VARIABLE FANOUT TRIMMED TREE-STRUCTURED VECTOR QUANTIZATION FOR MULTIRATE CHANNELS

BY

SHAWN MICHAEL HERMAN

B.S., University of Illinois, 1994

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1996

Urbana, Illinois

ABSTRACT

Data compression attempts to reduce the amount of information needed to recreate a signal. A source coder is a data compression system which attempts to maximize the quality of the signal reproduction for a fixed transmission rate or minimize the transmission rate for a fixed reproduction quality. Source coders are designed by exploiting the statistics of their signal sources. A vector quantizer can be used as a source coder.

In this thesis, a generalized pruning technique called trimming is introduced which improves upon the performance of pruned tree-structured vector quantization. The algorithm designs a large balanced nonbinary tree and trims it back to create useful lower-rate subtrees. Trimming differs from pruning in that it does not necessarily remove all descendants of the node being trimmed. However, appropriate codebook replacement is performed during each trim so that all codevectors within the trimmed subtree represent the centroids of their encoding cells.

With trimming replacing pruning, the generalized BFOS algorithm of Chou, Lookabaugh, and Gray still identifies those subtrees which lie on the lower convex hull of the initial tree's operational distortion-rate function. It is shown experimentally that this new convex hull is often *lower* than the one found using pruning. In this sense, the new algorithm outperforms pruned tree-structured vector quantization. Both algorithms are allowed to time share at rates where no subtrees are located.

To design the initial balanced nonbinary tree, it is sufficient to specify the number of children each tree node will have. However, a constraint is imposed so that all nodes located at the same depth within the initial tree have the same number of children. For a given maximum rate constraint, there are clearly many

initial tree structures which could be used. Specifying the initial tree structure is an extra degree of freedom not found with binary pruned tree-structured vector quantization. The new algorithm identifies the nonbinary tree which is optimal with respect to the transmission statistics of a multirate channel. Experimental results are supplied which demonstrate this performance.

ACKNOWLEDGEMENTS

I thank my advisor, Professor Kenneth Zeger, for guiding my research during the past two years. His numerous helpful suggestions made a vast improvement in the quality of this work and my abilities as a practicing engineer.

This work was supported in part by a National Science Foundation Graduate Research Fellowship.

TABLE OF CONTENTS

Page

1 INTRODUCTION 1

2 VECTOR QUANTIZATION FUNDAMENTALS 5

2.1 Tree-Structured Vector Quantization 7

2.2 Pruned Tree-Structured Vector Quantization 9

3 THE VARIABLE FANOUT TRIMMED TSVQ ALGORITHM 14

3.1 Design of the Initial Variable Fanout TSVQ 15

3.2 Trimming of Variable Fanout TSVQ 17

3.3 Storage of Variable Fanout Trimmed TSVQ 19

3.4 Choosing the Fanout Vector to Minimize $\bar{D}$ 22

4 EXPERIMENTAL RESULTS 23

4.1 Lower Convex Hull Results 23

4.2 Minimizing $\bar{D}$ for a Known f_C 27

5 CONCLUSION AND FUTURE RESEARCH 32

APPENDIX A PSEUDOCODE FOR OPTIMAL TRIMMING 33

APPENDIX B PROOF OF NESTING PROPERTY 36

REFERENCES 38

LIST OF FIGURES

Figure Page

1: A block diagram of an N-point vector quantizer. 5

2: Example of a balanced binary tree with maximum depth 4. 7

3: Example of an unbalanced binary tree. 9

4: Node t is pruned by removing all of its descendants (dashed lines). 10

5: The lower convex hull of the operational distortion-rate function $\hat{D}_T(R)$. 12

6: Example of a variable fanout TSVQ; $t_{i,j}$ denotes the j^{th} node at depth-i within the tree. 15

7: Example of a subtree for variable fanout trimmed TSVQ. The dashed lines indicate a possible way to trim at node $t_{1,2}$. 18

8: Trimmed and pruned-only lower convex hulls using initial tree with fanout vector $\langle 2, 2, 2 \rangle$. Squares are trimmed subtrees; crosses are pruned subtrees. ... 24

9: Enlargement of range 4–6 bits/vector from Figure 8. 25

10: Lower convex hulls of all four algorithms. 26

11: The product of available rate distribution and MSE for test data. The channel density is $\mathcal{N}(5.5, 0.01)$. 29

12: Interpretation of (33) and (34) on the lower convex hull of the operational distortion-rate function for T. 37

1 INTRODUCTION

In recent years, compression of digital images has found applications in areas such as television transmission, video conferencing, relay of sensing images obtained from satellites, and storage for video databases [1]. Vector quantization techniques, as applied to image coding, attempt to exploit the statistical dependence between neighboring pixels in order to reduce the bit rate for either transmission or storage. These efforts are motivated by Shannon's rate-distortion theory, which states that superior performance is theoretically achievable by coding vectors instead of scalars.

Vector quantization (VQ) techniques can be grouped into several broad categories and their hybrids [1]. Spatial VQ blocks an image into vectors and then replaces each using a fixed set of reproduction vectors according to some fidelity criterion. Predictive VQ takes advantage of intervector dependencies by predicting future vectors and then coding only the residual error. Transform VQ attempts to reduce correlation between vector components through a suitable linear transformation in addition to allocating bits on the basis of human perceptual importance. Within each of these categories, there are adaptive versions of these algorithms which modify VQ parameters based on the short-term statistics of the source. Of these techniques, spatial VQ is generally the simplest to implement because it is memoryless and has a table lookup for its decoder. Spatial VQs are often evaluated on the basis of arithmetic complexity and codebook storage in addition to compression performance.

The simplest spatial VQ technique is full search VQ (FSVQ). Input vectors are encoded by the codebook vector that minimizes the chosen fidelity criterion. To find this best reproduction vector, a full (or unstructured) search is made through the entire codebook. For this reason, FSVQ has high arithmetic complexity. Tree-structred VQ (TSVQ) was introduced in [2] as a means of reducing arithmetic complexity by using a decision tree to create a search which is logarithmic in codebook size instead of linear. Pruned tree-structured VQ (PTSVQ)

was introduced in [3] as a means of improving the performance of TSVQ. All three of these techniques will be discussed in further detail in Chapter 2.

In addition to structuring the search at the VQ encoder, trees can also be used to represent a variable block-size segmentation of an image. In [4], quad-trees were used to vary the size of the pixel vectors based on the local activity within the image. Small blocks were used for regions with high levels of activity (as determined by the variance) while large regions were used for regions of low activity. Separate quad-trees were used for both mean-subtraction and subsequent detail segmentation. It was found that at low bit rates, edges within the images were reproduced perceptually much better than with FSVQ. In [5], a tree structure was created that varied the dimension of its vectors with depth from large to small. The authors used the algorithm introduced in [3] to adjust the average rate and performance of the encoding tree.

For high quality image reproduction, very large reproduction codebooks are often needed. As such, special attention has been paid in the literature to reducing storage without sacrificing performance. In [6], a spatial VQ technique was introduced which designs M codebooks for N different sources where $M < N$. The design algorithm is based on the generalization of necessary conditions for FSVQ optimality and can be used to create a cascade of encoders in which each stage is used to select the subsequent codebook. In [7], TSVQ storage was reduced by quantizing the reproduction vectors used to guide the search. The secondary VQ also had to be stored in order to generate the initial TSVQ. Although this method increases complexity and decreases performance, it was found that significant storage reductions were possible at modest costs.

If arithmetic complexity and codebook size are not constrained, entropy-constrained VQ (ECVQ) can typically perform better than all techniques mentioned thus far [8]. The ECVQ design algorithm minimizes both the entropy of the VQ output and the average distortion, because the ECVQ is followed by an entropy coder which yields even further rate compres-

sion. To reduce arithmetic complexity, various tree structures that minimize some form of entropy have been introduced in the literature. In [3], entropy-pruned TSVQ (EPTSVQ) was introduced as a method of generating TSVQs which traded quantization distortion for output entropy. In [9], a technique was introduced which redesigned the leaf nodes of a PTSVQ under an entropy constraint. Although this procedure generates trees which satisfy necessary conditions for optimality at the leaves, the important nesting property of PTSVQ to be discussed in Section 2.2 is lost. In [10], both entropy and storage constraints were placed upon each level of TSVQ design, resulting in a tree of ECVQs. Once again, the important nesting property of PTSVQ is sacrificed to improve performance.

The nesting property of PTSVQ is important because it allows efficient access to many TSVQs through the storage of one large initial tree. This property is most useful when images must be transmitted over channels whose transmission rates vary with time. Additionally, the PTSVQ's tree structure permits images to be sent progressively due to the successive approximation character of its interior codevectors. *Progressive* transmission systems produce estimations of the fully decoded signal as data are being received. However, no extra data are sent to create these intermediate reproductions. Instead, the indices generated by the encoder specify the appropriate approximations for each input block. Progressive transmission, as it applies to TSVQ, is discussed further in Section 2.1.

Because PTSVQ has been shown experimentally to have rate-distortion performance close to full search vector quantization over a substantial range of bit rates [3], in applications where variable rate coding is acceptable, PTSVQ's reduced encoding complexity makes it an attractive spatial VQ technique. However, in a packet network environment where bandwidth availability fluctuates in time, it is of interest to have a data compression system that works well when averaged over the available transmission rates. This problem has not previously been addressed with PTSVQ.

In this thesis, I introduce a generalized pruning technique called "trimming" which improves the performance of PTSVQ. This new algorithm achieves a lower mean-square encoding error than PTSVQ while retaining a tree structure for efficient encoding and progressive transmission. Moreover, the tree structure is optimized with regard to a given histogram of available transmission rates on the channel. The interior nodes of the encoding tree are allowed to have fanouts greater than two. For a fixed number of bits spent per input vector, this technique can improve the signal-to-noise ratio at the cost of less flexible progressivity since the nodes at each interior level within a TSVQ can be used to create an intermediate signal reproduction. The main idea is to first grow a large balanced tree with variable fanouts and then trim it back to create useful subtrees.

The idea of using variable fanout trees is not new [11], and, in fact, the process of locating all pruned subtrees which lie on the lower convex hull of the operational rate-distortion function for nonbinary TSVQs was also previously studied in [3]. However, trimming, when applied to a variable fanout TSVQ, generates additional trees (i.e., rate-distortion pairs), some of which often lie below the lower convex hull as found by the generalized BFOS algorithm.

In Chapter 2, the fundamentals of FSVQ, TSVQ, and PTSVQ are presented, with emphasis on their interrelations. In Chapter 3, the variable fanout trimmed TSVQ algorithm is introduced, and its storage requirements and application to multirate channels are discussed. In Chapter 4, experimental results which quantify the performance gain achievable using variable fanout trimmed TSVQ are provided. Chapter 5 concludes the thesis.

2 VECTOR QUANTIZATION FUNDAMENTALS

A *vector quantizer* is a mapping from $\mathcal{R}^k$ into a finite set $C_N \subset \mathcal{R}^k$ containing N *codevectors*. A vector quantizer Q is entirely specified by its partition *cells* $R_i = \{\mathbf{x} \in \mathcal{R}^k : Q(\mathbf{x}) = \mathbf{y}_i\}$ for $i \in \{0, 1, \ldots, N-1\}$ and its *codebook* $C_N = \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{N-1}\}$. The partition determines the encoding rule for Q while the codebook determines the decoding rule. The input to Q is a k-dimensional random vector $\mathbf{X}$ with probability density function $f_{\mathbf{X}}$; thus, $Q(\mathbf{X})$ is also a random vector. Figure 1 is a useful representation of a vector quantizer. The indices generated by the encoder are either stored or transmitted for retrieval by the decoder. Since the exact value of the source random vector $\mathbf{X}$ is not reproduced at the decoder, vector quantization is a *lossy* compression technique.

