Optimal Pruning with Applications to Tree-Structured Source Coding and Modeling

PHILIP A. CHOU, TOM LOOKABAUGH, AND ROBERT M. GRAY, FELLOW, IEEE

Abstract — An algorithm recently introduced by Breiman, Friedman, Olshen, and Stone in the context of classification and regression trees is reinterpreted and extended to cover a variety of applications in source coding and modeling in which trees are involved. These include variable-rate and minimum-entropy tree-structured vector quantization, minimum expected cost decision trees, variable-order Markov modeling, optimum bit allocation, and computer graphics and image processing using quadtrees. A concentration on the first of these and a detailed analysis of variable-rate tree-structured vector quantization are provided. We find that variable-rate tree-structured vector quantization outperforms not only the fixed-rate variety but also full-search vector quantization as well. Furthermore, the “successive approximation” character of variable-rate tree-structured vector quantization permits it to degrade gracefully if the rate is reduced at the encoder. This has applications to the problem of buffer overflow.

I. INTRODUCTION

A $k$-BIT UNIFORM scalar quantizer on the interval $[0, 1]$ can be represented by a complete binary tree of length $k$, where each of the $2^k$ leaves corresponds to a quantization bin. Using this representation, we assign a number $x \in [0, 1]$, say $x = 1/3$, to a bin as follows: from the root node, go to the left child if $x < 1/2$; from that node, go to the right child if $x > 1/4$, and so on. The $k$-bit path map from the root to the leaves is sent through the channel by the encoder; the decoder first uses the path map to retrace the encoder’s steps to the appropriate leaf, then outputs the center of the corresponding bin. Analog-to-digital converters implemented in this way are commonly known as successive approximation quantizers [1]. These tree-structured scalar quantizers have been generalized to tree-structured vector quantizers (TSVQ) by Buzo et al. [2]. Called tree-search vector quantizers in [2], they are more appropriately called tree-structured vector quantizers to distinguish them from recent tree-structured search methods [3], [4] which need not have a “successive approximation” character.

Previous tree-structured quantizers (both vector and scalar) have had a fixed rate since they were based on trees of fixed length. In this paper we generalize them to variable-rate tree-structured quantizers simply by varying the length of the tree. The central problem studied here is how to find the particular tree with a prescribed average length which minimizes the average distortion. Forming a variable-rate code as a subtree of a TSVQ provides the code with the “successive approximation” character of a TSVQ. Thus the code can degrade gracefully if the channel rate is reduced, simply by sending shorter words and suffering the minimal increase in distortion. This is accomplished without fundamentally changing the code and, since the code is instantaneous, the decoder can track the change without side-information.

Our approach is to use an algorithm recently introduced by Breiman, Friedman, Olshen, and Stone (BFOS) in the context of classification and regression trees [5]. Their algorithm begins with an initial large tree, and prunes back until it reaches the subtree with the fewest number of leaves for a given probability of error. We demonstrate the generality of their algorithm by extending it to cover a variety of applications in source coding and modeling in which trees are involved. These include not only variable-rate tree-structured vector quantization, but also minimum entropy vector quantization, vector quantization of noisy sources, minimum expected cost decision trees, variable-length Markov modeling, optimum bit allocation, fast pattern matching, and computer graphics and image processing using quadtrees.

In the next section we cast our objectives in an information-theoretic framework and define the operational distortion-rate function for tree-structured quantizers. In Section III we introduce the necessary notation and the concept of tree functionals. In Section IV we give the generalized BFOS algorithm and interpret it geometrically: optimal subtrees are found by searching the vertices of a convex polygon. In source coding problems this has the interpretation of tracing out the convex hull of the operational distortion-rate function. Section V is an introduction to the theory behind probabilistic trees, while Section VI contains various applications of the pruning algorithm to such trees. One such application, variable-rate tree-structured vector quantization, is analyzed in detail in the
to determine how best to trade off the rate and distortion by varying the size of the tree.

According to distortion-rate theory [7], [8] for memoryless sources, if the distortion measure \( \rho(x, y) \) and the source distribution \( P_x \) are fixed, then every test channel \( P_{y|x} \) defines a rate \( I(X; Y) \) and a distortion \( E[\rho(X, Y)] \), which are achievable by some deterministic block code (not necessarily tree-structured) in the limit of large blocklength. The distortion-rate function

\[
D(R) = \inf_{P_{y|x}} \{ E[\rho(X, Y)] | I(X; Y) \leq R \}
\]  

(1)

tightly lowerbounds this achievable region in the distortion versus rate plane, and hence specifies the optimal trade-off between the rate and distortion, in the limit of large blocklength. (Sources with memory need an additional limit.)

We are more interested in the set of (rate, distortion) pairs achievable by all pruned subtrees of a given tree-structured vector quantizer for a fixed blocklength. More specifically, if \( T \) is a large tree, there corresponds to every pruned subtree \( S \) of \( T \) (denoted \( S \subseteq T \)) a variable-rate tree-structured vector quantizer with average rate \( I(S) \) and average distortion \( \delta(S) \); the operational distortion-rate function

\[
\hat{D}_T(R) = \min_{S \subseteq T} \{ \delta(S) | I(S) \leq R \}
\]  

(2)
	hen specifies the optimal trade-off between rate and distortion in the restricted setting, where the quantizer is constrained to be some pruned subtree of a given tree \( T \). For the time being we will not be concerned with finding the initial tree \( T \). In some applications, such as variable-order Markov modeling, the initial tree is obvious. In other applications, such as classification, the problem of finding the optimal tree is NP-complete (i.e., complete in nondeterministic polynomial time) [9], so it is likely that no tractable algorithm for finding the optimal tree exists. A common approach in these cases is to grow the tree using a stepwise optimal “greedy” heuristic, as in [5], for classification and regression trees, or [2], for tree-structured vector quantizers.

Because any tree \( T \) has a finite number of pruned subtrees, the operational distortion-rate function \( \hat{D}_T(R) \) has the staircase form shown in Fig. 2. Thus the function is not convex; in fact some of its “steps” may lie strictly above the convex hull of achievable points.

Because of this nonconvexity, computing the function \( \hat{D}_T(R) \) essentially requires an exhaustive search over all pruned subtrees of the given tree \( T \). In contrast, determining the function \( D(R) \) is simply a convex programming problem; in the finite-dimensional case, the Blahut algorithm [10] can be used.

The Blahut algorithm and more general convex programming algorithms for finding \( D(R) \) are based on minimizing the Lagrangian \( J(P_{y|x}) = E[\rho(X, Y)] + \lambda I(X; Y) \), where the Lagrange multiplier \( \lambda \) is interpreted as the slope of the hyperplane supporting the convex achievable region. One suspects that similar algorithms may exist for finding
If one were interested only in the convex hull, this is indeed the case. The generalized BFOS algorithm is Lagrangian in flavor, in that it minimizes the functional

$$J(S) = \delta(S) + \lambda I(S)$$

over all pruned subtrees $S$ of $T$. Again, $\lambda$ is interpreted as the slope of a hyperplane supporting the achievable set. Thus the algorithm is capable of listing the extreme points of the convex hull of the operational distortion-rate function, along with their associated subtrees. Although the convex hull is only a lower bound on the operational distortion-rate function, its extreme points lie on $\hat{D}_{r}(R)$. Hence almost any point on the convex hull can be achieved by time sharing between two subtrees.

A system designer can use the algorithm first to decide on the appropriate level at which to trade off rate for distortion, and then to obtain the optimum subtree or subtrees at that level. Time sharing may be necessary to achieve the minimum distortion for a given rate. However, time sharing is not usually necessary in practice because the target rate is often very close to the rate of some optimally pruned subtree.

A piecewise linear function passing through the list of points produced by the algorithm must be convex, although taking the logarithm of either axis may destroy this convexity. Taking the logarithm is common in practice because absolute distortion is often converted to a signal-to-quantization noise ratio in decibels. All of our plots in Section VIII are of this form. Such a conversion in no way affects the optimality of the pruned subtrees, which in any case must lie on $\hat{D}_{r}(R)$.

### III. Monotonic Affine Tree Functionals

Real-valued functions on trees and their subtrees, such as the average length $l(S)$ or the average distortion $\delta(S)$, are known as tree functionals. These and many other interesting tree functionals share two desirable properties: linearity and monotonicity. A tree functional is said to be linear if it is the sum of values at the leaves; it is affine if it is the sum of values at all nodes. A tree functional is monotonic if it increases or decreases monotonically as the tree grows. This section formalizes these notions and introduces some necessary notation.

