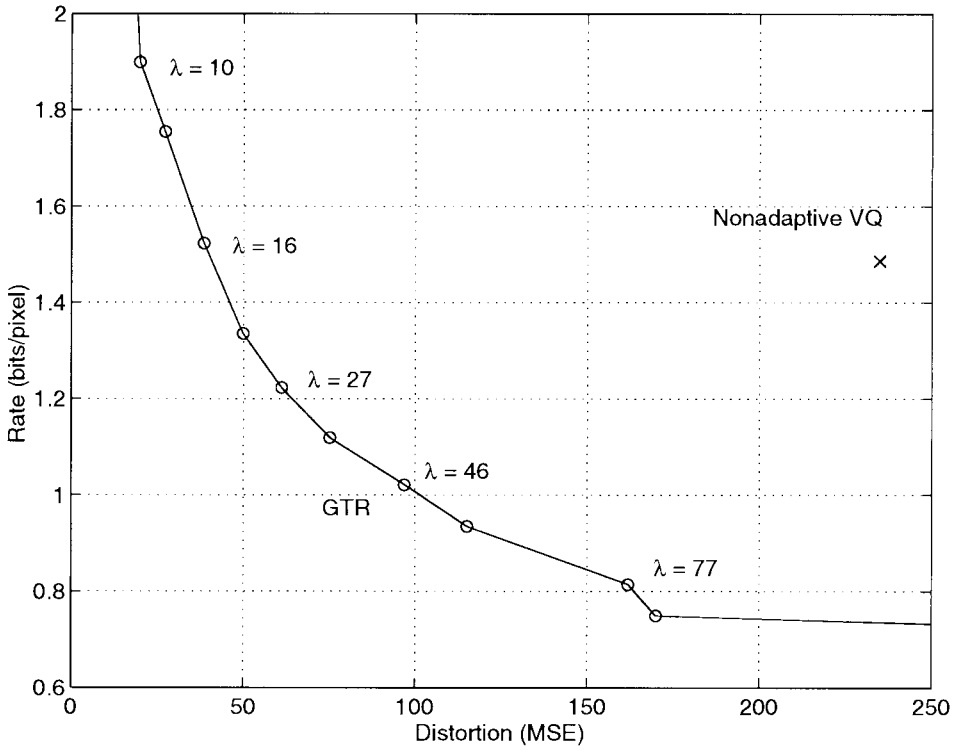


Fig. 7. GTR versus nonadaptive VQ for the image sequence using 4-D vectors (2×2 tiles) and a local codebook of 256 codewords. The GTR algorithm operates at approximately the same rate as the nonadaptive vector quantizer when $\lambda = 16$.

Fig. 5(b), the frequency of codebook updates is plotted as a percentage of the number of vectors coded. From Fig. 5(b), we see that an increasing number of codebook updates are needed to achieve a lower distortion for each vector dimension. However, when the vector dimension is small, each codeword update needs fewer bits. Therefore, for low distortion levels, the side information accounts for less of the total rate when the vector dimension is small than when the vector dimension is large, as verified in Fig. 5(a). The algorithm consequently achieves better rate-distortion performance at low distortion levels with smaller vector dimensions. We now continue to the next section wherein we investigate the performance of GTR on real image-sequence data.

B. Image-Sequence Results

In this section, we compare the rate-distortion performance of the GTR algorithm against that of nonadaptive VQ for the coding of the image sequence shown in Fig. 6. This sequence consists of eight image frames: four frames from the image sequence "Miss America" followed by four frames from the "garden" sequence. Each image is grayscale with 256 levels and has a resolution of 352×240 pixels. To produce an initial local codebook for the GTR algorithm, we use an additional frame from the Miss America sequence as a training data set to the generalized Lloyd algorithm. This initial codebook is also used as the fixed codebook for the nonadaptive-VQ results.