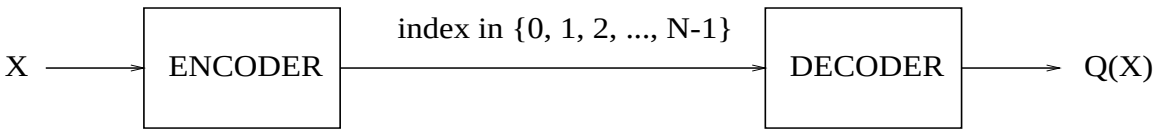

Figure 1: A block diagram of an N-point vector quantizer.

An important measure of performance for a vector quantizer which takes into account the statistical properties of the input random vector $\mathbf{X}$ is the average distortion. Let $d(\mathbf{x}, \mathbf{y})$ denote a nonnegative, real-valued cost incurred by replacing $\mathbf{x}$ by the codevector $\mathbf{y}$. In all that follows, the squared error, defined by

$$d(\mathbf{x}, \mathbf{y}) \;=\; \|\mathbf{x} - \mathbf{y}\|^2 \;=\; \sum_{i=0}^{k-1} (x_i - y_i)^2 \;, \tag{1}$$

is used as the cost function. With this d, the average distortion for Q is the mean square error (MSE) defined as

$$D_Q \;=\; \int_{\mathcal{R}^k} d(\mathbf{x}, Q(\mathbf{x}))\, f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} \;=\; \int_{\mathcal{R}^k} \|\mathbf{x} - Q(\mathbf{x})\|^2\, f_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} \;. \tag{2}$$

In order for D_Q to be minimized for a given source $\mathbf{X}$, it is necessary (but not sufficient) that Q satisify the following two conditions [12]:

$$\textbf{Nearest Neighbor Condition:} \quad Q(\mathbf{x}) = \mathbf{y}_i \text{ only if } d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j) \quad \forall j \tag{3}$$

$$\textbf{Centroid Condition:} \quad \mathbf{y}_i = E[\, \mathbf{X} \mid \mathbf{X} \in R_i \,] \,. \tag{4}$$

The Generalized Lloyd Algorithm (GLA) is an iterative descent technique which generates vector quantizers which have *locally* optimal values of average distortion [12]. The GLA repeatedly improves the VQ partition by enforcing the nearest neighbor condition (3) and the codebook by enforcing the centroid condition (4). Each application of the Lloyd iteration either reduces D_Q or leaves it unchanged.

The VQ technique described so far is typically referred to as full search VQ because the minimum distortion codevector is determined by computing $d(\mathbf{x}, \mathbf{y})$ for each $\mathbf{y} \in C_N$. In Section 2.1, a VQ algorithm is considered which uses a structured codebook to avoid an exhaustive search of C_N. Upon determining the optimum codevector, the encoder generates the corresponding index. For FSVQ, all indices are specified by the same number of bits (assuming $N = 2^m$ for some positive integer m). Thus, FSVQ generates a fixed rate code. If r denotes the number of bits transmitted per vector component (i.e., the rate), then each FSVQ index is specified by kr bits. Since the FSVQ encoder performs an exhaustive search over C_N for each source vector, FSVQ has encoding complexity on the order of $2^{kr} = N$.

Increasing k allows a quantizer to exploit dependencies between vector components and to take advantage of the vector partition's shape-filling efficiency [13]. In [14], it was shown that, for N large, D_Q is asymptotically

$$D_Q(N; k, 2) \approx C(k, 2)\, N^{-2/k}\, \| f_{\mathbf{X}}(\mathbf{x}) \|_{k/(k+2)} \tag{5}$$

where $C(k, 2)$ is the coefficient of quantization and

$$\| f_{\mathbf{X}}(\mathbf{x}) \|_{\nu} = \left(\int f_{\mathbf{X}}(\mathbf{x})^{\nu} d\mathbf{x} \right)^{1/\nu} . \tag{6}$$

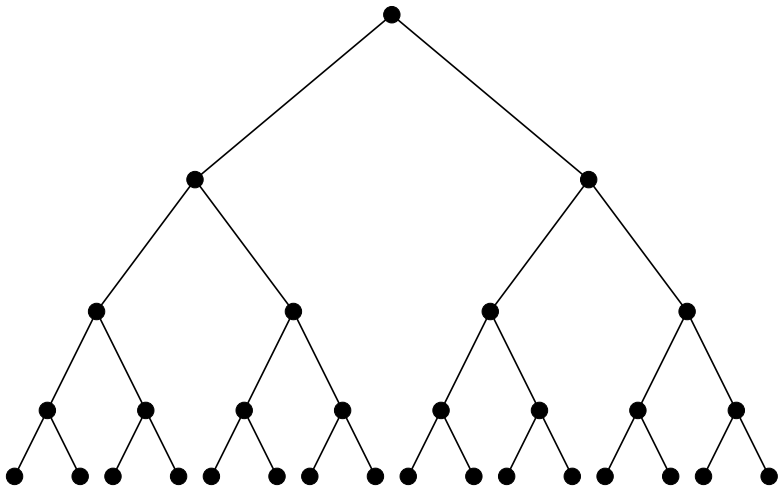

Figure 2: Example of a balanced binary tree with maximum depth 4.

This implies that increasing the codebook size N by increasing r will lower the MSE of the reproduced output. Thus, for applications requiring low average distortion, the size of the product kr can make the encoding complexity of FSVQ, which is on the order of 2^{kr}, prohibitively large.

2.1 Tree-Structured Vector Quantization

A useful approach for reducing computational complexity is to impose some form of structural constraints on the encoder. *Tree-Structured Vector Quantization* is one such technique [2]. A tree T is a finite collection of nodes with a unique root node. Each node, except the root, has a single parent node. If a node has children, it is called an *interior* node. Nodes without children are called *leaf* nodes. A node's *fanout* indicates how many children it has.

The TSVQ encoder performs its search iteratively so that a substantial number of possible codevectors are eliminated from further consideration upon each repetition. More specifically, a balanced m-ary tree reduces the set of possible codevectors by a factor of $1/m$ at each iteration. (A balanced tree is one where all leaf nodes are located at the same depth.) An example of a balanced binary tree is given in Figure 2.

A balanced binary tree with 2^{kr} leaf nodes requires only kr dot-products and threshold comparisons to encode a given input vector since

$$\|\mathbf{x}-\mathbf{y}_i\| \leq \|\mathbf{x}-\mathbf{y}_j\| \quad \Leftrightarrow \quad \mathbf{x}^t(\mathbf{y}_j-\mathbf{y}_i) \leq \frac{1}{2}(\|\mathbf{y}_j\|^2 - \|\mathbf{y}_i\|^2) \; . \tag{7}$$

Thus, for a fixed vector dimension k, TSVQ's encoding complexity is a *linear* function of the rate as opposed to exponential. The codevectors associated with TSVQ's interior nodes also give TSVQ a progressive quality. To accomplish this, the first $i < kr$ bits for each index are transmitted before sending additional bits for any block. The decoder uses each sequence of i bits to map a path through the tree to a single interior node. The codevector associated with this interior node can then be used as an approximation to the input block in an intermediate reconstruction. These two advantages come at the cost of an increase in storage and suboptimality of the encoder. The number of codevectors which must be stored for a TSVQ decoder with 2^{kr} leaf nodes is

$$\sum_{i=0}^{kr} 2^i = 2^{kr+1} - 1 \; , \tag{8}$$

which is nearly twice that of FSVQ since all interior nodes must be stored to enable progressive output. The TSVQ encoder is suboptimal because it typically does not obey the nearest neighbor condition.

To improve the performance of TSVQ, the tree is often allowed to be *unbalanced*. An unbalanced TSVQ does not have all of its leaf nodes at the same depth. Thus, the encoder is able to devote more bits to regions of the input space that contribute most to the average distortion. A popular method of designing an unbalanced TSVQ, called *greedy growing*, was described in [15] and [16]. The algorithm proceeds by repeatedly splitting the node which yields the maximum ratio of decrease in average distortion to increase in average rate. The process terminates when a constraint on maximum average rate is reached. Figure 3 shows an example of an unbalanced binary tree. In [16] it was noted that unbalanced trees usually require greatly increased storage relative to balanced trees with the same average rate. This

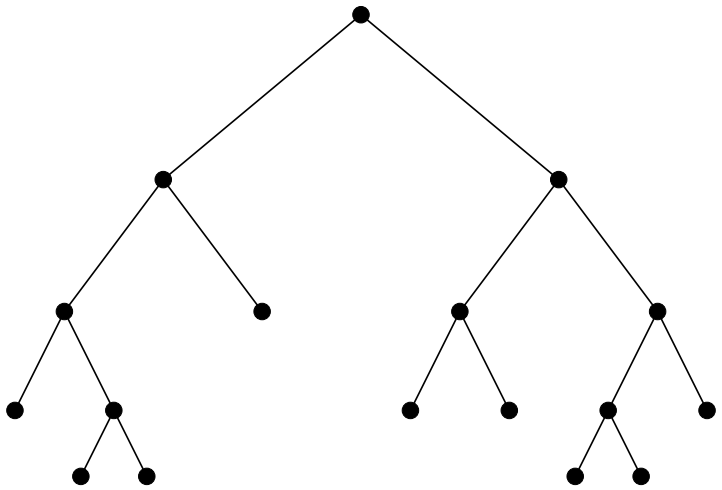

Figure 3: Example of an unbalanced binary tree.

occurs since the greedy growing technique places no constraint on the actual number of nodes within the tree; only average rate is limited. Also, the codevector indices, which represent a path map through the tree, are no longer fixed-length for unbalanced TSVQ due to the variable depths of the leaf nodes. This is another disadvantage because a single channel error can cause a loss of synchronization between the encoder and decoder. If the codevector indices of the unbalanced tree are mapped into fixed length codewords, the progressive nature of the tree is lost.

2.2 Pruned Tree-Structured Vector Quantization

If variable-length indices are allowable, an appropriate question to pose is, "What tree structure minimizes average distortion for a given average rate constraint and input source $\mathbf{X}$?" Although there is no known closed-form solution to this problem, Chou et al. addressed a related issue in [3]. In order to discuss their results, the concept of pruning, as it relates to TSVQ, must be defined. *Pruning* is a tree operation which takes a node as input and modifies the tree structure by removing all descendants of the node. In Figure 4, node t

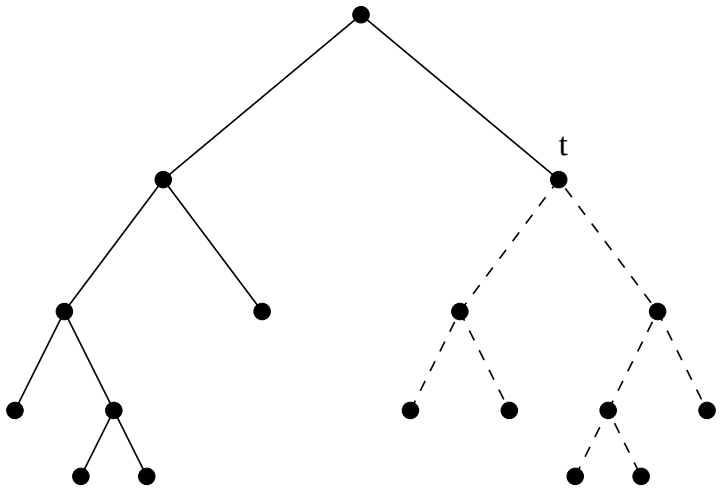

Figure 4: Node t is pruned by removing all of its descendants (dashed lines).

is pruned by removing the dashed portion of the tree, excluding t itself. If S is a pruned subtree of T, the notation $S \preccurlyeq T$ is used, signifying that S contains a subset of the nodes in T.