Rigorous definitions of tree, node, root, leaf, parent, child, ancestor, descendant, path height, etc., can be found in [11]-[13]. We adopt the looser definitions and notation of Breiman et al. [5], in which a tree $T$ is simply a finite set of nodes, $T = \{t_0, t_1, t_2, \ldots \}$, with a unique root node, designated by $t_0$. The set of leaves of a tree $T$ is denoted by $\ell$. Thus $\ell$ is a subset of $T$. In general, a subtree $S$ of a tree $T$ is a tree rooted at some node $t \in T$ whose leaves $\ell$ are not necessarily a subset of $\ell$. (They may be interior nodes of $T$.) There are two special kinds of subtrees of $T$. If indeed $S \subseteq T$, then $S$ is called a branch of $T$ from node $t$, and is denoted by $T_t$. If on the other hand $t = t_0$, then $S$ is called a pruned subtree of $T$; in this case we write $S \preceq T$. (If both cases occur, then $S = T_{t_0} = T$.) If a subtree consists of only the single node $t$, then with a slight abuse of notation we use $t$ to refer to the subtree ($t$) itself. These definitions are illustrated in Fig. 3.
Let \( u \) be a tree functional defined on the set of all subtrees of \( T \), and suppose it has the following recursive decomposition. The value of \( u \) on subtrees of height 0 (singletons) and subtrees of height 1 (a root plus its children) can be arbitrary, but the value of \( u \) on a subtree \( S \) of height greater than 1 must be expressible as

\[
\begin{align*}
u(S) &= u(R) + \sum_{t \in \hat{R}} [u(S_t) - u(t)] \\
u(S) &= u(R) + \sum_{t \in \hat{R}} \Delta u(S_t) 
\end{align*}
\]

where \( R \) is the height-1 pruned subtree of \( S \), \( S_t \) is the branch of \( S \) from node \( t \in \hat{R} \), and by definition, \( \Delta u(S) = u(S) - u(\text{root}(S)) \). A tree functional with this decomposition property will be called affine. It can be shown by induction that an equivalent definition is given by (3), where \( R \) need not be restricted to be height-1, i.e., \( R \) can be an arbitrary pruned subtree of \( S \). Other equivalent definitions include

\[
\Delta u(S) = \Delta u(R) + \sum_{t \in \hat{R}} \Delta u(S_t),
\]

(4)

derived from (3) by subtracting \( u(\text{root}(S)) = u(\text{root}(R)) \) from both sides, and

\[
u(S) = a(R) + \sum_{t \in \hat{R}} u(S_t),
\]

(5)

derived from (3) by regrouping terms into \( a(R) = u(R) - \sum_{t \in \hat{R}} u_t(t) \). If \( a(R) = 0 \), the tree functional will be called linear. Subaffine and sublinear tree functionals, in which the equality is replaced by an inequality, can also be handled by our theory in most cases of interest. For the sake of exposition, however, we will focus on the case of equality.

The number of nodes in a tree is a simple example of an affine tree functional. If \( u(S) \) is the number of nodes in \( S \), and \( R \) is any pruned subtree of \( S \), then \( u(S) \) can be expressed by (5) as \( u(R) + \sum_{t \in \hat{R}} u_t(t) \), where \( a(R) \) is the number of interior nodes of \( R \). (When \( R \) is height-1, \( a(R) = 1 \). See also Table I.) Equivalently, \( u(S) \) can be expressed by (3) as \( u(R) + \sum_{t \in \hat{R}} \Delta u(S_t) \), where \( \Delta u(S_t) = u(S_t) - u(t) = u(S_t) - 1 \) is nonnegative. This nonnegativity guarantees that as the tree grows, the functional does not decrease. Such an affine tree functional is said to be monotonically increasing. In general, a tree functional is said to be monotonically increasing if \( u(R) \leq u(S) \) whenever \( R \) is a subtree of \( S \). If the functional is affine, this is indicated by \( \Delta u(S) \geq 0 \) for all subtrees \( S \) of \( T \). Similar definitions hold for monotonically decreasing functionals.

Another example of a monotonically increasing affine tree functional is the number of leaves in a tree. If \( u(S) \) is the number of leaves in \( S \), and \( R \) is any pruned subtree of \( S \), then \( u(S) \) can be expressed by (5) with \( a(R) = 0 \). Hence \( u(S) \) is in fact linear. It is monotonically increasing because the number of leaves always increases as the tree grows.

Other examples of monotonic affine tree functionals include the average length \( l(S) \) and the average distortion \( d(S) \), but their exact definitions in terms of \( \Delta u \) and \( a(R) \) will be deferred until the discussion on probabilistic trees in Section V. These and other probabilistic tree functionals are shown in Table I.

### IV. The Algorithm

Let \( u_1 \) and \( u_2 \) be two monotonic affine tree functionals, with \( \Delta u_1(S) \geq 0 \) (\( u_1 \) increasing) and \( \Delta u_2(S) \leq 0 \) (\( u_2 \) decreasing). A typical example is \( u_1(S) = l(S) \) and \( u_2(S) = d(S) \). Let \( u(S) = (u_1(S), u_2(S)) \) be the vector-valued function on the set of all subtrees of \( T \), with \( u_1 \) and \( u_2 \) as components. Then consider the set of points \( \{u(S) \mid S \in T \} \) and their convex hull. When \( u_1(S) = l(S) \) and \( u_2(S) = d(S) \), this is just the set of all pruned subtrees of \( T \), plotted in the distortion-rate plane, and their convex hull. By monotonicity, the singleton tree consisting of just the root \( t_0 \) has the smallest \( u_1 \) and the largest \( u_2 \); the full tree \( T \) itself has the largest \( u_1 \) and the smallest \( u_2 \). Therefore \( u(t_0) \) is the upper left corner of the convex hull; \( u(T) \) is the lower right corner. This is the case shown in Fig. 2, where \( u_1 \) represents the average rate and \( u_2 \) represents the distortion.

It is a remarkable fact and the key to the algorithm that if \( u(T), u(S_1), u(S_2), \ldots, u(S_n), u(t_0) \) is the list of vertices clockwise around (the lower boundary of) the convex hull,

### Table I

**Tree Functionals**

<table>
<thead>
<tr>
<th>Description</th>
<th>( u(T) )</th>
<th>( u(t) )</th>
<th>( u(R) )</th>
<th>( \Delta u(R) )</th>
<th>( a(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Number of leaves</td>
<td>(</td>
<td>T</td>
<td>)</td>
<td>1</td>
<td>(</td>
</tr>
<tr>
<td>(b) Number of nodes</td>
<td>(</td>
<td>T</td>
<td>)</td>
<td>1</td>
<td>(</td>
</tr>
<tr>
<td>(c) Number of interior nodes</td>
<td>(</td>
<td>T - \hat{T}</td>
<td>)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(d) Average length</td>
<td>( E[l(x_T)] )</td>
<td>( P(t \cdot l(t)) )</td>
<td>( \sum_{t \in \hat{R}} P(t \cdot l(t)) )</td>
<td>( P(t) )</td>
<td>0</td>
</tr>
<tr>
<td>(e) Average cost</td>
<td>( E[c(x_T)] )</td>
<td>( P(t \cdot c(t)) )</td>
<td>( \sum_{t \in \hat{R}} P(t \cdot c(t)) )</td>
<td>( P(t) \cdot c(t) )</td>
<td>0</td>
</tr>
<tr>
<td>(f) Shannon entropy</td>
<td>( H(x_T) )</td>
<td>( - P(t) \log P(t) )</td>
<td>( - \sum_{t \in \hat{R}} P(t) \log P(t) )</td>
<td>( P(t) \cdot H(x_T) )</td>
<td>0</td>
</tr>
<tr>
<td>(g) Relative entropy</td>
<td>( H_{\text{R}}(x_T) )</td>
<td>( P(t) \log [P(t)/Q(t)] )</td>
<td>( \sum_{t \in \hat{R}} P(t) \log [P(t)/Q(t)] )</td>
<td>( P(t) - H_{\text{R}}(x_T) )</td>
<td>0</td>
</tr>
<tr>
<td>(h) Mutual information</td>
<td>( I(Y; x_T) )</td>
<td>( P(t) \cdot H(Y) - H(Y) )</td>
<td>( \sum_{t \in \hat{R}} P(t) \cdot H(Y) - H(Y) )</td>
<td>( P(t) \cdot I(Y; x_T) )</td>
<td>0</td>
</tr>
<tr>
<td>(i) Average distortion</td>
<td>( E[d(x_T)] )</td>
<td>( P(t) \cdot d(t) )</td>
<td>( \sum_{t \in \hat{R}} P(t) \cdot d(t) )</td>
<td>( u(R) - u(t) )</td>
<td>0</td>
</tr>
<tr>
<td>(j) Conditional entropy</td>
<td>( H(x_T</td>
<td>y_T) )</td>
<td>( P(t) \cdot H(Y</td>
<td>x_T) )</td>
<td>( \sum_{t \in \hat{R}} P(t) \cdot H(Y</td>
</tr>
</tbody>
</table>