In Fig. 7, we plot the rate-distortion performance of nonadaptive VQ versus that of GTR for the image sequence using a local-codebook size of 256 codewords and four-dimensional (4-D) vectors. For the nonadaptive-VQ result, the VQ codebook used over the entire image sequence is the same initial local codebook used by GTR, and the rate is the first-order entropy of the VQ indices, divided by the vector dimension. For GTR, we vary the parameter λ to obtain the rate-distortion performance curve. We show quantized image sequences produced at the same average rate by GTR and nonadaptive VQ in Figs. 8 and 9, respectively.

We see from Fig. 7 that GTR achieves much better rate-distortion performance than nonadaptive VQ when the performance over the entire process is considered. However, when we visually compare the image quality at equal rates by examining Figs. 8 and 9, we observe that nonadaptive VQ performs slightly better on the first four frames of the sequence, while GTR is substantially better on the last four frames of the sequence. This subjective observation is quantitatively verified in Fig. 10, where we plot the average distortion and rate obtained for each frame. Since the nonadaptive vector quantizer is using a codebook trained on data very similar to that of the first four frames of the sequence, its performance is quite good on those early frames. However, since the nonadaptive-VQ codebook has very few codewords suited for coding the latter frames of the sequence, poor distortion performance is observed for the nonadaptive vector quantizer on these latter frames. On the other hand, the adaptive nature of GTR allows it to increase the rate on the latter frames in order to maintain roughly even distortion performance over the entire sequence. Additionally, we observe that GTR preserves edges and other areas of high detail much better than nonadaptive VQ, as it is these areas that are likely to induce a codebook update within the GTR algorithm. Since these areas, which compose a majority of the regions in the latter frames of the sequence, are crucial to perceptual image quality, not only

Fig. 8. Quantized image sequence for GTR for 4-D vectors (2×2 tiles) and a local codebook of 256 codewords ($\lambda = 16$, MSE = 38.6, rate = 1.522 b/pixel).

Fig. 9. Quantized image sequence for nonadaptive VQ for 4-D vectors (2×2 tiles) and a local codebook of 256 codewords (MSE = 234.9, rate = 1.487 b/pixel).

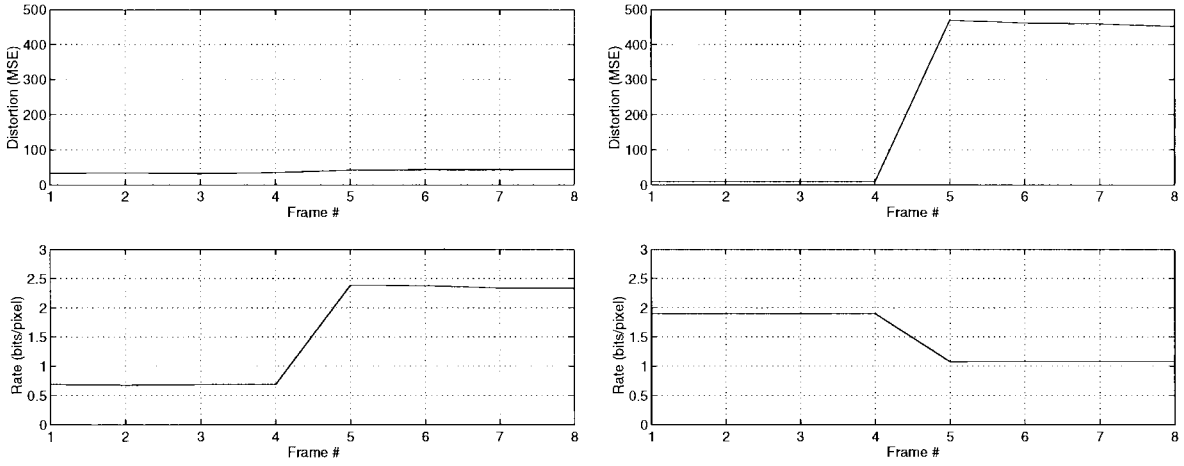


Fig. 10. Average distortion and rate per frame of the image sequence for 4-D vectors (2 × 2 tiles) and a local codebook of 256 codewords. (a) GTR algorithm ($\lambda = 16$). (b) Nonadaptive VQ.