Pruned Tree-Structured Vector Quantization generalizes an algorithm introduced by Breiman, Friedman, Olshen, and Stone (BFOS) [17] under the context of classification and regression trees. The notation $s \in S$ denotes that s is a node in the tree S. For $S \preccurlyeq T$ and $s \in S$, let R_s represent the encoding cell associated with node s and let $\mathbf{y}_s$ denote its corresponding codevector. Next, define the following quantities: $p_{\mathbf{X}}(s) = \Pr[\mathbf{X} \in R_s]$, $l(s)$ is the number of bits needed to specify the path from the root to s, and $\bar{d}(s) = E[d(\mathbf{X}, \mathbf{y}_s) \mid \mathbf{X} \in R_s]$. Finally, let $\tilde{S}$ contain the leaf nodes from S (i.e., $\tilde{S} \subseteq S$). For a given source $\mathbf{X}$, the average rate and average distortion of S are

$$R_S = \sum_{s \in \tilde{S}} l(s)\, p_{\mathbf{X}}(s) \tag{9}$$

$$D_S = \sum_{s \in \tilde{S}} \bar{d}(s)\, p_{\mathbf{X}}(s)\ . \tag{10}$$

Each subtree of T is represented by a point (R_S, D_S) in the rate-distortion plane. Using

these points, the *operational distortion-rate function*

$$\hat{D}_T(R) = \min_{S \preccurlyeq T} \{D_S | R_S \leq R\} \tag{11}$$

then defines the optimal tradeoff between rate and distortion under the constraint that the quantizer must be a pruned subtree of T. This constraint is imposed so that the subtrees considered are still TSVQs designed for the source $\mathbf{X}$, although their maximum average rates will vary. The generalized BFOS algorithm is an efficient method for locating the extreme points of the lower convex hull of the operational distortion-rate function and their associated subtrees [3].

Tree functionals are real-valued functions on trees and their subtrees. Examples include D_S and R_S. A *monotonic nondecreasing tree functional* is a tree functional which cannot increase as branches are pruned from the tree. R_S is an example of a monotonic nondecreasing tree functional. Likewise, a *monotonic nonincreasing tree functional* is a tree functional which cannot decrease as branches are pruned. D_S is an example of a monotonic nonincreasing tree functional. A tree functional $u(S)$ is said to be *linear* if it has the property that its value is given by the sum of its values at the leaf nodes of S:

$$u(S) = \sum_{s \in \tilde{S}} u(s) \; . \tag{12}$$

In [3], it is shown that the generalized BFOS algorithm can be used to optimize the tradeoff between any two monotonic, linear tree functionals. Since both R_S and D_S are monotonic, linear tree functionals, the generalized BFOS algorithm can optimally trade average rate versus average distortion for the set $\{S : S \preccurlyeq T\}$, under a given maximum rate constraint.

Denote the pruned subtrees representing the vertices of the lower convex hull of $\hat{D}_T(R)$ as $S_1, S_2, \ldots, S_n$ when traveling right to left on the lower convex hull starting from T (as shown in Figure 5). A key result proven in [3] is that $t_{0,0} \preccurlyeq S_n \preccurlyeq \cdots \preccurlyeq S_2 \preccurlyeq S_1 \preccurlyeq T$ where $t_{0,0}$ denotes the root node of T. In words, the pruned subtrees determing the lower convex hull of $\hat{D}_T(R)$ are nested from the initial tree T back to the root node $t_{0,0}$. This permits very

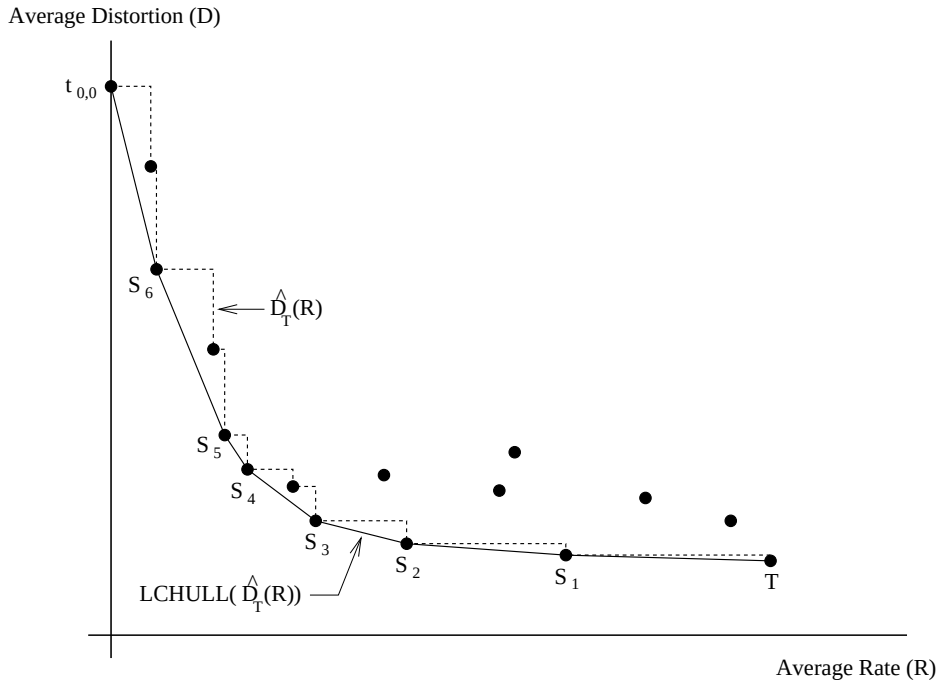

Figure 5: The lower convex hull of the operational distortion-rate function $\hat{D}_T(R)$.

efficient use of memory since only T (and the optimal pruning sequence) has to be stored in order to access all such subtrees. In addition, the generalized BFOS algorithm can locate these subtrees without an exhaustive search over $\{S : S \preccurlyeq T\}$ due to this nesting property.

In [3] it is shown that PTSVQ can outperform both TSVQ and FSVQ, in a rate-distortion sense, over a substantial range of bit rates. To accomplish this, time-sharing between adjacent subtrees on the lower convex hull is permitted. More specifically, to *time share* at rate R, PTSVQ uses S_i as the codebook for the fraction γ of the time and S_{i+1} as the codebook for $(1-\gamma)$ of the time where $0 \leq \gamma \leq 1$ and $R_{S_{i+1}} \leq R \leq R_{S_i}$. The average rate and distortion with this time-sharing technique are then $\gamma R_{S_i} + (1-\gamma)R_{S_{i+1}}$ and $\gamma D_{S_i} + (1-\gamma)D_{S_{i+1}}$ respectively. If the lower convex hull of $\hat{D}_T$ is denoted by ϕ, then $\phi(R)$ is the minimum average distortion attainable at a rate no greater than R (with time-sharing between pruned subtrees of T). In [18], it was noted that time-sharing can result in a nonstationary distribution of distortion over the quantizer output because adjacent image blocks may have significantly

different distortions when encoded using different codebooks. PTSVQ also suffers from the same disadvantages as unbalanced TSVQ, namely, the variable length of its indices and the increased storage needed for the tree.

3 THE VARIABLE FANOUT TRIMMED TSVQ ALGORITHM

Although the formulation of the generalized BFOS algorithm presented in [3] assumed a binary tree structure, it was noted that the algorithm did not, in general, require this. By applying the generalized BFOS algorithm to a broader class of tree structures, a new tree operator can be exploited in order to find an improved collection of subtrees. A *variable fanout TSVQ* is defined to be a TSVQ with the property that all nodes at the same depth have the same number of children. Between different tree depths, however, the number of children may vary, but all node fanouts are assumed to be integer powers of two. Because of these constraints, variable fanout TSVQs are special cases of nonbinary TSVQs.

Let T be a variable fanout TSVQ (e.g., see Figure 6). As with binary TSVQ, the binary sequence specifying the path from root to leaf node is an instantaneous code. Since T is balanced, this code is also fixed rate (all paths beginning at the root of T and ending at a leaf node are described by the same number of bits). The variable fanout tree T is completely specified by the number of bits used to address children of nodes at each depth, represented by the *fanout vector*, $\mathbf{b} = \langle b_0, b_1, \ldots, b_{L-1} \rangle$, where L is the maximum depth of the tree and "depth-0" refers to the root. Thus, each interior node at depth-i has 2^{b_i} children. As an example, the variable fanout TSVQ shown in Figure 6 has $L = 2$ and the fanout vector $\langle b_0 = 2, b_1 = 1 \rangle$. To determine the number of variable fanout TSVQs for a given maximum rate constraint $B = \sum_{i=0}^{L-1} b_i$ bits/vector, the number of sequences of positive integers which sum to B are counted:

$$\sum_{m=0}^{B-1} \binom{B-1}{m} = 2^{B-1}. \tag{13}$$

Thus, the number of variable fanout TSVQs increases exponentially with the maximum rate B.

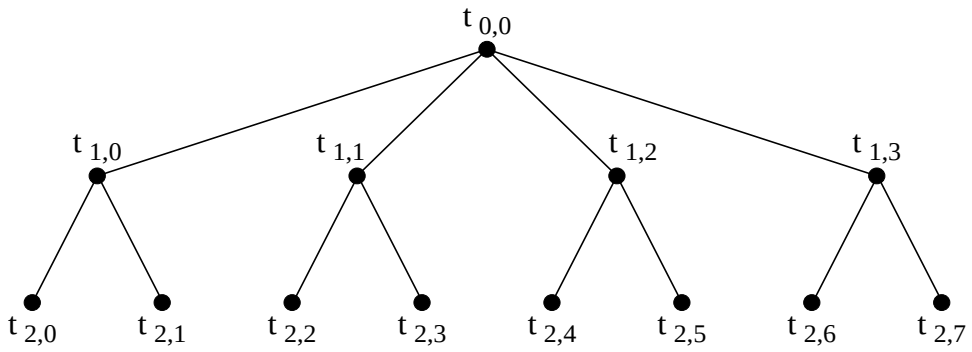

Figure 6: Example of a variable fanout TSVQ; $t_{i,j}$ denotes the j^{th} node at depth-i within the tree.

Recall that $\phi(R)$ denotes the minimum average distortion attainable at rate no greater than R by pruning a given TSVQ. Let C be a random variable representing the available transmission rate for a channel connecting a PTSVQ encoder to its decoder. Let f_C be the density function of available rates for the channel being used. One measure of performance of a source coder for this channel is the *MSE averaged over available transmission rates*, that is

$$\bar{D} = \int_0^B \phi(r)\ f_C(r)\ dr \tag{14}$$

where time-sharing between codebooks is allowed. Note that $\bar{D}$ depends both on the statistics of the source $\mathbf{X}$ and the statistics of the channel C.