*Functional (b) and (c) are affine; the rest are linear. Functionals (a)–(h) are monotonic nondecreasing; functionals (i) and (j) are monotonic nonincreasing. For the definitions of \( u(t), u(R), \Delta u(R), \) and \( a(R) \); see (3), (4), (5). (In column \( \Delta u(R) \), \( t \) is the root of \( R \).)*
then $t_0 \leq S_0 \leq \cdots \leq S_2 \leq S_1 \leq T$. Hence it is possible to start with the full tree at $u(T)$ and prune back to the root at $u(t_0)$, producing a list of nested subtrees which trace out the vertices of the lower boundary of the convex hull. (Although the upper boundary could be traced in a similar manner, we are less interested in it since it represents the worst possible $u_1-u_2$ trade-off.) Lemma 1 of the Appendix and its corollary fact.

It remains to show how to obtain $S_{i+1}$ by pruning back $S_i$. Actually, we will show only how to obtain $S'$ by pruning back $S$, where $S$ and $S'$ are subtrees on the convex hull with $S_{i+1} \leq S' \leq S_i$ and $S' \neq S$. The algorithm will use this procedure repeatedly, starting with $T$, until $t_0$ is reached, which occurs after a finite number of steps since there is only a finite number of subtrees.

Suppose we are at a point $u(S)$ on face $F$ of the convex hull. Let $\lambda$ be the maximum of the slope of $F$. Consider the set of all pruned subtrees of $S$ obtained by pruning off a single branch $S_t$ from some interior node $t \in S$. Each interior node $t \in S$ corresponds to a pruned subtree, say $R(t)$, in this set. Then $u(S) = u(R(t)) + \Delta u(S_t)$ for each $R(t)$, by (3). Hence, as illustrated in Fig. 4, each vector $\Delta u(S_t)$ must have slope $\Delta u_t(S_t)/\Delta u(S_t)$ no smaller in magnitude than $\lambda$, else some $u(R(t))$ would lie outside the convex hull. In fact, Lemma 2 of the Appendix shows that there exists at least one $\Delta u_t(S_t)$ with slope precisely equal to $\lambda$ in magnitude. In particular, $u(R(t)) \in F$ for all $t \in S_{i+1}$.

Hence the algorithm only needs to find the interior node $t \in S$ for which the magnitude of the slope of $\Delta u(S_t)$ is a minimum. For this $t$, the slope of $\Delta u(S_t)$ coincides with the slope of $F$, and the subtree $S' = R(t)$, obtained by pruning off the branch $S_t$, lies on $F$. When $u(S)$ can take on a continuum of values, it is unlikely that there will be two $\Delta u_t(S_t)$'s with the same slope; thus $S' = S_{i+1}$. When $u(S)$ is discrete-valued, however, it is quite possible that several $\Delta u_t(S_t)$'s will have the same slope. All of the branches $S_t$ will have to be pruned off before reaching $S'$. It is for this reason we have posed only the general problem of obtaining $S'$ from $S$, where $S_{i+1} \leq S' \leq S_i$.

To make matters simple, we now assume that the tree $T$ is binary. This, however, is by no means necessary. The tree can be stored as an array of nodes, each containing the following information:

$$
\begin{bmatrix}
\Delta u(S_t) \\
\lambda(t) \\
\min_{\text{left}(t)}(\lambda(t)) \\
\min_{\text{right}(t)}(\lambda(t))
\end{bmatrix}
$$

The (negative) change in $u(\cdot)$ obtained by pruning off the branch $S_t$ from node $t$ is stored in $\Delta u(S_t) = (\Delta u_1(S_t), \Delta u_2(S_t))$. The magnitude of the slope of $\Delta u_t(S_t)$ is stored in $\lambda_t = -\Delta u_2(S_t)/\Delta u_1(S_t)$. The minimum such magnitude, over all interior nodes of the branch $S_t$, is stored in $\lambda_{\text{min}}(t) = \min \{\lambda(t), \lambda_{\text{min}}(t_t), \lambda_{\text{min}}(t_R)\}$, where $t_L = \text{left}(t)$ and $t_R = \text{right}(t)$ are the left and right children of $t$. Pointers to $\text{left}(t)$, $\text{right}(t)$, and $\text{parent}(t)$ are usually also necessary unless they can be computed from $t$, e.g., $\text{left}(t) = t_{2i+1}$, $\text{right}(t) = t_{2i+2}$, and $\text{parent}(t) = t_{(i-1)/2}$. The following one-time initialization is performed.

For each leaf node $t$,

$$
\Delta u(S_t) \leftarrow 0,
$$

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

For each interior node $t$,

$$
\Delta u(S_t) \leftarrow u(S_t) - u(t),
$$

$$
\lambda(t) \leftarrow -\Delta u_2(S_t)/\Delta u_1(S_t),
$$

$$
\lambda_{\text{min}}(t) \leftarrow \min \{\lambda(t), \lambda_{\text{min}}(t_t), \lambda_{\text{min}}(t_R)\}.
$$

The minimum $\lambda(t)$, over all interior nodes of the tree, is maintained in $\lambda_{\text{min}}(t_0)$. So, starting with the full tree $T$, branches are successively pruned off until $\lambda_{\text{min}}(t_0) = \infty$, indicating that only the root node, $t_0$, remains. Here is the complete algorithm:

$$
u \leftarrow u(t_0) + \Delta u(S_t),
$$

while $(\lambda_{\text{min}}(t_0) < \infty)$ do

$$
t \leftarrow t_0
$$

while $(\lambda(t) > \lambda_{\text{min}}(t_0))$ do

$$
\text{if } \lambda_{\text{min}}(t_t) = \lambda_{\text{min}}(t_0)
$$

then $t \leftarrow t_L$

else $t \leftarrow t_R$

end

$$
\Delta \leftarrow \Delta u(S_t),
$$

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

while $(t \neq t_0)$ do

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

$$
\Delta u(S_t) \leftarrow \Delta u(S_t) - \Delta,
$$

$$
\lambda(t) \leftarrow -\Delta u_2(S_t)/\Delta u_1(S_t),
$$

$$
\lambda_{\text{min}}(t) \leftarrow \min \{\lambda(t), \lambda_{\text{min}}(t_t), \lambda_{\text{min}}(t_R)\}
$$

end

$$
u \leftarrow u - \Delta,
$$

print $\lambda_{\text{min}}(t_0), u$

end.
The vector variable $u$ maintains a running account of $u(t)$ for the tree as it is pruned. The first inner “while” loop searches a path beginning at the root for the node with the minimum $\lambda(t)$. When it is found, the branch from this node is pruned off and the change in $u(t)$ is recorded in the vector variable $\Delta$. The second inner “while” loop retraces the path back to the root, updating all ancestors to reflect the pruning. Finally, $u$ is updated and printed out, along with the slope $\lambda_{\min}(t_0)$ if desired.

The complexity of the initialization step is $O(N)$, where $N$ is the number of nodes in the tree. The main loop of the algorithm is performed somewhere between $M$ and $N$ times, where $M$ is the number of extreme points around the lower boundary of the convex hull. Let us say $O(N)$ times, to be conservative. Each inner loop of the algorithm is performed $O(\log N)$ times. In sum, the complexity of the algorithm is $O(N \log N)$.