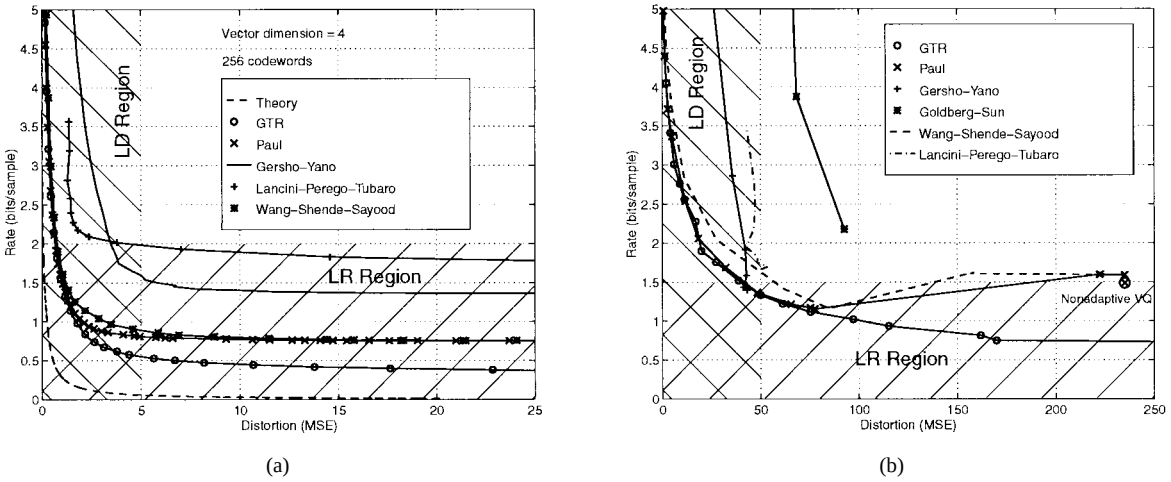


Fig. 11. Rate-distortion performance for prominent AVQ algorithms [2], [4]–[6], [8] using 4-D vectors and a local-codebook size of 256 vectors. The LD and LR regions are indicated with cross hatching. (a) Wiener process, the theoretic rate-distortion function is indicated by the dashed line. (b) Image sequence, the symbol × represents the operation point of the nonadaptive vector quantizer for the same data.

does GTR have a much better empirically measured average distortion, it also achieves better subjective performance on the latter frames of the sequence when compared to the nonadaptive vector quantizer operating at the same rate.

C. Comparisons to Prior AVQ Algorithms

We now compare the performance of GTR to that of several previously published AVQ algorithms [2], [4]–[6], [8]. We note that, although numerous AVQ algorithms have appeared in previous literature, many are quite similar from a theoretic perspective; consequently, we limit discussion here to only those prior algorithms that have either historical or innovative prominence in AVQ literature.

In the results of this section, we plot performance curves on the rate-distortion plane. To aid in characterizing general performance properties, it is convenient to identify two regions of the rate-distortion plane that are of particular interest: the low-rate (LR) region and the low-distortion (LD) region. The first of these regions corresponds to high-compression performance; the natural tradeoff between rate and distortion usually implies a high distortion for this LR region. In general, LR-region performance is of importance in applications whose available rate is severely limited; this is the case for many practical applications of current interest, such as network and wireless communications. The second region of interest is the LD region, which corresponds to high-fidelity performance. Generally, performance in this region will require a high rate. Although LR performance is usually considered more important, LD performance is of interest in certain scientific and medical applications that perform measurements and calculations on coded data and, consequently, need to maintain the integrity of the original data source. We note that we must establish boundaries for the LR and LD regions subjectively for each individual application.