3.1 Design of the Initial Variable Fanout TSVQ

The method used to produce the initial variable fanout TSVQ generalizes the design procedure for balanced binary TSVQ (i.e., "splitting"). For now, a prescribed fanout vector $\mathbf{b} = \langle b_0, b_1, \ldots, b_{L-1} \rangle$ is assumed. Let $C_N = \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{N-1}\}$ denote an N-point, k-dimensional vector codebook generated by the generalized Lloyd algorithm using a training set $\mathcal{V}$. If $t_{i,j}$ is the j^{th} node at depth-i within T, then let $\mathcal{V}^{(t_{i,j})}$ denote all those vectors in $\mathcal{V}$ which are closer to $t_{i,j}$ than any other node at depth-i. Using this notation, $C_n^{(t_{i,j})}$ denotes a size-n codebook designed for node $t_{i,j}$ using $\mathcal{V}^{(t_{i,j})}$.

The design algorithm for the initial variable fanout TSVQ follows:

STEP ZERO: Design $C_1^{(t_{0,0})}$ for the root node by finding the centroid of $\mathcal{V}$ (initialize counters $n \leftarrow 1$ and $t_{i,j} \leftarrow t_{0,0}$).

STEP ONE: Split the vectors in $C_n^{(t_{i,j})}$ and design $C_{2n}^{(t_{i,j})}$ by running the GLA with $\mathcal{V}^{(t_{i,j})}$ ($n \leftarrow 2n$).

STEP TWO: If $n < 2^{b_i}$, the current set of vectors split again (go to step one), else assign the codevectors in $C_n^{(t_{i,j})}$ to the $n = 1$ codebooks of each of $t_{i,j}$'s children.

STEP THREE: If all nodes at depth-i have been designed, go to step four, else move laterally to next node ($n \leftarrow 1$, $t_{i,j} \leftarrow t_{i,j+1}$), determine the new set $\mathcal{V}^{(t_{i,j})}$, and go to step one.

STEP FOUR: If $i+1$ is the terminal level of T, then quit, else descend one level in T ($n \leftarrow 1$ and $t_{i,j} \leftarrow t_{i+1,0}$), determine $\mathcal{V}^{(t_{i,j})}$, and go to step one.

The algorithm described above begins by creating a 2^{b_0}-point FSVQ using the GLA. It then repeatedly designs a single 2^{b_i}-point FSVQ for each of the encoding cells generated during the previous iteration ($i = 1, 2, \ldots, L-1$). The procedure for generating balanced binary TSVQs is identical except that 2-point FSVQs are generated at each iteration (i.e., the *else* directive in STEP TWO is always executed). However, as discussed in Section 2.1, greedily grown binary TSVQs do not follow this procedure.

For variable fanout trimmed TSVQ, all intermediate (or additional) codebooks $\{C_n^{(t_{i,j})} : n = 2, 4, \ldots, 2^{b_i - 1} \text{ and } t_{i,j} \notin \tilde{T}\}$ are used for the trimming of T. Note that $\{C_1^{(t_{i,j})} : t_{i,j} \in T\}$ represents the set of codevectors used by the initial tree T, and as such, these are not considered additional codebooks. It will be shown in Section 3.3 that the storage needed for the variable fanout trimmed TSVQ decoder is identical to that of a balanced binary TSVQ at the same maximum coding rate B. Thus, a decoder using either of these TSVQs will have to store the same number of vectors. For greedily grown binary TSVQ,

the resulting tree has experimentally been observed to require considerably more storage space at the decoder than variable fanout trimmed TSVQ since the number of nodes is not constrained [16].

3.2 Trimming of Variable Fanout TSVQ

The generalized BFOS algorithm can be modified to exploit the additional codebooks stored with variable fanout trimmed TSVQ, and thus potentially generate an even *lower* convex hull of the operational distortion-rate function. The tree operation of trimming provides this extension. Again let T be a variable fanout TSVQ with fanout vector $\mathbf{b}$. Let $t_{i,j}$ denote the j^{th} node at depth-i within T. $\beta(t_{i,j})$ is defined as the current fanout of node $t_{i,j}$, so that before trimming, $\beta(t_{i,j}) = 2^{b_i}$ for all interior nodes. In order to trim node $t_{i,j}$, the procedure below is followed:

Trimming a Node

STEP ONE: Prune any branches extending from $t_{i,j}$'s children.

STEP TWO: Reduce $t_{i,j}$'s children by a factor of $1/2^m$ where m is a positive integer such that $2^m \leq \beta(t_{i,j})$ (i.e., $\beta(t_{i,j}) \leftarrow \beta(t_{i,j})/2^m$).

STEP THREE: If $\beta(t_{i,j}) = 1$, prune at node $t_{i,j}$, else replace the codevectors from the remaining children with the vectors stored in $C^{(t_{i,j})}_{\beta(t_{i,j})}$.

The codebook replacement in STEP THREE guarantees that $t_{i,j}$'s new children generate the minimum MSE partition (as determined by the GLA) for its input cell. Without such replacement, the codevectors of the remaining children would no longer satisfy the centroid condition. Figure 7 shows a subtree of a variable fanout TSVQ with $\mathbf{b} = \langle 2, 2, 1 \rangle$ which could be produced by this trimming procedure. The dashed portion of the tree identifies the nodes which are removed in order to trim $t_{1,2}$ using $m = 1$. Note that during trimming,

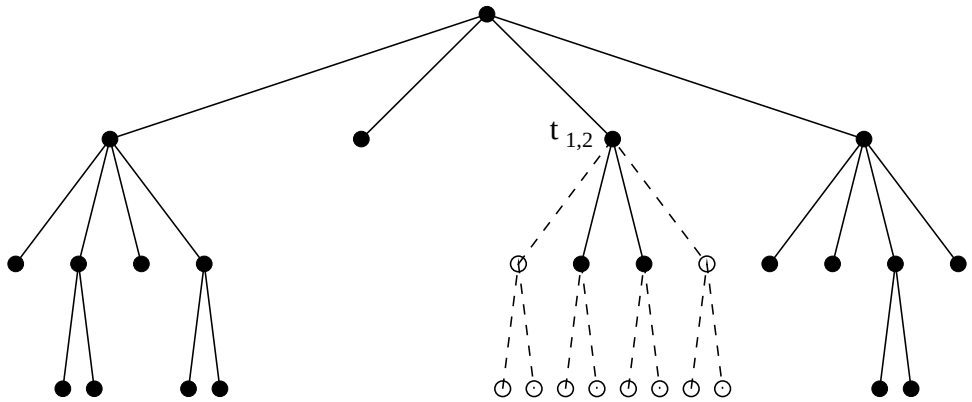

Figure 7: Example of a subtree for variable fanout trimmed TSVQ. The dashed lines indicate a possible way to trim at node $t_{1,2}$.

the remaining two children of $t_{1,2}$ will have their codevectors replaced by the 2-point FSVQ which was created for $t_{1,2}$ during the design of the initial variable fanout tree.

Let $\mathcal{P}_T$ denote the set of all subtrees which can be created by pruning some initial tree T. Likewise, let $\mathcal{T}_T$ denote the set of all subtrees which can be created by trimming T. It is important to observe that when the *if* directive in STEP THREE of the trimming procedure is executed, trimming is identical to pruning since STEP ONE and STEP TWO only affect tree nodes which are then removed in STEP THREE. Thus, trimming is a generalization of pruning. This implies $\mathcal{P}_T \subseteq \mathcal{T}_T$. In words, *any* pruned subtree of T can be generated by some trimming sequence. However, there exist, in general, trimmed subtrees of T which cannot be generated by any pruning sequence. It is the subtrees in $\mathcal{T}_T \setminus \mathcal{P}_T$ which can enable variable fanout trimmed TSVQ to outperform PTSVQ in a rate-distortion sense. For this to occur, there has to be at least one $S \in \mathcal{T}_T$ that generates a rate-distortion pair which lies below the lower convex hull of $\mathcal{P}_T$. Even if there is no such S in $\mathcal{T}_T \setminus \mathcal{P}_T$, since $\mathcal{P}_T \subseteq \mathcal{T}_T$, variable fanout trimmed TSVQ will always perform at least as well as PTSVQ.

It can be shown, in a manner similar to [3], that the trimmed subtrees lying at the vertices of the lower convex hull of $\hat{D}_T$ are nested in the sense that each can be obtained from its higher rate neighbor through a sequence of trims. Referring back to Figure 5, this implies $S_{i+1} \preccurlyeq S_i$

where $S \preccurlyeq T$ denotes that S is a trimmed subtree of T. (This is not an abuse of notation since trimming is a generalization of pruning.) The proof of this property from [3] and how it applies to variable fanout trimmed TSVQ are provided for completeness in Appendix B. A modified implementation of the generalized BFOS algorithm which locates the trimmed subtrees which determine the lower convex hull of $\hat{D}_T$ is presented in Appendix A.

Since the rate-distortion optimal subtrees are nested, they can be generated efficiently for use at both the encoder and decoder from the initial variable fanout tree structure. A method introduced in [18] for accessing nested PTSVQ subtrees is now extended so that it also applies to variable fanout trimmed TSVQ. Corresponding to each codebook $C_n^{(t_{i,j})}$ where $n = 1, 2, \ldots, 2^{b_i - 1}$ and $t_{i,j} \in T$ is a *trim number* denoted as $\alpha(t_{i,j}, n)$. $\alpha(t_{i,j}, n) > 0$ indicates that, during the generalized BFOS algorithm, node $t_{i,j}$ is trimmed, and codebook replacement is performed using $C_n^{(t_{i,j})}$ (i.e., n equals the size of the codebook replaced during the trim). Initially, $\alpha(t_{i,j}, n) = 0 \quad \forall\, t_{i,j}, n$. If trimming at node t yields the first subtree on the lower convex hull, then $\alpha(t, n) = 1$ ($n = 1$ if the trim is actually a prune). Likewise, if trimming at s generates the second subtree on the lower convex hull, then $\alpha(s, n) = 2$. This process continues as T is trimmed back to the root node. When the variable fanout trimmed TSVQ encoder generates the reproduction index for an input vector using the η^{th} trimmed subtree on the lower convex hull (where T is the 0^{th} subtree), it simply descends from the root until it encounters either a leaf node or an interior node with $\alpha(t_{i,j}, n) \leq \eta$ where $n = 1, 2, \ldots, 2^{b_i - 1}$. Note that an interior node with $\alpha(t_{i,j}, n) \leq \eta$ for some valid n would be trimmed in the process of generating the η^{th} subtree.