If one is only interested in minimizing $u(T(\cdot)) + \lambda_u(T)$ over all $S \subseteq T$ for a single slope $\lambda = \lambda_u$, then the following algorithm performs the minimization in time $O(N)$:

$$\text{MINCOST}(T) = \min \left\{ u(t_0), a(R) + \sum_{t \in R} \text{MINCOST}(T_t) \right\}.$$

The MINCOST algorithm can be embedded in a control loop that employs the method of bisection, Newton’s method, or other means to find every pruned subtree $S \subseteq T$ partition $\Omega$ into $|S|$ events. In fact, the leaves of every pruned subtree $S \subseteq T$ partition $\Omega$ into $|S|$ events. Thus we can speak of nodes as events: the root $t_0$ is the whole space $\Omega$, its children partition the space into $d_0$ events, and the leaves partition the space into $|T|$ events. In fact, the leaves of every pruned subtree $S \subseteq T$ partition $\Omega$ into $|S|$ events. However, the meaning will always be clear from context.

In a natural way, the partition $S$ induces the random variable $\psi_S$, which takes the node value $t \in S$ with probability $P(t)$. For example, $\psi_S$ has an entropy $H(\psi_S) = E[-\log P(\psi_S)] = -\sum_{t \in S} P(t) \log P(t)$. The entropy of the entire tree is $H(\psi_T)$. Any function of a discrete random variable is also a random variable; thus, for example, if $c(t)$ is the cost of node $t$, then $E[c(\psi_S)]$ is the expected cost of the tree. In particular, if $l(t)$ is the length of the path from the root to node $t$, then $E[l(\psi_S)]$ is the expected length of the tree. If $d(t)$ is the distortion at node $t$, then $E[d(\psi_S)]$ is the expected distortion of the tree.

Conditioning on an event is possible as usual. $P(A|B) = P(A \cap B)/P(B)$ denotes the elementary conditional probability of the event $A$, given the event $B$ (provided $P(B) > 0$). If $S' \subseteq S$, $t' \in S'$, and $t \in S$, then $S'$ is a refinement of $S$ (as partitions), either $t' \cap t = t$ or $t \cap t' = 0$ (as events). Hence either $P(t|t') = P(t)/P(t')$ or $P(t|t') = 0$, depending on whether or not $t$ is a descendant of $t'$. However, the entropy of $\psi_S$, conditioned on the event $t' \in S', S' \subseteq S$, is $E[c(\psi_S)] = -\sum_{t \in S} P(t) \log P(t')$. In addition to the usual entropy of a partition, we have the following conditional entropies. The entropy of $\psi_S$, conditioned on the event $t' \in S', S' \subseteq S$, is denoted by $H(\psi_{S'|t'}) = -\sum_{t \in S'} P(t) \log P(t')$, and the conditional entropy of $\psi_S$, given the random variable $\psi_{S'}$, is denoted by $H(\psi_S|\psi_{S'}) = -\sum_{t \in S} P(t) H(\psi_{S'|t})$.

Let $Y$ be another random variable on $\Omega$, with alphabet $A_Y$. Then $Y, \psi_S, \psi_T$, all $f(X)$ for $f \in \mathcal{F}$, etc., are all jointly distributed random variables. In analogy to the aforementioned definitions of entropy, we can define the entropies $H(Y)$, $H(Y|t)$, and $H(Y|\psi_S)$, if $A_Y$ is finite. Similarly, we can define the mutual informations: $I(Y; \psi_S)$, $I(Y; \psi_T)$, and $I(Y; \psi_S|\psi_T)$. These exist even if $A_Y$ is neither finite nor countably infinite.

Let $\rho: A_Y \times A_Y \to \mathbb{R}^+$ be a distortion measure on $A_Y$, that is, a nonnegative mapping of pairs into reals. Then if $y \in A_Y$ represents $Y$ at node $t$, the conditional distortion at node $t$ equals $E[\rho(Y, y)|t]$. We shall always pick the representative $y \in A_Y$, called the centroid, which minimizes this conditional distortion, and define $d(t) = \inf_y \rho(y, t)$. With this definition, the linear tree

V. Probabilistic Trees

Let $(\Omega, \mathcal{A}, P)$ be a probability space underlying the random variable $X: \Omega \to A_X$, and let $\mathcal{F}$ be a possibly uncountable set of measurable functions on $A_X$. Assume each $f \in \mathcal{F}$ has range $[0, 1, \ldots, d-1]$, where $d$ is a small positive integer in general depending on $f$. A tree $T$ can be defined on this space in the following way. Each interior node $t_i$ with $d_i$ children corresponds to a measurable function $f_i \in \mathcal{F}$ with range $[0, 1, \ldots, d_i-1]$, by associating the outcome $f_i(x) = j$ with the $j$th child of $t_i$. Thus each node of $T$, including the leaves, represents an event in $\mathcal{A}$. For example, a node $t$ at depth $n$ with ancestral path $(t_0, t_1, \ldots, t_n, t)$ represents the event \{ $f_0(X) = j_0$ \} \cap \{ $f_1(X) = j_1$ \} \cap \cdots \cap \{ $f_n(X) = j_n$ \}, where $t_i$ is the $j_i$th child of $t_0$, $t_i$ is the $j_i$th child of $t_{i-1}$, etc. We can speak of nodes as events: the root $t_0$ is the whole space $\Omega$, its children partition the space into $d_0$ events, and the leaves partition the space into $|T|$ events. In fact, the leaves of every pruned subtree $S \subseteq T$ partition $\Omega$ into $|S|$ events. However, the meaning will always be clear from context.

In a natural way, the partition $S$ induces the random variable $\psi_S$, which takes the node value $t \in S$ with probability $P(t)$. For example, $\psi_S$ has an entropy $H(\psi_S) = E[-\log P(\psi_S)] = -\sum_{t \in S} P(t) \log P(t)$. The entropy of the entire tree is $H(\psi_T)$. Any function of a discrete random variable is also a random variable; thus, for example, if $c(t)$ is the cost of node $t$, then $E[c(\psi_S)]$ is the expected cost of the tree. In particular, if $l(t)$ is the length of the path from the root to node $t$, then $E[l(\psi_S)]$ is the expected length of the tree. If $d(t)$ is the distortion at node $t$, then $E[d(\psi_S)]$ is the expected distortion of the tree.

Conditioning on an event is possible as usual. $P(A|B) = P(A \cap B)/P(B)$ denotes the elementary conditional probability of the event $A$, given the event $B$ (provided $P(B) > 0$). If $S' \subseteq S$, $t' \in S'$, and $t \in S$, then $S'$ is a refinement of $S$ (as partitions), either $t' \cap t = t$ or $t \cap t' = 0$ (as events). Hence either $P(t|t') = P(t)/P(t')$ or $P(t|t') = 0$, depending on whether or not $t$ is a descendant of $t'$. However, the entropy of $\psi_S$, conditioned on the event $t' \in S', S' \subseteq S$, is $E[c(\psi_S)] = -\sum_{t \in S} P(t) \log P(t')$. In addition to the usual entropy of a partition, we have the following conditional entropies. The entropy of $\psi_S$, conditioned on the event $t' \in S', S' \subseteq S$, is denoted by $H(\psi_{S'|t'}) = -\sum_{t \in S'} P(t) \log P(t')$, and the conditional entropy of $\psi_S$, given the random variable $\psi_{S'}$, is denoted by $H(\psi_S|\psi_{S'}) = -\sum_{t \in S} P(t) H(\psi_{S'|t})$.

Let $Y$ be another random variable on $\Omega$, with alphabet $A_Y$. Then $Y, \psi_S, \psi_T$, all $f(X)$ for $f \in \mathcal{F}$, etc., are all jointly distributed random variables. In analogy to the aforementioned definitions of entropy, we can define the entropies $H(Y)$, $H(Y|t)$, and $H(Y|\psi_S)$, if $A_Y$ is finite. Similarly, we can define the mutual informations: $I(Y; \psi_S)$, $I(Y; \psi_T)$, and $I(Y; \psi_S|\psi_T)$. These exist even if $A_Y$ is neither finite nor countably infinite.