We first consider performance on the Wiener process from Section IV-A. In Fig. 11(a), we show the rate-distortion performance curves for GTR and several prominent AVQ algorithms [2], [4]–[6] on this Wiener-process data. Although

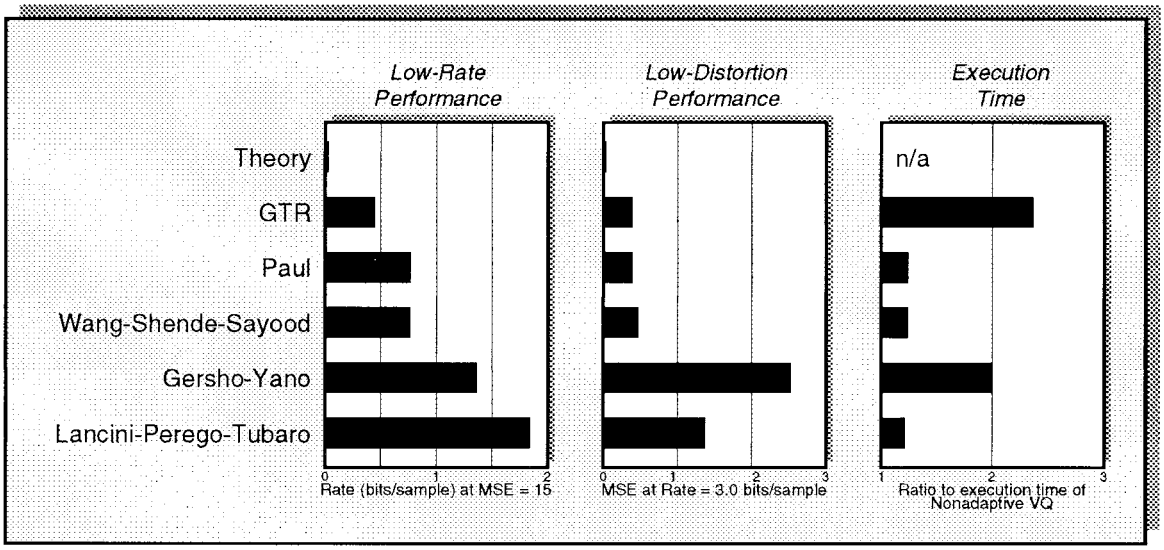


Fig. 12. Summary of experimental results for the Wiener process. For the LR performance, we plot the rates from Fig. 11(a) for each algorithm for a MSE distortion of 15. For the LD performance, we plot the MSE distortions from Fig. 11(a) for each algorithm for a rate of 3.0 b/sample. For the execution time, we plot the cumulative processing time for each algorithm divided by that of nonadaptive VQ; the parameters for each algorithm were adjusted to yield performance on the "knee" of the corresponding rate-distortion curve in Fig. 11(a). In each plot, the smaller the bar, the better is the performance.

performances for other vector dimensions and codebook sizes were investigated (see [17]), the only results considered here are for 4-D vectors and a local-codebook size of 256 codewords. Fig. 12 summarizes the LR and LD performance associated with each of the AVQ algorithms under consideration. To determine LR performance, we select a value of distortion for which all the rate-distortion curves in Fig. 11(a) lie in the LR region. We then measure the rate of each algorithm for that value of distortion and plot it in Fig. 12 as the measure of LR performance. Similarly for LD performance, we choose a rate for which the rate-distortion curves all lie in the LD region and then measure the distortion of each algorithm for that rate. On examining Fig. 12, we see that GTR achieves performance better than that of the other AVQ algorithms in both the LR and LD regions. In fact, in the LR region, the GTR algorithm achieves performance almost 50% better than that of the next best algorithms (the Paul [2] and Wang–Shende–Sayood [4] algorithms). In the LD region, GTR achieves performance nearly equivalent to that of the Paul and Wang–Shende–Sayood algorithms, but substantially better than that of the other two algorithms considered.

Superior performance is also observed for GTR on the image sequence from Section IV-B. In Fig. 11(b), we show the rate-distortion performance curves for GTR and several prominent AVQ algorithms [2], [4]–[6], [8] on this image-sequence data. We have also indicated on the plot LR and LD regions for this data. We see that most of the AVQ algorithms have distortion performance significantly better than that of nonadaptive VQ when compared at the same rate. That is, at a rate equal to that of the nonadaptive vector quantizer (about 1.5 b/pixel), most of the AVQ algorithms achieve an MSE in the LD region (MSE of 50 or less), which results in very little visual distinction between their quantized images. However, all the AVQ algorithms achieve substantially less distortion than nonadaptive VQ at this rate.