3.3 Storage of Variable Fanout Trimmed TSVQ

Since available storage is often a limiting factor in VQ system design, the storage required by variable fanout trimmed TSVQ is now compared to variable fanout pruned TSVQ and balanced binary PTSVQ. Let T denote a variable fanout TSVQ with

maximum depth L. Recall that if T encodes each input vector $\mathbf{X}$ with B bits, then $\sum_{i=0}^{L-1} b_i = B$. Let $\mathcal{C}$ denote the set of all additional codebooks associated with T, namely $\mathcal{C} = \{C_n^{(t_{i,j})} : n = 2, 4, \ldots, 2^{b_i - 1} \text{ and } t_{i,j} \notin \tilde{T}\}$, and let T_{bin} be a depth-B balanced binary tree. In order for a variable fanout tree-structured decoder to be progressive, it must store a codevector for each interior node. If the encoder transmits the first n bits for each input block, the decoder can use the interior codevectors at depth-m where $\sum_{i=0}^{m-1} b_i \leq n$ to generate an approximation to the final image. These interior vectors are specified using the first $\sum_{i=0}^{m-1} b_i$ bits of the received index for each block. Thus, the variable fanout trimmed TSVQ decoder must store one k-dimensional codevector for each node in T. The decoder must also store all codevectors contained within $\mathcal{C}$ to permit all trimmed subtrees to be accessed. The decoder storage for T is

$$
\begin{aligned}
& |T| + |\mathcal{C}| && (15)\\
=& |T| + \sum_{i=0}^{L-1} \left((2 + 4 + \cdots + 2^{b_i - 1}) \prod_{j=-1}^{i-1} 2^{b_j} \right) && (16)\\
=& |T| + \sum_{i=0}^{L-1} \left((2^{b_i} - 2) \prod_{j=-1}^{i-1} 2^{b_j} \right) && (17)\\
=& |T| + |T| - 1 - 2 \cdot \sum_{i=0}^{L-1} \prod_{j=-1}^{i-1} 2^{b_j} && (18)\\
=& |T| + |T| - 1 - 2 \cdot \left(|T| - \prod_{j=0}^{L-1} 2^{b_j} \right) && (19)\\
=& 2 \cdot (2^{b_0 + b_1 + \cdots + b_{L-1}}) - 1 && (20)\\
=& 2 \cdot 2^B - 1 && (21)\\
=& 1 + 2 + 4 + 8 + \cdots + 2^B && (22)\\
=& |T_{bin}| && (23)
\end{aligned}
$$

In the equations above, $b_{-1} = 0$. Also, (19) follows by noting that the last term in (18) is twice the number of interior nodes in T. For a maximum encoding rate B, a progressive

decoder using either variable fanout trimmed TSVQ or balanced binary PTSVQ has to store the same number of vectors. Variable fanout *pruned* TSVQ, however, requires less storage at the decoder since it does not use additional codebooks.

A storage comparison at the encoder is more complex because any tree node with a fanout of two only needs to store a single vector equal to the difference between the codevectors of its children. A dot-product between an input $\mathbf{X}$ and this difference vector, followed by a threshold comparison, is enough to determine which child is closer to $\mathbf{X}$ (see Eq. (7)). This technique reduces the balanced binary TSVQ encoder storage by a factor of two. Variable fanout TSVQ cannot take advantage of this shortcut for any nodes with fanouts greater than 2 (i.e., $b_i > 1$) because the number of vector differences that would need to be stored equals $\binom{2^{b_i}}{2}$ for any node at depth-i. Clearly, it is more memory efficient to simply store the 2^{b_i} codevectors whenever $b_i > 1$. The encoder storage for T is

$$\sum_{i=0}^{L-1} \left((1 + 4 + 8 + 16 + \cdots + 2^{b_i}) \prod_{j=-1}^{i-1} 2^{b_j} \right) \tag{24}$$

$$= \sum_{i=0}^{L-1} \left((2^{b_i+1} - 3) \prod_{j=-1}^{i-1} 2^{b_j} \right) \tag{25}$$

$$= 2 \cdot (|T| - 1) - 3 \cdot (|T| - 2^B) \tag{26}$$

$$= |T_{bin}| - |T| + 2^B - 1 \tag{27}$$

$$= |\mathcal{C}| + 2^B - 1 \tag{28}$$

$$= |\mathcal{C}| + \frac{1}{2}(|T_{bin}| - 1) \tag{29}$$

Thus, variable fanout trimmed TSVQ requires that $|\mathcal{C}|$ more codevectors be stored at the encoder than does balanced binary PTSVQ. Variable fanout *pruned* TSVQ also requires less encoder storage because it does not use additional codebooks.

3.4 Choosing the Fanout Vector to Minimize $\bar{D}$

So far, the fanout vector $\mathbf{b}$ has been assumed to be fixed. However, in Section 2.2 it was shown that for a given maximum rate constraint B, there exist 2^{B-1} possible fanout vectors. This introduces an extra degree of freedom which is not present in PTSVQ systems. Since each fanout vector $\mathbf{b}$ yields a different initial tree structure, there is generally a unique lower convex hull associated with each $\mathbf{b}$. If the allowable transmission rates for a given channel are known, the fanout vector yielding the trimmed subtrees with the lowest MSEs at or near these rates should be chosen. Likewise, if a probability distribution for available transmission rates, f_C, is either known or can be estimated for the channel, the fanout vector which minimizes $\bar{D}$ (see Eq. (14)) should be chosen.

Since the variable fanout trimming process generates rate-distortion pairs which determine the lower convex hull (see Appendix A), a numeric integration subroutine can be embedded within the algorithm in order to approximate $\bar{D}$ for a given rate density f_C. Unfortunately, as the maximum rate B grows, it may become computationally infeasible to design all 2^{B-1} variable fanout TSVQs in order to minimize $\bar{D}$. The following heuristic approach will be shown experimentally to be satisfactory in Section 4.2.

Given a transmission rate density f_C with peaks at $r_1, r_2, \ldots, r_n$, the initial tree structures which minimize $\bar{D}$ will be those which satisy

$$\left(r_j - \frac{1}{2}\right) \leq \sum_{i=0}^{m_j} b_i \leq \left(r_j + \frac{1}{2}\right) \tag{30}$$

for each $j = 1, 2, \ldots, n$ where $m_j \in \{0, 1, 2, \ldots, L-1\}$. For example, if f_C is bimodal with peaks at 5 bits/vector and 8 bits/vector, fanout vectors such as $\langle b_0 = 5, b_1 = 3, \ldots \rangle$ or $\langle b_0 = 1, b_1 = 4, b_2 = 3, \ldots \rangle$ would likely perform well. Thus, if computational resources are limited, these tree structures alone can be generated and tested.

4 EXPERIMENTAL RESULTS

In this chapter, two ways of experimentally evaluating the performance of variable fanout trimmed TSVQ are considered. First, the lower convex hull of the rate-distortion pairs is generated by each of the various algorithms operating under the same maximum rate constraint, B. Since the convex hull is determined as part of the codebook design procedure, this analysis evaluates how the various algorithms perform only on training data. Second, $\bar{D}$, the MSE averaged over available rates, is evaluated for the various algorithms and choices of $\mathbf{b}$. To accomplish this, several test images are encoded using the lower convex hull subtrees generated by each algorithm in order to find their experimental average rates and MSEs. In addition to providing performance results using test data, this also demonstrates how variable fanout trimmed TSVQ can be optimized for the channel.

4.1 Lower Convex Hull Results

First, a comparison is made between the locations within the rate-distortion plane of the subtrees generated by variable fanout trimmed TSVQ and variable fanout pruned TSVQ. This is accomplished by running both algorithms on the same initial variable fanout tree and then plotting the resulting lower convex hulls. The variable fanout tree structure was designed for a maximum rate of $B = 6$ bits/vector using the fanout vector $\langle 2, 2, 2 \rangle$. The initial tree was trained using two 512×512 images ("Man" from the USC database and "Goldhill" from the RPI database) and vector dimension $k = 4$. Figure 8 shows the lower convex hulls resulting from optimal trimming and optimal pruning. As expected, some of the additional rate-distortion pairs generated by trimming fall below the lower convex hull created by pruning alone. Most noticeably, at the average rate $R = 1$ bit/vector, the trimmed hull dips lower than the pruned one. Since pruning does not incorporate additional codebooks, it cannot generate any subtrees between 2 bits/vector and 0 bits/vector.

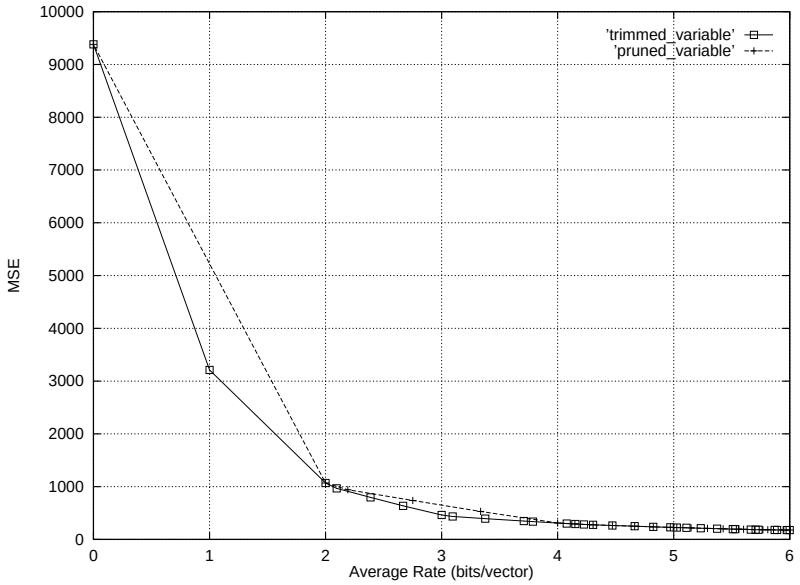

Figure 8: Trimmed and pruned-only lower convex hulls using initial tree with fanout vector $\langle 2, 2, 2 \rangle$. Squares are trimmed subtrees; crosses are pruned subtrees.

In Figure 9, the 4–6 bits/vector range from Figure 8 is expanded. It is interesting to note that the six subtrees located from $R = 4.30$ bits/vector to $R = 5.02$ bits/vector are identical. This implies that trimming either did not generate additional subtrees in this region, or the additional subtrees generated were all located above the pruned lower convex hull.

Since variable fanout trimmed TSVQ is not guaranteed to outperform variable fanout pruned TSVQ, it is interesting to observe empirically how often trimming generates a rate-distortion pair below the pruned convex hull. Using the same two training images and vector dimension, all possible variable fanout trees for $B = 10$ bits/vector were designed. Recall from Section 2.2 that there are $2^{B-1} = 512$ fanout vectors possible for this maximum rate. However, only 444 initial variable fanout trees were generated. The other 68 combinations had some node $t_{i,j}$ which could not be split during the design process because $|\mathcal{V}^{(t_{i,j})}| < 2^{b_i}$. After using the generalized BFOS algorithm to optimally trim each variable fanout tree, 443 of them were found to have used codebook replacement while trimming. The only variable

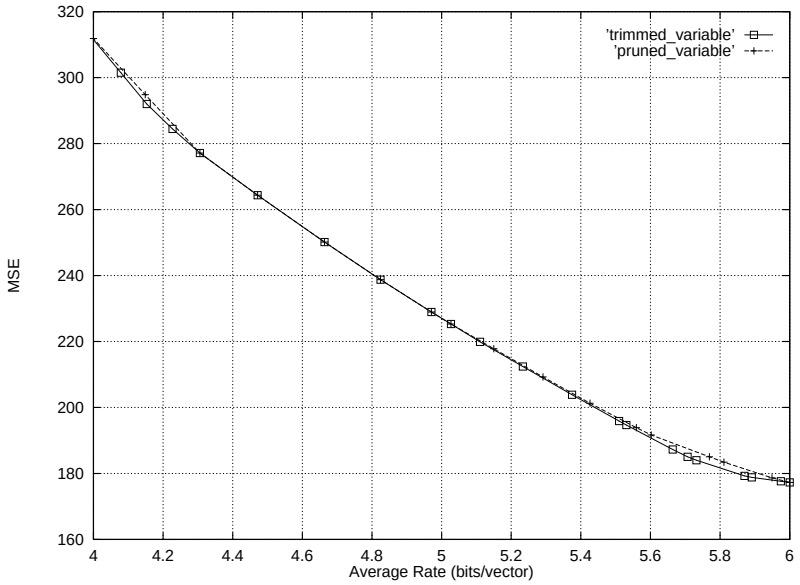

Figure 9: Enlargement of range 4–6 bits/vector from Figure 8.

fanout tree that used pruning alone (no codebook replacement) was the one representing balanced binary TSVQ (i.e., fanout vector $\langle 1,1,1,1,1,1,1,1,1,1 \rangle$).