Let $\rho: A_Y \times A_Y \to \mathbb{R}^+$ be a distortion measure on $A_Y$, that is, a nonnegative mapping of pairs into reals. Then if $y \in A_Y$ represents $Y$ at node $t$, the conditional distortion at node $t$ equals $E[\rho(Y, y)|t]$. We shall always pick the representative $y \in A_Y$, called the centroid, which minimizes this conditional distortion, and define $d(t) = \inf_y \rho(y, t)$. With this definition, the linear tree
functional $\delta(T) = E[d(\psi_T)]$ is monotonic decreasing, for
$$
\delta(S) = \inf_{\gamma \in A_r, \tau \in S} \sum_{\gamma \in A_r} E[p(Y, y)] \rho(\tau) 
$$
is never greater than
$$
\delta(\text{root}(S)) = \inf_{\gamma \in A_r, \tau \in S} \sum_{\gamma \in A_r} E[p(Y, y)] \rho(\tau) 
$$
$$
= \inf_{\gamma \in A_r} E[p(Y, y)] \rho(\text{root}(S)).
$$

In practice, the probability measure $P$ underlying the random variables $X$ and $Y$ is often an empirical probability measure giving weight $1/N$ to each of the points $(Y_1, X_1), \ldots, (Y_N, X_N)$ in a representative training sequence. A separate test sequence $(Y_{N+1}, X_{N+1}), \ldots, (Y_{N+M}, X_{N+M})$ from the same process is then necessary to test the consistency of the results. Assuming the process is stationary, performance results on the two sequences should agree. If they do not, then both $N$ and $M$ may have to be increased.

VI. APPLICATIONS

A variety of monotonic affine tree functionals can be found in Table I. Many of these are used in the following applications.

A. Tree-Structured Quantization

As discussed in Section II, complete tree-structured quantizers can be pruned back to yield variable-rate tree-structured quantizers in which the path map is used as the channel code. To perform the pruning optimally in a distortion-rate sense, the generalized BFOS algorithm can be used with $u_1$ as the expected length of the code, or equivalently, the expected length of the tree (d) of Table I, and $u_2$ as the expected distortion (i) of Table I. The resulting sequence of pruned subtrees will lie on the convex hull of the operational distortion-rate function $D_f(R)$. In the vector case with binary trees, optimal pruning provides a significant improvement in the signal to quantization noise (SQNR) ratio. Such variable-rate tree-structured quantizers have the desirable feature, not found in variable-rate quantizers based on entropy coding, of graceful degradation in the face of buffer overflow. This application will be investigated more fully in the following sections.

B. Regression Trees

In a closely related problem, we wish to model the real random variable $Y$ as a function of the real random variable $X$ plus an error term, $Z$; that is, $Y = q(X) + Z$. The function $q(\cdot)$ shall be chosen from a given family $\mathcal{F}$ to minimize the mean squared error $E[(Y - q(X))^2]$. This is called linear regression when the family $\mathcal{F}$ is the set of affine functionals on $\mathbb{R}$; otherwise it is called nonlinear regression. In the latter category is the set of all functions that are constant over each cell in a partition of $\mathbb{R}$ generated by the leaves of a binary tree. The internal nodes of these trees are required to partition $\mathbb{R}$ into two semi-infinite intervals, e.g., $X < a$ and $X \geq a$. Of course, it is always possible to decrease the mean squared error by increasing the length of the tree; thus there is a trade-off with the operational distortion-rate function.

Breiman et al. [5] have chosen good regression trees by growing them in a stepwise optimal fashion to a large initial length and then optimally pruning them using the equivalent of $u_1$ as the number of leaves (a) in Table I and $u_2$ as the mean squared error distortion (i) in Table I. Tree-structured quantization under the mean squared error distortion can be regarded as a special case of regression trees with $Y = X$. However, growing an initial large tree is much easier in the more restricted case. The regression problem can be generalized to vector $X$ by taking $E$ to be the set of all functions that are constant over each rectangular cell in a partition of $\mathbb{R}^n$ generated by the leaves of a binary tree, where each internal node tests only a single coordinate of $X$: $X < a$ or $X \geq a$. The problem can be further generalized to vector $Y$ by choosing $q \in \mathcal{F}$ to minimize $E[|Y - q(X)|^2]$ or some other more general distortion $E[p(Y, q(X))]$.

C. Quantization of Noisy Sources

Suppose that we can only observe a noisy version $X$ of the source $Y$, and that we wish to encode $X$ into a bit string with a given average length, send the bit string across a noisless channel, and reproduce it as closely on average to the source $Y$ as possible. Theoretically, this can be transformed into the standard quantization problem by using the modified distortion measure $p_M(x, y) = E[p(Y, y)|X = x]$. Unfortunately, this is usually intractable, except in the notable case of additive Gaussian noise under an input weighted quadratic distortion measure [14]. However, this is really just a nonlinear regression problem: we wish to find a piecewise constant function $q$ on $X$ that minimizes the expected square error $E[|Y - q(X)|^2]$ (or some other more general distortion $E[p(Y, q(X))]$) over all such functions whose cells of constant value can be encoded into bit strings with a given average length. A regression tree can be used to specify such a function in the natural way. First grow a large initial regression tree in the stepwise optimal fashion of Breiman et al. and prune back using $u_1$ as the expected length of the tree and $u_2$ as the expected distortion, as in the quantization problem. Then, for every noisy input $X$, map $X$ to the appropriate leaf and send the path map through the channel to the decoder. The decoder can use the path map to reconstruct the leaf and the output the appropriate value $q(X)$. Thus $q(\cdot)$ can be regarded as the tree-structured encoder/decoder for the noisy source that minimizes the overall average distortion subject to a constraint on the average rate.

D. Decision Trees

Let $Y$ be chosen with probability $P_Y$ from a finite alphabet $A_Y$, and conditioned on the outcome of $Y$, let $X$
be chosen with probability \( p_{x|y} \) from the alphabet \( A_X \) (usually \( \mathbb{R}^k \)). In the pattern-recognition literature, \( Y \) is called the unknown class and \( X \) is called the observed feature vector. A decision tree is a tree-structured classifier which attempts to infer \( Y \) by looking only at \( X \). If \( X \) is a vector, the internal nodes of these trees are usually constrained to test only one coordinate at a time. Each leaf \( t \) is labeled with the most probable class, i.e., the class \( y \) which maximizes \( P(Y=y|t) \). The decision tree problem is very similar to the regression tree problem, and the trees are grown in a similar stepwise optimal way. The decision trees grown in [5] are pruned using the equivalent of \( u_1 \) as the number of leaves ((a) in Table I) and \( u_2 \) as the probability of error ((i) in Table I). More generally, \( u_1 \) could be the expected cost of the tree ((e) in Table I) as in [13]. The pruned tree would then reflect the fact that observing certain components of \( X \) is expensive while observing certain other components is cheap. This becomes particularly relevant in decision trees for medical diagnosis. Sublinear costs can be modeled by a sublinear \( u_1 \).

E. Variable-Order Markov Modeling

Let \( \{ X_n \} \) be a stationary random process with a finite \( d \)-ary alphabet. A decision tree with \( Y = X_n \) and \( X = (X_{n-1}, X_{n-2}, \cdots) \) can be used as a predictor for the process. If each leaf \( t \) is labeled with the entire probability mass function \( P(Y=y|t) \) instead of the single letter \( y_1 \), which maximizes it, then the tree can be used as a model for the process. If, in addition, each internal node at depth \( l \) of the decision tree has \( d \) children, each corresponding to a different value of \( X_{n-l} \), then the tree is a variable-order Markov model for the process. The usual \( k \)-th-order Markov model corresponds to a complete tree of depth \( k \). Variable-order Markov models have been studied by Rissanen [15], Itoh and Kawabata [16], and Elnahas et al. [17] as context models for arithmetic coding. Rissanen applied the following minimum description length principle: the best tree-structured model for the distribution of \( X_n \), given the observations \( (X_1, \cdots, X_{n-1}) \), is the tree \( T \), which minimizes the total description length \( u(T) = u_3(T) + u_4(T) \), where \( u_1(T) \) is the number of leaves required to describe the tree \( T \) and \( u_2(T) \) is the number of bits required to describe the data \( (X_1, \cdots, X_{n-1}) \) using the tree as a model for that data. Rissanen tried to approximately minimize \( u(T) \) by repeatedly expanding leaves into new children as long as it caused \( u(T) \) to decrease. Itoh and Kawabata noted that this solution could be improved by growing a large tree and then pruning back to the optimal subtree, using the MINCOST algorithm. As mentioned in Section IV, the MINCOST algorithm can find the subtree minimizing \( u(S) = u_3(S) + \lambda u_4(S) \) in time \( O(N) \), where \( N \) is the number of nodes in the initial tree, when \( \lambda \) is fixed; in this case, \( \lambda = 1 \). When the trade-off rate \( \lambda \) is unknown, the generalized BFOS algorithm can be used to solve the problem for every \( \lambda \) from 0 to \( \infty \), in time \( O(N \log N) \). Elnahas et al. solved the problem directly, without Lagrange multipliers, by minimizing the conditional entropy \( H(X_n|\psi_f) \) ((i) in Table I) subject to a constraint on the number of leaves ((a) in Table I); however, the algorithm used there was an exhaustive search with exponential complexity.