In Fig. 11(b), more distinction between the AVQ algorithms is observed as operation moves to lower rates. Particularly, several algorithms were unable to achieve rates in the LR region (below about 1.5 b/pixel). Of those algorithms that were able to produce a coding at a rate below 1.25 b/pixel, only GTR was able to maintain a monotonic decrease in rate for increasing distortion. As a consequence, GTR was the only algorithm to achieve a coding at a rate less than 1.0 b/pixel.

It is clear from Fig. 11(b) that conclusions similar to those illustrated in Fig. 12 may also be drawn for the image-sequence data. That is, in general, GTR achieves LD performance comparable to that of the better algorithms in that region. More importantly however, the LR performance of GTR is consistently superior to that of the other algorithms.

A comparison of execution times of AVQ algorithms is included in Fig. 12. These times are presented as ratios to the execution time of nonadaptive VQ on the same data (the Wiener-process data). These execution-time figures are intended to give merely a rough estimate of the relative speed of the algorithms rather than a thorough complexity comparison as times will vary somewhat for different local-codebook sizes, vector dimensions, and algorithm-parameter values. From Fig. 12, we see that GTR is just over twice as costly in execution time as nonadaptive VQ.

Before making some concluding remarks in the next section, we note that two prominent AVQ algorithms, those by Lightstone and Mitra [10], [11] and Chan and Vetterli [12], were omitted from the experimental comparisons of this section. The reason for the omission of the Lightstone–Mitra algorithm was that this algorithm suffered from "dormant" codewords. This effect, due to the batch application of ECVQ training, has a tendency to reduce the size of the local codebook to a single vector because of the accumulation of "empty" partition regions. As the Lightstone–Mitra algorithm provides no method for the "reactivation" of these dormant codewords,

its performance has not been competitive with that of the other AVQ algorithms in our simulations [17]. Consequently, results for the Lightstone–Mitra algorithm were not considered here. Chan and Vetterli [12] introduced mechanisms for adding, splitting, and deleting codewords to an ECVQ-based algorithm similar to that of Lightstone and Mitra. These additional components are a promising solution to the dormant-codeword problem faced by the Lightstone–Mitra algorithm; however, the brevity of [12] necessitated the omission of many relevant implementation details. Thus, an implementation of the Chan–Vetterli algorithm, as well as its experimental evaluation, was not possible for our investigations.

V. Conclusions

In this paper, we have presented GTR, a new AVQ algorithm. GTR differs from other AVQ algorithms in that, rather than requiring large amounts of batch computation and sizeable source buffering, GTR is an online algorithm. GTR distributes its computational load over time so that a small amount of computation is performed for each source vector. GTR is consequently more amenable to real-time hardware and software implementation than are many other AVQ algorithms. In addition, GTR is one of the few AVQ algorithms to employ cost criteria involving both rate and distortion. As a result, GTR achieves rate-distortion performance superior to that of other AVQ algorithms, the majority of which focus on the minimization of distortion alone regardless to resulting consequences in rate. In particular, we have seen that GTR consistently achieves superior performance for low-rate coding.

We have presented here a body of experimental results investigating the performance of GTR on a real image sequence as well as on a Wiener process, an artificial nonstationary random process. These experimental results included comparisons between GTR, nonadaptive VQ, other AVQ algorithms, and theoretic bounds (i.e., the rate-distortion function of the Wiener process). For the Wiener process, we saw that GTR achieved rate-distortion performance closer to the theoretic rate-distortion function than did other AVQ algorithms. Particularly superior performance in the low-rate, or LR, region of the rate-distortion plane was observed. Similar results were obtained for the image-sequence data; the performance of GTR consistently surpassed that of other AVQ algorithms by a wide margin in the LR region. In fact, of the AVQ algorithms considered here, only GTR was able to maintain a monotonic decrease in rate for increasing distortion in the LR region for the image sequence.