Both balanced and unbalanced binary PTSVQ are now incorporated into the figures. Binary PTSVQ is included because it represents a popular implementation of PTSVQ. The lower convex hulls of rate-distortion pairs for all four algorithms are shown in Figure 10 for the same training images, dimension, and maximum rate of 10 bits/vector. However, the fanout vector used for the figure is $\langle 1,1,1,3,2,2 \rangle$. Figure 10 displays the restricted range of 6–8 bits/vector so that the various subtrees can be distinguished. Again, variable fanout trimmed TSVQ generates subtrees which lie below the hull of variable fanout pruned TSVQ. Also, the nonbinary fanouts allow variable fanout trimmed TSVQ to outperform balanced binary PTSVQ by 0.4–0.8 dB over the displayed range. Finally, unbalanced binary PTSVQ does not approach variable fanout trimmed TSVQ's rate-distortion performance until $R \approx 8$ bits/vector, and this only occurs because the unbalanced subtrees become substantially larger than the variable fanout subtrees at higher rates. For example, the unbalanced

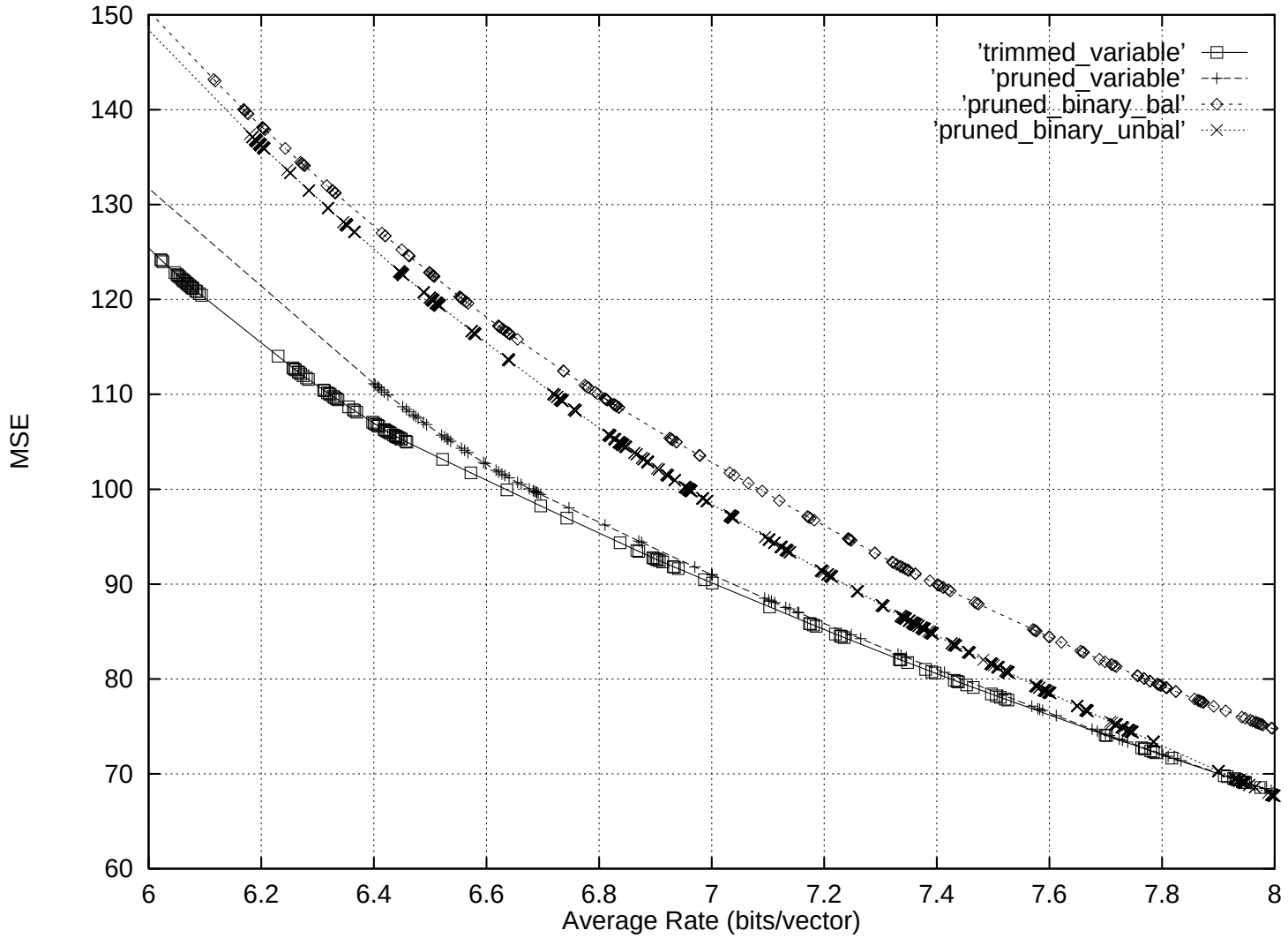

Figure 10. Lower convex hulls of all four algorithms. Squares and solid line are variable fanout trimmed TSVQ; crosses and dashed line are variable fanout pruned TSVQ; diamonds and dotted line are balanced binary PTSVQ; x's and dashed line are unbalanced binary PTSVQ.

binary PTSVQ subtrees with $R \approx 8$ bits/vector contain approximately 7500 nodes while the variable fanout subtrees contain only 835 nodes. Thus, even though the rate-distortion performance of unbalanced binary PTSVQ might eventually exceed that of variable fanout trimmed TSVQ as R grows, trimming would still have the advantage of reduced storage.

4.2 Minimizing $\bar{D}$ for a Known f_C

Now, several examples are considered in order to demonstrate that the heuristic method introduced in Section 3.4 for choosing $\mathbf{b}$ given f_C yields tree structures which minimize $\bar{D}$. To accomplish this, the 10 minimum-$\bar{D}$ fanout vectors for various channel rate distributions are listed. The training data, vector dimension, and maximum rate all remain equal to their values from the end of Section 4.1. Let $f_{C,0}$ be uniform on $R \in [0, 10]$. Let $f_{C,1}$ and $f_{C,2}$ both be Gaussian with $\sigma^2 = 0.01$, $\mu_1 = 5.5$ bits/vector, and $\mu_2 = 8.0$ bits/vector. Let $f_{C,3} = \frac{1}{2}(f_{C,1} + f_{C,2})$, a bimodal distribution. $\bar{D}$ was evaluated for all 444 variable fanout trees which could be generated (see Section 4.1). The results are presented in Table 1.

For $f_{C,0}$, the uniform distribution, most of the minimum-$\bar{D}$ fanout vectors begin with 2 or 3 unitary allocations. This assures that there are trimmed subtrees at low bit rates. This is important because the MSE is greatest for subtrees with small R. For $f_{C,1}$, the Gaussian distribution with $\mu_1 = 5.5$ bits/vector, the minimum-$\bar{D}$ fanout vectors all have $\mathbf{b} = \langle 1, 4, \ldots \rangle$. It is somewhat surprising that trees with $b_0 = 5$ are not ranked highest for $f_{C,1}$, since their trimmed subtrees are approximately size-32 FSVQs at $R \approx 5$ bits/vector. However, such trees only have trimmed subtrees at integer rates below $R = 5$ bits/vector (i.e., $R = 4, 3, 2, 1, 0$). Thus, any variable fanout TSVQs with subtrees satisfying $4 < R < 5$ will tend to have lower values of $\bar{D}$ for $f_{C,1}$ since these subtrees typically lie below the line segment connecting a 16-point FSVQ to a 32-point FSVQ within the rate-distortion plane. Furthermore, for $f_{C,1}$, the fanout vectors whose $\bar{D}$ values ranked 18–32 smallest (not included in Table 1) all have $\mathbf{b} = \langle 5, \ldots \rangle$ while those from 33–44 all have $\mathbf{b} = \langle 2, 3, \ldots \rangle$. Thus, a tree's

Table 1: Fanout vectors which minimize $\bar{D}$ on the training images "Man" and "Goldhill" for various available rate distributions ($k = 4$ and $B = 10$ bits per vector).

	Fanout Vector $\mathbf{b}$			
Rank	$f_{C,0}$	$f_{C,1}$	$f_{C,2}$	$f_{C,3}$
1	$\langle 1,1,1,3,4\rangle$	$\langle 1,4,2,1,2\rangle$	$\langle 7,3\rangle$	$\langle 1,4,3,2\rangle$
2	$\langle 1,1,4,4\rangle$	$\langle 1,4,2,3\rangle$	$\langle 1,6,3\rangle$	$\langle 1,4,2,3\rangle$
3	$\langle 1,1,1,3,2,2\rangle$	$\langle 1,4,2,2,1\rangle$	$\langle 7,1,2\rangle$	$\langle 1,4,2,1,2\rangle$
4	$\langle 1,4,2,3\rangle$	$\langle 1,4,2,1,1,1\rangle$	$\langle 1,6,1,2\rangle$	$\langle 1,4,3,1,1\rangle$
5	$\langle 1,1,1,3,1,3\rangle$	$\langle 1,4,3,2\rangle$	$\langle 7,2,1\rangle$	$\langle 1,4,2,2,1\rangle$
6	$\langle 1,1,4,1,3\rangle$	$\langle 1,4,3,1,1\rangle$	$\langle 1,6,2,1\rangle$	$\langle 1,4,2,1,1,1\rangle$
7	$\langle 1,1,1,3,3,1\rangle$	$\langle 1,4,1,2,2\rangle$	$\langle 1,1,5,3\rangle$	$\langle 5,3,2\rangle$
8	$\langle 1,1,3,2,3\rangle$	$\langle 1,4,1,1,3\rangle$	$\langle 1,2,4,3\rangle$	$\langle 5,2,3\rangle$
9	$\langle 1,4,2,1,2\rangle$	$\langle 1,4,1,2,1,1\rangle$	$\langle 7,1,1,1\rangle$	$\langle 1,4,5\rangle$
10	$\langle 1,1,4,2,2\rangle$	$\langle 1,4,1,3,1\rangle$	$\langle 1,5,4\rangle$	$\langle 5,2,1,2\rangle$

performance is closely related to whether nodes are located at the channel's peak rates. The results in Table 1 for $f_{C,2}$ can be explained in a manner similar to $f_{C,1}$ (note that none of the top 10 fanout vectors have $b_0 = 8$ for $f_{C,2}$). Finally, $f_{C,3}$ illustrates the importance of performing well at those rates which contribute most to $\bar{D}$ since many of the best tree structures for $f_{C,3}$ are also optimal for $f_{C,1}$. If $f_{C,1}$ and $f_{C,2}$ had been weighted differently in creating $f_{C,3}$, this would not necessarily be the case.