F. Tree-Structured Search

Searching a set of points in a high-dimensional space for a nearest neighbor is a problem in computational geometry familiar in pattern matching and vector quantization. A full search through a set of \( N \) points has a search time proportional to \( N \), or \( \Theta(N) \). It is well known that for \( N \) points in one dimension, a binary search can reduce the search time to \( \Theta(\log N) \). In higher dimensions a similar approach using a tree-structured search can also reduce the search time to \( \Theta(\log N) \) [3], [4]. The generalized BFOS algorithm could be applied to these trees using, for example, \( u_1 \) equal to the number of nodes ((b) of Table I) and \( u_2 \) equal to the expected search time. (The search time would include the time spent traversing the tree as well as any time spent performing a search at the leaves.) This would minimize the search time subject to a constraint on the tree size.

In the aforementioned trees, only one leaf can "occur" at a time since the leaves represent events partitioning a sample space. We now turn to trees in which all leaves "occur" simultaneously.

G. Quadtree Models for Image Coding

Consider an image consisting of 1024 \( \times \) 1024 pixels. The image can be partitioned into four 512 \( \times \) 512 subimages, each of which can be partitioned into four 256 \( \times \) 256 subimages, and so forth, until each subimage consists of a single pixel. Naturally, this recursive partitioning can be represented by a depth-10 quaternary tree, or quadtree. Quadtrees are used in image compression to partition an image into regions of similar characteristics, so that each region can be coded separately by a code optimized to local statistics [18]. This is a form of adaptive compression. The distortion for the entire image is the sum of the distortions from each region, and the code length for the entire image is the sum of the code lengths for each region plus the overhead required to describe the codebook for each region and the tree itself. Since these can be made into monotonic affine tree functionals, the generalized BFOS algorithm can be applied to the full depth-10 quadtree to find the subtree with the optimal distortion-rate trade-off.

H. Quadtree Models for Computer Graphics

Quadtrees are also used in computer graphics to model surface textures, surface normals, and surface deformations. For example, an arbitrarily curved two-dimensional manifold with parameter space \([0,1] \times [0,1]\) can be modeled as a quilt of bilinear patches, where each patch corresponds to a leaf of a quadtree in parameter space [19].
Low patch densities occur around regions of low curvature or other regions where high resolution is not required, such as regions viewed from broadside. High patch densities occur around regions of high curvature or regions viewed in profile. Clearly, better approximations to the surface as well as the rendering time, which is critical in heavy computational tasks such as ray tracing. The generalized BFOS algorithm provides an optimum way of balancing these requirements with, say, \( u_1 \) as the rendering time for the tree and \( u_2 \) as the mean squared error in the image domain.

1. Bit Allocation

It is common in voice and image compression applications to first transform a block of data into the frequency domain and then perform scalar or vector quantization on the frequency components. This leads to the problem of how many bits should be allocated to each frequency band. The generalized BFOS algorithm can be used to solve this problem as follows. Suppose the \( i \)th frequency band has its own sequence of \( n \) codebooks \( C_{i,1}, C_{i,2}, \ldots, C_{i,n} \), where along the sequence the distortion is decreasing and the bit rate is increasing. Construct the following tree: the root node has one child per frequency band and the subtree rooted at each child is the degenerate unary tree of length \( n \) corresponding to the linearly linked list of codebooks for that band. Then, with \( u_1 \) as the sum of bit rates over all the leaves and \( u_2 \) as the sum of distortions, the generalized BFOS algorithm will find the optimum trade-off between the overall rate and distortion and, furthermore, will specify the codebooks to be used at each frequency. In this application, the generalized BFOS algorithm is quite similar to an algorithm proposed recently by Shoham and Gersho [20], which also uses Lagrangian methods to search the convex hull of the achievable distortion-rate region. In addition, since the generalized BFOS algorithm generates a set of good allocations that cover a wide range of total available bits, it provides a solution to the “soft” bit allocation problem of Bruckstein [21], in which the total number of bits available is allowed to vary based on the resulting performance attainable (leading to a variable-rate system). He provides an analytic solution when quantizer error cost and bit usage penalties are monotonic smooth (differentiable) functions; the generalized BFOS algorithm might be applicable when these quantities are discrete or otherwise not well modeled by such functions.

J. Resource Allocation

The generalized BFOS algorithm may also have applications outside engineering and computer science. Consider, for example, an organization which wishes to assign a finite number of salespeople to its regional and subregional offices throughout the world. Opening a regional or subregional sales office requires some fixed overhead regardless of the number of salespeople. Once open, adding salespeople to a subregional office increases revenues but also increases expenses in some known way. A good set of allocations for different sales force sizes can be found by constructing a tree as follows: the root has one child for each regional office, and each regional office has one child for each of its subregional offices. Rooted at each subregional office is the degenerate unary tree whose \( i \)th level gives the variable cost of assigning \( i \) salespeople to that office as well as the revenue they can be expected to generate. Total cost (fixed overhead plus variable cost) and total revenue functionals for the tree can easily be generated recursively from this information, and one can then prune the tree to find how many salespeople to assign to each office, or even whether one should close a subregional or regional office altogether.

VII. TREE-STRUCTURED VECTOR QUANTIZATION

Vector quantization (VQ) is a data compression technique which approximates a block or vector of source symbols by its closest (minimum distortion) representative from a finite number of vectors (codewords) stored in a codebook. The index of the closest codeword is determined during encoding, transmitted or stored, and used to look up the codeword at the decoder. Reviews of vector quantization may be found in [22]-[24].

A popular codebook design algorithm for VQ, the generalized Lloyd algorithm [25], [26], produces a locally optimal codebook from a training sequence typical of the source to be coded. The codebook is unstructured; determination of the closest codeword therefore requires that the distortion between the source vector and each codeword be computed. Since the number of codewords, and hence the computational load of this “full search,” increases exponentially with rate and vector dimension, much effort has been spent in reducing the search complexity. One approach is to retain the optimal codebook (as designed by the generalized Lloyd algorithm) and build a data structure that allows rapid search. Application of optimal pruning to such systems is mentioned in Section VI under “tree-structured search” but will not be pursued here. The alternative approach is to require that the codebook itself be structured in a way that facilitates the search. Such structure generally produces systems whose performance is worse than that of full-search VQ; the hope is that the reduced complexity will more than offset the performance loss.

In tree-structured VQ (TSVQ), a tree is constructed with a codeword at each node. TSVQ was introduced in [2] and has been studied or implemented as part of a larger compression scheme in many different systems [27]-[33]. For simplicity, we restrict ourselves to binary trees, although nonbinary TSVQs have been found advantageous in some applications [28], [29], [33]. The structure of a binary TSVQ is shown in Fig. 5. During encoding, the distortion between a source vector and both children of the root is calculated; if the left one is “closer,” a ‘0’ is
dispatched to the channel and we descend to the left child node. If the right one is closer, we dispatch a ‘1’ and descend to the right child node. The process repeats at each node until a terminal node is reached, at which time we return to the root node and get another source vector. The decoder uses the transmitted path map and its own copy of the tree structure to look up the desired terminal node and then dispatches the vector stored there to its output. Hardware architectures for TSVQ have been proposed in [34]–[36].

![Figure 5. Structure of binary TSVQ.](image)

The computational load of TSVQ is $\mathcal{O}(\log N)$ rather than $\mathcal{O}(N)$ for full-search VQ, where $N$ is the number of codewords on the terminal layer. Memory requirements (to store the codebook) are nearly double those of full-search VQ. Our experience in this and other studies [2], [27], [28] suggests that the performance penalty over full-search is quite small (less than 1 dB), although it may be necessary to use nonbinary trees.

Design of the quantizer proceeds by applying the generalized Lloyd algorithm to the codebook formed by the children of the root node. The training sequence is then split into those vectors that are closer to the left child and those that are closer to the right. The “left” portion is then used to design the codebook formed by the children of the left child; the “right” portion is used to design that of right child. The process repeats until the terminal layer is reached. Such a design algorithm is “greedy” since a layer’s codewords are frozen while subsequent layers are designed.