The performance results reported here indicate that GTR has significant potential in source-coding applications. However, several issues in the algorithm remain open for further investigation; we briefly mention a few of these now as potential topics for further research. First, GTR, like all the other AVQ algorithms under consideration in this paper, requires the specification of several parameters in advance. In particular, for GTR, the value of the rate-distortion parameter λ determines the specific operating point on the algorithm's rate-distortion performance curve. Currently, one must select λ individually for each application in advance of coding. There would be considerable use for techniques to provide online, dynamic estimations for an "optimal" setting for this parameter. For example, it would be useful to have an online method of adjusting λ so as to automatically choose a particular operating point, such as operation in the LR region or on the knee of the rate-distortion curve. Other AVQ algorithms could also benefit from similar dynamic parameter estimations.

Second, it would be useful to have some measure of local stationarity since GTR, like most AVQ algorithms, relies, to some degree, on an assumption of slowly varying source statistics. An estimation of the length of time over which source statistics are approximately stationary would have utility in dynamic estimation methods for λ as mentioned above. More immediately, it would serve as a good basis for the static setting of the windowing parameter ω.

Finally, we recommend, for some applications, a more thorough consideration of the tradeoff involved between the codebook coder and the vector coder. As mentioned in Section II-B, it has been argued that the simple scalar quantizer we use for the codebook coder of GTR is sufficient when side information accounts for only a small part of the total rate [17]. As illustrated in Fig. 5(a), this assumption appears to hold for LR coding. However, because performance in the LD region appears to require a sizeable amount of side information, the investigation of other, more complex codebook coders is perhaps warranted for GTR systems designed to operate in this region.

As multimedia applications such as video-on-demand and teleconferencing gain in prevalence, they are expected to increasingly burden available communication resources. Future visual applications will require fast, online coding algorithms amenable to real-time hardware as well as software implementation. Additionally, as there is increasing interest in providing real-time communication over network and wireless channels, new coding techniques will be expected to deliver low-rate performance suited to the severe rate constraints inherent to these asynchronous channels. Because of its online nature and superior low-rate performance, the GTR algorithm proposed in this paper has significant potential for the incorporation of AVQ into practical, low-rate coding techniques at the heart of future communication systems.

Acknowledgment

The author would like to thank Dr. S. C. Ahalt for his invaluable support, both academic and otherwise, which has made this work possible.

References

[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression,* Boston, MA: Kluwer, 1992.

[2] D. B. Paul, "A 500–800 bps adaptive vector quantization vocoder using a perceptually motivated distance measure," in *Conf. Rec. IEEE Globecom,* 1982, pp. 1079–1082.

[3] R. M. Goodman, B. Gupta, and M. Sayano, "Neural network implementation of adaptive vector quantization for image compression," Tech. Rep. Department of Electrical Engineering, California Inst. Technol., Pasadena, CA, 1991.

[4] X. Wang, S. Shende, and K. Sayood, "Online compression of video sequences using adaptive VQ codebooks," in *Proc. IEEE Data Com-*