In order to demonstrate that this minimum-$\bar{D}$ ranking for various $\mathbf{b}$ is empirically correct, allocations $\langle 1,4,3,1,1\rangle$ and $\langle 1,2,4,3\rangle$ were tested on the channel with density $f_{C,1}$ using three 512×512 images ("Plane," "Woman," and "Peppers" from the USC database). In Figure 11, the product of $f_{C,1}$ and the MSE is displayed, where the MSE is averaged over the three test images for each subtree. The integral of this curve yields $\bar{D}$, the MSE averaged over available rates. This curve is important to view because it identifies those rates which contribute most to $\bar{D}$. The points plotted are not subtrees; instead they represent the

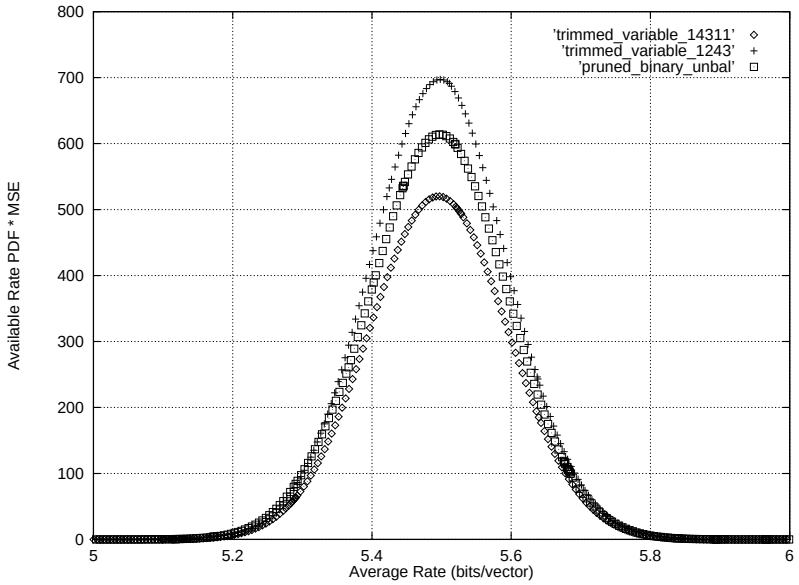

Figure 11: The product of available rate distribution and MSE for test data. The channel density is $\mathcal{N}(5.5, 0.01)$.

rates used to approximate $\bar{D}$. The curve for unbalanced binary PTSVQ is also included as a benchmark. As predicted in Table 1, fanout vector $\mathbf{b} = \langle 1, 4, 3, 1, 1 \rangle$ provides superior performance to $\langle 1, 2, 4, 3 \rangle$ (a fanout vector optimized for $f_{C,2}$).

Using the same test images, the performance gain using variable fanout trimmed TSVQ on two channels with rate distributions $f_{C,1}$ and $f_{C,2}$ was numerically evaluated. Three fanout vectors were chosen: one which is optimal for $f_{C,0}$, one which is optimal for $f_{C,1}$, and one which is otimal for $f_{C,2}$ (see Table 1). The peak signal-to-noise ratio ($\text{PSNR} = 10 \log_{10}(255^2/\bar{D})$) achieved on the test data is listed in Table 2 for each of these, normalized to the PSNR attainable using unbalanced binary PTSVQ. In addition, pruned versions of these same variable fanout tree structures are included along with balanced binary PTSVQ. The approximate subtrees sizes are included for all entries to permit a fair comparison. Note that all four algorithms use time-sharing at channel rates where they have no pruned or trimmed subtrees.

Table 2: PSNR gain of variable fanout TSVQ (both pruned and trimmed) compared to unbalanced binary PTSVQ on test images for two different channel densities.

		$f_{C,1}$		$f_{C,2}$	
Algorithm	Bits/Level	Gain (dB)	Nodes	Gain (dB)	Nodes
unbalanced binary PTSVQ	NA	0.000	15,105	0.000	16,260
balanced binary PTSVQ	1111111111	-0.022	1700	-0.499	1895
variable fanout trimmed TSVQ	14311	0.707	1540	-0.271	1665
variable fanout PTSVQ	14311	0.682	1585	-0.483	1670
variable fanout trimmed TSVQ	1243	-0.491	940	0.241	1075
variable fanout PTSVQ	1243	-1.652	1000	0.266	1120
variable fanout trimmed TSVQ	11134	0.055	900	-0.053	1010
variable fanout PTSVQ	11134	-0.763	815	-0.094	1025

Referring to Table 2, trimming offers a PSNR gain of up to 1.16 dB over variable fanout PTSVQ for density $f_{C,1}$. The pattern in the experimental data demonstrates that trimming can achieve a substantial gain over pruned TSVQ for a given channel rate distribution. Performance gains are found at rates where trimming creates additional subtrees which lie below the lower convex hull found by the ordinary BFOS algorithm. This occurs more at lower rates because PTSVQ generates fewer subtrees in this region for variable fanout trees. For $f_{C,2}$, this phenomenon is best seen for $\mathbf{b} = \langle 1, 4, 3, 1, 1 \rangle$.

When compared to binary tree structures, the best variable fanout trimmed TSVQ in each case offers a PSNR gain of approximately 0.7 dB over balanced binary PTSVQ and 0.24–0.71 dB over unbalanced binary PTSVQ. The reason the performance gap narrows between variable fanout trimmed TSVQ and unbalanced binary PTSVQ as the rate increases is because the difference in subtree sizes becomes significant. For example, the pruned subtrees of the unbalanced binary PTSVQ with average rate near 8.0 bits/vector contain approximately 16,260 nodes. This is in comparison to both variable fanout trimmed TSVQ and balanced binary PTSVQ whose *initial* trees ($R = 10$ bits/vector) require the storage

Table 3: PSNR gain of variable fanout trimmed TSVQ compared to unbalanced binary PTSVQ on individual test images for two different channel densities.

	$f_{C,1}$		$f_{C,2}$	
Test Image	Bits/Level	Gain (dB)	Bits/Level	Gain (dB)
"Plane"	111322	1.433	1243	-0.652
"Woman"	1144	0.810	1243	0.917
"Peppers"	1423	2.364	1612	0.612

of only 2047 codevectors at the decoder. Therefore variable fanout trimmed TSVQ offers a performance gain over unbalanced binary PTSVQ while actually reducing the storage. This memory reduction is significant since it implies that multiple variable fanout tree structures can be stored at the decoder with less memory than a single greedily grown binary PTSVQ. Thus, if the channel rate distribution has a bimodal distribution such as $f_{C,3}$, variable fanout trees with $\mathbf{b} = \langle 1, 4, 3, 1, 1 \rangle$ and $\langle 1, 2, 4, 3 \rangle$ can both be stored. Since the initial unbalanced binary PTSVQ contained 17,177 nodes, storing both variable fanout trees would provide superior performance at only 1/4 the memory usage.

The experimental results presented so far are averaged over three images. However, it is interesting to quantify how well variable fanout trimmed TSVQ can perform on a single image. In Table 3, the maximum PSNRs achieved for each test image using the fanout vectors from the first three columns and the top eight rows of Table 1 are listed. All PSNRs in Table 3 are relative to the PSNR achieved by unbalanced binary PTSVQ on the corresponding test image. In many cases, variable fanout trimmed TSVQ can outperform unbalanced binary PTSVQ by a substantially larger margin than found in Table 2. For example, coding "Peppers" with $\mathbf{b} = \langle 1, 4, 2, 3 \rangle$ provides a PSNR gain of over 2 dB. However, "Plane" transmitted over the channel with density $f_{C,2}$ is an exception, a result which may be due to unbalanced binary PTSVQ's substantially larger codebook at higher rates.

5 CONCLUSION AND FUTURE RESEARCH

In this thesis, I have introduced a new variable-rate encoding technique called variable fanout trimmed TSVQ which experimentally outperforms PTSVQ. This improvement is made possible through the generalized pruning technique of trimming which creates new subtrees for a given initial tree. Often, these new rate-distortion pairs lie below the lower convex hull generated by pruning alone. I have also shown that variable fanout trimmed TSVQ's memory use at both the encoder and decoder is on the order of other PTSVQ techniques. In the case of greedily grown binary PTSVQ, variable fanout trimmed TSVQ generally requires considerably less storage. Thus, variable fanout trimmed TSVQ provides the benefits of progressivity and low encoding complexity while providing a lower MSE than PTSVQ. Additionally, the fanout vector for the initial tree can be optimized with respect to an available rate density of a channel, providing a further advantage over PTSVQ.

Since the design of the initial tree is separate from the generation of trimmed subtrees, a different design algorithm could be used prior to trimming. Thus, the initial nonbinary tree could be created using the greedy growing technique described in Section 2.1. Since a variable fanout tree is not likely to result, the subsequent trimming algorithm would have to be altered. The experimental performance of greedily grown trimmed TSVQ could then be compared to variable fanout trimmed TSVQ and PTSVQ. However, it should be noted that growing the initial nonbinary tree in a greedy fashion prevents the channel optimization possible with variable fanout trimmed TSVQ.

APPENDIX A PSEUDOCODE FOR OPTIMAL TRIMMING

In this appendix, the algorithm used to generate the rate-distortion optimal sequence of trimmed subtrees is presented. It is very similar to the generalized BFOS algorithm introduced in [3] except for the incorporation of additional codebooks. Let $\mathbf{u} = (u_1, u_2)$ represent the pair of average rate and MSE, respectively, for the tree currently under consideration. Associated with each interior node at depth-i is a data structure containing several arrays of length b_i and various pointers. Each array element corresponds to a codebook stored at the node (i.e., $C_n^{(t)}$ for $n = 1, 2, 4, \ldots, 2^{b_i - 1}$). Similarly to [3], the following quantities for interior node t are defined:

$\triangle u_1(t, n) =$ change in average rate of tree by trimming at node t and performing codebook substitution if $n \neq 1$

$\triangle u_2(t, n) =$ change in MSE of tree by trimming at node t and performing codebook substitution if $n \neq 1$

$\lambda(t, n) =$ negative slope of line segment within the rate-distortion plane resulting from trimming node t

$\lambda_{\min}(t) =$ minimum value of λ achievable by trimming at either node t or any of its descendants (not an array)

$\alpha(t, n) =$ trim number described in Section 3.2 .

Additionally, each interior node contains pointers to its parent node and each of its children. In the following pseudocode, *parent(t)* dereferences the pointer to t's parent node. Leaf nodes have these same data fields except each array has only one entry since leaf nodes have no additional codebooks. Two additional parameters are needed for the initialization of the trimming algorithm. Given input random vector $\mathbf{X}$, let $u_1(t, n) = p_{\mathbf{X}}(t) \cdot (l(t) + \log_2(n))$, and let $u_2(t, n) = p_{\mathbf{X}}(t) \cdot \bar{d}_n$ where $\bar{d}_n$ represents the average distortion of those input vectors

mapped into node t and encoded using $C_n^{(t)}$. The trimming algorithm begins with the following initialization.