A pruned TSVQ occurs when we remove some of the branches of the complete TSVQ that is typically designed by the algorithm described in the preceding. The application of pruned TSVQ and its design by the generalized BFOS algorithm are described in the next section.

VIII. PRUNED TSVQ

We examine three source-coding systems based on pruned TSVQ. The first and most deeply treated is the natural variable-rate vector quantizer that arises when we prune a complete TSVQ but retain the tree-search path map as the string that is sent to the channel. Since the length of this string will vary depending on the depth of the terminal node that is reached, the instantaneous rate of the TSVQ will vary. The average rate is the sum over the terminal nodes of the probability of reaching a given terminal node times its depth. A second system arises if we use a pruned TSVQ structure; but rather than sending the tree-search path map, we number the terminal nodes in binary and simply send a terminal node’s number. Such a system has a fixed rate. Finally, we can use the node’s number as the input to a noiseless entropy coder (such as a Huffman or arithmetic coder). We call these three systems variable-rate pruned TSVQ, fixed-rate pruned TSVQ, and entropy pruned TSVQ, respectively.

In our experiments with pruned TSVQ, we used a locally generated 2-min training sequence of 8-kHz sampled speech from three males and three females. The test sequence was 40 s of speech from a male and female not represented in the training sequence. Our distortion measure was mean squared error. All vectors had eight dimensions, so that the training sequence was composed of 120000 vectors. The starting (complete) TSVQ had depth 12 (that is, rate 1.5 bits/sample). We emphasize that we chose this experiment as a good example of a reasonably complex source that could demonstrate the effects of pruning. However, direct waveform vector quantization of speech (especially at these rates) has had limited success, and we expect that more profitable use of pruned TSVQ will be found in other VQ applications such as image, transform, and sub-band coding. Preliminary results suggest that the technique cannot be used to full advantage on LPC–VQ, however.

Fig. 6 demonstrates test sequence performance of variable-rate pruned TSVQ compared with full-search VQ and fixed-rate complete tree-structured VQs designed by the generalized Lloyd algorithm. To generate the pruned trees, we applied the algorithm of Section IV with the pruning criteria given by $u_1 = \text{average length} ((e) \text{ of Table I})$ and $u_2 = \text{average distortion} ((i) \text{ of Table I})$. Note that, providing the additional complexity and delay inherent in the buffering of a variable-rate system is acceptable, our pruned TSVQs outperform both alternative systems for a significant range of average rates. In addition, at most rates our systems require significantly fewer computational resources than the same rate full-search VQ, since the number of distortion calculations is bounded by twice the depth of the starting tree for the pruned TSVQ. Memory requirements can be significantly greater, however, although they are also (obviously) bounded by those of the starting tree.

In Fig. 7 we compare fixed-rate pruned TSVQ with fixed-rate full-search VQ and complete TSVQ. Here we kept $u_2$ unchanged, but $u_1$ was given by the number of leaves ((a) of Table I). As noted in Section II, any subtree which lies on the operational optimal $u_1 - u_2$ tradeoff function remains there after taking the logarithm of either axis, so that the $u_1$ criterion is equivalent to the log of the number of leaves (i.e., the rate). Since the standard generalized Lloyd algorithm attempts to determine precisely the best VQ for a fixed number of codewords, it is not surprising that under this criterion we cannot outperform the full-search VQ. On the other hand, complete subtrees of the starting tree are a subset of the pruned subtrees; we therefore expect to outperform the complete TSVQs. This is in fact what we observe.

An interesting alternative variable-rate VQ arises when a standard full-search VQ is followed by a variable-rate
Fig. 6. Performance of variable-rate pruned TSVQ. Circles and solid line are pruned systems; crosses and dashed line are full-search VQ; x’s and dotted line are complete TSVQ.

Fig. 7. Performance of fixed-rate pruned TSVQ. Circles and solid line are pruned systems; crosses and dashed line are full-search VQ; x’s and dotted line are complete TSVQ.

entropy coder. If we assume that the average rate of the entropy coder is as low as the entropy of the VQ’s output symbols (this may require entropy coding of blocks of independent VQ output symbols), we can compare (Fig. 8) this system’s average rate with our variable-rate pruned TSVQ. Performance is comparable for several rates. Again note that our system is much simpler since it has a single stage and uses a tree search. Of course, we could not hope to outperform a full-search VQ designed to minimize entropy for a given distortion, i.e., an entropy-constrained VQ [38]. However, even if such a system were designed, we would still have the advantage of simplicity. Finally, if we allow two-stage systems with entropy coding, it is interesting to compare entropy pruned TSVQ against full-search VQ and complete TSVQ when each is followed by entropy coding. In this case (Fig. 9), we let $u_i$ be the entropy of the leaves ((f) of Table I). Note that, given an efficient entropy coder (one whose average rate is very close to the TSVQ’s entropy), this system has the potential of outperforming any system considered in this paper; however, it loses the advantages of simplicity and robustness against the buffer overflow inherent in variable-rate pruned TSVQ.

To demonstrate the behavior of pruned TSVQ, we return to variable-rate pruned TSVQ and compare (Fig. 10)
the operation of our variable-rate coder with average rate 0.74 bits/sample to that of a full-search VQ with rate 0.75 bits/sample. We note that the pruned TSVQ throws away information about the low-energy portions of the waveform and expends effort on reproducing high-energy portions (which contribute most to the distortion). In the process, the pruned TSVQ achieves a smaller average distortion (by 1.8 dB) than the full-search VQ. Unfortunately, from the point of view of waveform speech coding, the result is to throw out fricatives and other low-energy sounds along with the silence. This, however, is a drawback of the mean square error distortion measure, not the pruned system. Other distortion measures may be more suitable for direct coding of the speech waveform.

It is interesting to evaluate pruned TSVQ on a source to which we can apply distortion-rate theory. However, for simple sources, such as the first-order Gauss-Markov source, we found experimentally that full-search VQ, complete TSVQ, and pruned TSVQ all have very similar performances (the closeness of full-search VQ and complete TSVQ for this source was noted in [28]). A source that is complex enough to demonstrate a significant discrepancy
between system performances, as waveform coding of speech does, is the switched source that, at the beginning of each switching period, chooses randomly between two memoryless Gaussian sources with different powers. As the switching period gets long, the distortion-rate function of this source approaches the average of the distortion-rate functions of its subsources [37]. We plot the performance of full-search VQ, complete TSVQ, and pruned TSVQ, as well as this asymptotically tight bound, on the distortion-rate curve in Fig. 11 for a switched source with long switching period and subsource variances 1 and 1000.

A final experiment concerned the robustness of variable-rate pruned TSVQ in the face of buffer overflow. First, note that variable-rate pruned TSVQ is an instantaneous code: the decoder can identify the end of a transmitted variable-length codeword without having to examine symbols beyond the codeword's end. Combining this with knowledge of the encoder buffer's starting state allows the decoder to track the encoder buffer's state. If the decoder knows there are only \( m \) bits left in the encoder buffer, and a word is received which is not completely decoded after \( m \) bits, then buffer overflow occurred. Bit stuffing during...
underflows can also be detected without side information. Although any instantaneous code can detect overflows, most such codes do little once an overflow has occurred, other than remain synchronized. Typically, a string of zeros, the last successfully decoded codeword, or some other ad hoc scheme is used to generate decoder output during buffer overflow. Variable-rate pruned TSVQ, on the other hand, can use internal nodes to provide good approximations to the desired reproduction: the more bits that are available in the buffer, the better the reproduction. Fig. 12 demonstrates SQNR performance versus buffer length for a 3.998 bits/vector eight-dimensional variable-rate pruned TSVQ, pruned from depth eight, designed using the six-speaker speech sequence and tested on the two-speaker sequence over a fixed rate 4 bits/vector channel. The graceful degradation strategy is significantly more robust than either generating zeros or repeating the last codeword. Although for buffer lengths less than 32 bits the performance of variable-rate pruned TSVQ drops below that of complete TSVQ with rate fixed at the channel rate, this too can be avoided by adopting a similar buffer underflow strategy. During underflows the search is continued beyond the variable-rate pruned TSVQ into the original TSVQ so that all transmitted bits carry information. Entropy-pruned TSVQ, which includes a stage of entropy coding, does not share the graceful degradation property of variable-rate pruned TSVQ.