pression Conf., J. A. Storer and M. Cohn, Eds, Snowbird, UT, 1994, pp. 185–194.
[5] A. Gersho and M. Yano, "Adaptive vector quantization by progressive codevector replacement," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, May 1985, pp. 133–136.
[6] R. Lancini, F. Perego, and S. Tubaro, "Neural network approach for adaptive vector quantization of images," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, San Francisco, CA, Mar. 1992, pp. 389–392.
[7] J. E. Fowler and S. C. Ahalt, "Adaptive vector quantization using generalized threshold replenishment," in *Proc. IEEE Data Compression Conf.*, J. A. Storer and M. Cohn, Eds. Snowbird, UT, Mar. 1997, pp. 317–326.
[8] M. Goldberg and H. Sun, "Image sequence coding using vector quantization," *IEEE Trans. Commun.*, vol. COMM-34, no. 7, pp. 703–710, July 1986.
[9] ———, "Frame adaptive vector quantization for image sequence coding," *IEEE Trans. Commun.*, vol. 36, pp. 629–635, May 1988.
[10] M. Lightstone and S. K. Mitra, "Adaptive vector quantization for image coding in an entropy-constrained framework," in *Proc. Int. Conf. Image Processing*, Austin, TX, Nov. 1994, vol. 1, pp. 618–622.
[11] ———, "Image-adaptive vector quantization in an entropy-constrained framework," *IEEE Trans. Image Processing*, vol. 6, pp. 441–450, Mar. 1997.
[12] C. Chan and M. Vetterli, "Lossy compression of individual signals based on string matching and one pass codebook design," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Detroit, MI, May 1995, pp. 2491–2494.
[13] N. M. Nasrabadi and Y. Feng, "A dynamic finite-state vector quantization scheme," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Albuquerque, NM, Apr. 1990, pp. 2261–2264.
[14] T.-C. Lee and A. M. Peterson, "Adaptive vector quantization using a self-development neural network," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 1458–1471, Oct. 1990.
[15] O. T.-C. Chen, B. J. Sheu, and Z. Zhang, "An adaptive vector quantizer based on the gold-washing method for image compression," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 143–157, Apr. 1994.
[16] R.-F. Chang, W.-T. Chen, and J.-S. Wang, "Image sequence coding using adaptive tree-structured vector quantization with multipath searching," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Toronto, Ont., Canada, May 1991, pp. 2281–2284.
[17] J. E. Fowler, "Adaptive vector quantization for the coding of nonstationary sources," Ph.D. dissertation, The Ohio State Univ., 1996.
[18] ———, "A survey of adaptive vector quantization—Part I: A unifying structure," IPS Lab. Tech. Rep. TR-97-01, The Ohio State Univ., Mar. 1997.
[19] J. E. Fowler and S. C. Ahalt, "A survey of adaptive vector quantization—Part II: Classification and comparison of algorithms," The Ohio State Univ., IPS Lab. Tech. Rep. TR-97-02, Mar. 1997.
[20] T. Berger, *Rate Distortion Theory*. Englewood Cliffs, NJ: Prentice-Hall 1971.
[21] D. L. Neuhoff, R. M. Gray, and L. D. Davisson, "Fixed rate universal block source coding with a fidelity criterion," *IEEE Trans. Inform. Theory*, vol. 21, pp. 511–523, Sept. 1975.
[22] Z. Zhang and V. K. Wei, "An on-line universal lossy data compression algorithm via continuous codebook refinement—Part I: Basic results," *IEEE Trans. Inform. Theory*, vol. 42, pp. 803–821, May 1996.
[23] P. A. Chou, M. Effros, and R. M. Gray, "A vector quantization approach to universal noiseless coding and quantization," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1109–1138, July 1996.
[24] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, Jan. 1980.
[25] T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed. Berlin, Germany: Springer-Verlag, 1988.
[26] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-constrained vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 31–42, Jan. 1989.
[27] K. Zeger, A. Bist, and T. Linder, "Universal source coding with codebook transmission," *IEEE Trans. Commun.*, vol. 42, pp. 336–346, Feb./Mar./Apr. 1994.
[28] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A locally adaptive data compression scheme," *Commun. ACM*, vol. 29, pp. 320–330, Apr. 1986.
[29] J. Sayir and T. Ernst, "Ordering the probabilities of an unknown discrete memoryless source," in *Proc. IEEE Int. Symp. Information Theory*, Ulm, Germany, 1997, to be published.
[30] T. Berger, "Information rates of Wiener processes," *IEEE Trans. Inform. Theory*, vol. 16, pp. 134–139, Mar. 1970.

James E. Fowler (S'91–M'96) received the B.S. degree in computer and information science engineering and the M.S. and Ph.D. degrees in electrical engineering in 1990, 1992, and 1996, respectively, all from The Ohio State University.

In 1995, he was an Intern Researcher at AT&T Labs in Holmdel, NJ, and, from January to July 1997, he held an NSF-sponsored postdoctoral assignment at the Université de Nice, Sophia Antipolis, France. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, Mississippi State University, Starkville, MS. His research interest include data-compression algorithms, video-coding algorithms and hardware systems, adaptive vector quantization, information theory, and neural-network algorithms for signal processing.