For each leaf node t,

$$\begin{aligned}
\triangle u_1(t,1) &\leftarrow 0 \\
\triangle u_2(t,1) &\leftarrow 0 \\
\lambda(t,1) &\leftarrow \infty \\
\lambda_{\min}(t) &\leftarrow \infty \\
\alpha(t,1) &\leftarrow 0 \ .
\end{aligned}$$

For each interior node t at depth-i and $n = 1, 2, 4, \ldots, 2^{b_i - 1}$,

$$\begin{aligned}
\Delta \mathbf{u}(t,n) &\leftarrow -\mathbf{u}(t,n) + \sum_{s:\text{parent}(s)=t} \left(\mathbf{u}(s,1) + \Delta \mathbf{u}(s,1) \right) \\
\lambda(t,n) &\leftarrow -\triangle u_2(t,n) / \triangle u_1(t,n) \\
\lambda_{\min}(t) &\leftarrow \min \left\{ \lambda(t,n),\ \lambda_{\min}(s) : n = 1, 2, 4, \ldots, 2^{b_i - 1} \text{ and parent}(s) = t \right\} \\
\alpha(t,n) &\leftarrow 0 \ .
\end{aligned}$$

With this initialization, the minimum value of λ over all interior nodes will be stored in the root's data structure, $\lambda_{\min}(t_{0,0})$. As in [3], this algorithm begins with the full tree and trims the node with the minimum value of λ. Then, all data structures affected by the trim are updated, and the process is repeated until only the root remains. The trimming algorithm follows.

$t \leftarrow t_{0,0}$
$\mathbf{u} \leftarrow \mathbf{u}(t,1) + \Delta \mathbf{u}(t,1)$
$\eta \leftarrow 0$
print $\eta,\ \lambda_{\min}(t_{0,0}),\ \mathbf{u}$
while ($\lambda_{\min}(t_{0,0}) < \infty$) do

$\eta \leftarrow \eta + 1$

$n \leftarrow 1$

while ($\lambda(t,n) \neq \lambda_{\min}(t_{0,0})$) do

if ($\lambda_{\min}(t_{0,0}) \in \{\lambda_{\min}(s) : \text{parent}(s) = t\}$)

$t \leftarrow s$ where $\lambda_{\min}(s) = \lambda_{\min}(t_{0,0})$

else

$n \leftarrow m$ where $\lambda(t,m) = \lambda_{\min}(t_{0,0})$

end

$\Delta \leftarrow \Delta\mathbf{u}(t,n)$

$\alpha(t,n) \leftarrow \eta$

if $(n = 1)$

$\lambda_{\min}(t) \leftarrow \infty$

else

$\lambda(t,n) \leftarrow \infty$

$m \leftarrow 1$

while $(m < n)$ do

$\Delta\mathbf{u}(t,m) \leftarrow \Delta\mathbf{u}(t,m) - \Delta$

$\lambda(t,m) \leftarrow -\triangle u_2(t,m)/\triangle u_1(t,m)$

$m \leftarrow 2 \cdot m$

end

$\lambda_{\min}(t) \leftarrow \min\left\{\lambda(t,m) : m = 1, 2, 4, \ldots, \frac{n}{2}\right\}$

while $(t \neq t_{0,0})$ do

$t \leftarrow \text{parent}(t)$

$n \leftarrow 1$

while ($n < \beta(t)$) do

$\Delta\mathbf{u}(t,n) \leftarrow \Delta\mathbf{u}(t,n) - \Delta$

$\lambda(t,n) \leftarrow -\triangle u_2(t,n)/\triangle u_1(t,n)$

$n \leftarrow 2 \cdot n$

end

$\lambda_{\min}(t) \leftarrow \min\left\{ \lambda(t,n),\ \lambda_{\min}(s) : n = 1, 2, 4, \ldots, \frac{\beta(t)}{2} \text{ and } \text{parent}(s) = t \right\}$

end

$\mathbf{u} \leftarrow \mathbf{u} - \Delta$

print η, $\lambda_{\min}(t_{0,0})$, $\mathbf{u}$

end.

APPENDIX B PROOF OF NESTING PROPERTY

For completeness, the proof from [3] is presented to show that the trimmed subtrees lying on the lower convex hull of T have tree structures which are nested from the initial balanced tree to the single root node. All unions and intersections are taken with respect to the tree structures themselves, not their associated codevectors. It is assumed that codebook replacement, as detailed in Section 3.2, is performed after each union and intersection in order to guarantee that the resulting subtree yields the minimum average distortion for its structure. In the subsequent lemma and corollary, if tree S can be attained from T through a sequence of trimming operations, the notation $S \preccurlyeq T$ is used.

Let $\mathbf{u}(S) = (u_1(S), u_2(S))$ where $u_1(S)$ represents the average rate of S and $u_2(S)$ represents the average distortion (MSE), both with respect to a given source $\mathbf{X}$. Let $\beta(t)$ denote the fanout at node t.

Lemma 1 *Let S, $S' \preccurlyeq T$ be trimmed subtrees of T, and let F be a face of the lower convex hull of $\{\mathbf{u}(\hat{S}) : \hat{S} \preccurlyeq T\}$. Define $R = S \cap S'$ and $R' = S \cup S'$. If $\mathbf{u}(S)$, $\mathbf{u}(S') \in F$, then $\mathbf{u}(R)$, $\mathbf{u}(R') \in F$.*

Proof The intersection and union imply the following about the structure of R and R':

$$\beta(r_{i,j}) = min\{\beta(s_{i,j}),\ \beta(s'_{i,j})\} \tag{31}$$

$$\beta(r'_{i,j}) = max\{\beta(s_{i,j}),\ \beta(s'_{i,j})\} \tag{32}$$

where if either s or s' are nonexistent, β returns zero. These relations imply that $R \preccurlyeq S, S'$ and $S, S' \preccurlyeq R'$. Thus, there exists a sequence of trimming operations that convert S' into R. These operations remove those branches of S' that are not found in S. Likewise, R' can be converted into S through a sequence of trimming operations. These operations remove those branches from S' that are not contained in S. In both cases, the set of branches being removed is identical. Since the subtrees S, S' represent partitions of the input space $\mathcal{R}^k$,

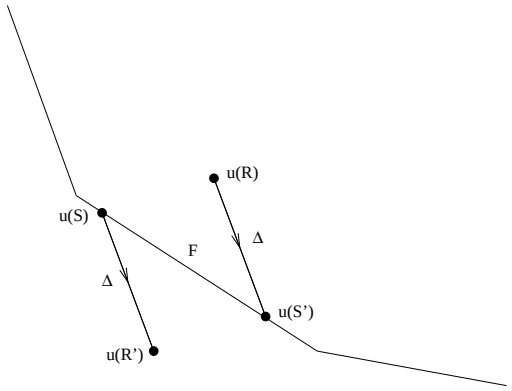

Figure 12: Interpretation of (33) and (34) on the lower convex hull of the operational distortion-rate function for T.

these adjustments will affect the average rate and MSE in the same way. This result can be expressed as

$$\mathbf{u}(R) + \Delta = \mathbf{u}(S') \tag{33}$$

$$\mathbf{u}(S) + \Delta = \mathbf{u}(R') \tag{34}$$

where $\triangle_1 \geq 0$ since average rate cannot decrease as branches are added and $\triangle_2 \leq 0$ since MSE cannot increase with a finer partition. Figure 12 along with (33) and (34) imply that both R and R' must lie on F since if either did not, there would be a rate-distortion pair outside the convex hull. □

Corollary 1 *Let F be a face of the convex hull $\{\mathbf{u}(\hat{S}) : \hat{S} \preccurlyeq T\}$ with lower right endpoint $\mathbf{u}_0$ and upper left endpoint $\mathbf{u}_1$. Then there exist S_0, $S_1 \preccurlyeq T$ such that $\mathbf{u}(S_0) = \mathbf{u}_0$, $\mathbf{u}(S_1) = \mathbf{u}_1$, and $S_1 \preccurlyeq S_0$.*

Proof Let $\mathcal{T}_F = \{S : S \preccurlyeq T,\ \mathbf{u}(S) \in F\}$ be the set of all trimmed subtrees of T lying on the face F of the lower convex hull. Repeated application of Lemma 1 yields $\mathbf{u}(\cap\mathcal{T}_F)$, $\mathbf{u}(\cup\mathcal{T}_F) \in F$. Clearly, $\cap\mathcal{T}_F \preccurlyeq S \preccurlyeq \cup\mathcal{T}_F$ for all $S \in \mathcal{T}_F$. Therefore, by monotonicity of average rate or MSE, $\mathbf{u}_1 = \mathbf{u}(\cap\mathcal{T}_F)$ and $\mathbf{u}_0 = \mathbf{u}(\cup\mathcal{T}_F)$. □

REFERENCES

[1] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: a review," *IEEE Trans. Comm.,* vol. 36, no. 8, pp. 957–971, 1988.

[2] A. Buzo, A. H. Gray, Jr., R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Inform. Theory,* vol. 28, no. 5, pp. 562–574, 1980.

[3] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inform. Theory,* vol. 35, no. 2, pp. 299–315, 1989.

[4] D. J. Vaisey and A. Gersho, "Variable block-size image coding," in *Proc. ICASSP-87,* vol. 2, pp. 1051–1054, 1987.

[5] S. M. Perlmutter and R. M. Gray, "A low complexity multiresolution approach to image compression using pruned nested tree-structured vector quantization," in *Proc. ICIP-94,* vol. 1, pp. 588–592, 1994.

[6] W.-Y. Chan and A. Gersho, "Constrained-storage quantization of multiple vector sources by codebook sharing," *IEEE Trans. Comm.,* vol. 39, no. 1, pp. 11–13, 1991.

[7] D. F. Lyons, D. L. Neuhoff, and D. Hui, "Reduced storage tree-structured vector quantization," in *Proc. ICASSP-93,* vol. 5, pp. 602–5, 1993.

[8] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-constrained vector quantization," *IEEE Trans. Acoust. Speech Signal Proc.,* vol. 37, no. 1, pp. 31–42, 1989.

[9] K. Rose, D. Miller, and A. Gersho, "Entropy-constrained tree-structured vector quantizer design," *IEEE Trans. Image Process.,* vol. 5, no. 2, pp. 393–398, 1996.

[10] W.-J. Hwang and H. Derin, "Multistage storage- and entropy- constrained tree-structured vector quantization," *IEEE Trans. Signal Process.* To appear.

[11] D. Y. Wong, B.-H. Juang, and A. H. Gray, Jr., "An 800 bit/s vector quantization LPC vocoder," *IEEE Trans. Acoust. Speech Signal Process.,* vol. 30, no. 5, pp. 770–779, 1982.

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

[13] T. Lookabaugh and R. M. Gray, "High-resolution quantization theory and the vector quantizer advantage," *IEEE Trans. Inform. Theory,* vol. 35, no. 5, pp. 1020–1033, 1989.

[14] P. Zador, "Asymptotic quantization error of continuous signals and the quantization dimension," *IEEE Trans. Inform. Theory,* vol. 28, no. 2, pp. 139–149, 1982.

[15] P. A. Chou, "Application of information theory to pattern recognition and the design of decision trees and trellises," Ph.D. dissertation, University of California, Stanford, CA, 1988.

[16] E. A. Riskin and R. M. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Trans. Signal Process.,* vol. 39, no. 11, pp. 2500–2507, 1991.

[17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees. The Wadsworth Statistics/Probability Series.* Belmont, CA: Wadsworth, 1984.

[18] S.-Z. Kiang, R. L. Baker, G. J. Sullivan, and C.-Y. Chiu, "Recursive optimal pruning with applications to tree structured vector quantizers," *IEEE Trans. Image Process.,* vol. 1, no. 2, pp. 162–169, 1992.