IX. IMPLEMENTATION ISSUES

The software implementation of our pruned TSVQ designer had three modules. A complete TSVQ designer produced the starting tree. A pruning initializer used a training sequence to calculate $\Delta u(R)$ for the height-1 subtrees rooted at each internal node of the complete tree and $u(\text{root}(T))$ from the appropriate definitions in Table I. The pruner used this information to recursively initialize the $\Delta u(S_i)$ for each internal node using (4) (this is a more expedient way of calculating $\Delta u(S_i)$ rather than directly via $\Delta u(S_i) = u(S_i) - u(t)$) and then applied the remainder of the pruning algorithm as detailed in Section IV. This approach provides an efficient implementation of the initialization and divides the algorithm into a tree-functional dependent module (the initializer) and a tree-functional independent module (the pruner). By far the most computational effort was expended in the TSVQ design stage: a typical ratio in design time would be 100 to 5 to 1 for the three modules.

Since the TSVQ design is so expensive, we investigated ways of reducing the design time without reducing performance of the pruned subtrees. We used two approaches suggested by Breiman et al. Both are characterized by "stopping rules," which cause the TSVQ designer to stop splitting a certain node when a criterion is satisfied. If enough nodes are stopped, the design time will be reduced since portions of the training sequence mapping into these nodes no longer need be used as the design continues.

One stopping rule stops splitting when the number of training vectors mapping into a node falls below a minimum number. This is not particularly effective since it is not related to the pruning criterion and results in small computational savings (the number of training sequence vectors we avoid encoding is no greater than the minimum number per node times the number of stopped nodes). An alternative stopping rule is based on Theorems 10.31 and 10.32 of Breiman et al. (5, p. 292). Applying their argument to the case of variable-rate pruned TSVQ, it is easy
to see that if we fix $\alpha_0$, any node whose conditional expected distortion is less than $\alpha_0$ will be pruned off before any optimal pruned subtree with $\lambda > \alpha_0$ is reached by the pruning algorithm. Hence, with this stopping rule we may reduce computation while still guaranteeing that some pruned subtrees of our new smaller starting tree are identical to those that would be produced from a larger complete tree. Unfortunately, the bound here is very loose, and in order to reduce design time, $\alpha_0$ must be set so high that only very few (and probably uninteresting) subtrees are guaranteed to be optimal with respect to larger starting trees. Nevertheless, an evaluation of all the subtrees of our reduced-size starting tree demonstrated only a slight degradation in performance (typically 0.1 dB) over their entire range of rates for our waveform speech coding experiments. Since larger starting trees yield better performance, we can use computational savings due to the stopping rule to grow starting trees with larger maximum depth (given limited computer time); if the increase in performance due to the larger tree outweighs the penalty for nonoptimality of the subtree (as it does in the experiments done for this paper), we have a net improvement in performance.

Another implementation detail concerns training sequence use and memory storage for the tree. Training sequence size is limited both by the availability of data and, more importantly, by the effect on design time. However, when large trees are grown using a limited training sequence, the number of training sequence vectors per leaf gets quite small. This leads to some statistical unreliability in the tree: certain high-distortion low-probability events may occur in the training sequence that are not truly representative of the source. The pruning algorithm will maintain a path for these events since the path is infrequently used (adds little to average rate) and results in a significant decrease in distortion, as estimated by the training sequence, when it is used. When run on test data, these paths may essentially never be used, with the consequence that we have some wasted memory in the tree structure. If we initialize the pruning algorithm using a second, different training sequence, many of these "outlier" cells will get no training sequence vector populations and will be pruned off in the first step of the pruning algorithm. In our experiments, this approach typically cost 0.2–0.3 dB in performance but cut memory by 30–40 percent.

X. CONCLUSIONS

The algorithm of Breiman et al. finds a nested sequence of pruned subtrees of a given tree by optimally trading decreases in distortion for decreases in the number of leaves. We have generalized their algorithm to balance any two monotonic affine tree functionals, and have interpreted this trade-off in the distortion-rate plane. We have also listed a number of practical monotonic affine tree functionals and outlined a variety of current applications in which these might be used, including tree-searched vector quantization, vector quantization of noisy sources, minimum expected cost decision trees, optimum bit allocation, fast pattern matching, and computer graphics and image processing using quadtrees. The algorithm is not limited to binary trees; general trees are permitted (although they must be finite).

We have introduced three variants of pruned TSVQ: variable-rate, fixed-rate, and entropy-pruned TSVQ. For 8-kHz sampled speech at blocklength 8, variable-rate pruned TSVQ outperforms full-search VQ at 0.75 bits/sample by 1.8 dB in the test sequence. Even when the full-search VQ is followed by entropy coding, variable-rate pruned TSVQ is comparable in performance with greatly reduced complexity. Entropy-pruned TSVQ followed by entropy coding greatly outperforms full-search VQ followed by entropy coding although, presumably, it would not outperform a full-search VQ designed with an entropy constraint rather than a number of codewords constraint. Fixed-rate pruned TSVQ has a performance nearly identical to the complete TSVQ. On switched Gaussian sources, variable-length pruned TSVQ closes the gap between the distortion-rate bound and the blocklength eight full-search VQ by a factor of two.

ACKNOWLEDGMENT

The authors wish to thank Shuichi Itoh for his helpful comments during the preparation of this paper.

APPENDIX

This section provides the lemmas which are used in Section IV to justify the generalized BFOS algorithm. The proofs are largely geometric, in contrast to the proofs of Breiman et al., which are algebraic. We feel that geometric proofs are more natural and intuitively appealing.

In what follows, $W(S) = \{u_i(S), u_j(S)\}$, where $u_i(S)$ and $u_j(S)$ are, respectively, monotonically increasing and decreasing affine tree functionals.

Lemma 1: Let $S, S' \subseteq T$ be pruned subtrees of $T$ and let $F$ be a face of the convex hull of $\{u(R) | R \subseteq T\}$. If $u(S), u(S') \in F$, then $u(S \cap S'), u(S \cup S') \in F$, where $S \cap S'$ is the pruned subtree of $T$ containing all nodes common to both $S$ and $S'$, and $S \cup S'$ is the pruned subtree of $T$ containing all nodes that lie in $S, S'$, or both.

Proof: If either $S$ or $S'$ is the root, we are done. Otherwise, let $R = S \cap S'$ and $R' = S \cup S'$, as illustrated in Fig. 13. Then by
In Fig. 14, for example, \( \Delta = \Delta u(R_1) + \Delta u(R_2) + \Delta u(R_3) \). Fig. 15 shows these vectors in the plane. It is easy to see that if \( R \) did not lie on the face, then \( R' \) would be outside the convex hull, and vice versa. Hence they must both lie on the face of the convex hull.

**Corollary 1:** Let \( F \) be a face of the convex hull with lower right endpoint \( u_2 \) and upper left endpoint \( u_1 \). Then there exist \( S_0, S_1 \subseteq T \) such that \( u(S_0) = u_0, u(S_1) = u_1 \), and \( S_1 \subseteq S_0 \).

**Proof:** Let \( \mathcal{F}_r = \{ S | S \subseteq T, u(S) \in F \} \) be the set of all pruned subtrees lying on face \( F \) of the convex hull. Repeated application of Lemma 1 yields \( u(\cap \mathcal{F}_r), u(\cup \mathcal{F}_r) \in F \), that is, \( \cap \mathcal{F}_r, \cup \mathcal{F}_r \subseteq \mathcal{F}_r \). Thus, clearly, \( \cap \mathcal{F}_r \subseteq S \subseteq \cup \mathcal{F}_r \); for all \( S \subseteq \mathcal{F}_r \). Hence, by monotonicity, \( u_1 = u(\cap \mathcal{F}_r) \) and \( u_0 = u(\cup \mathcal{F}_r) \).

**Lemma 2:** Let \( S' \subseteq S \subseteq T \) and let \( F \) be a face of the convex hull. If \( u(S), u(S') \in F \), then \( u(S' \cup S), u((S - S') \cup \{ i \}) \in F \), \( \forall i \in S' \). That is, a subtree obtained by adding a branch \( S' \) to \( S \) or pruning a branch \( S' \) from \( S \), where \( i \in S' \), lies in \( \mathcal{F}_r = \{ S | S \subseteq T, u(S) \in F \} \).

**Proof:** Let \( R = S' \cup S \) and let \( R' = (S - S') \cup \{ i \} \), as illustrated in Fig. 16. Then \( S' = R \cap R' \) and \( S = R \cup R' \); so by (3),

\[
\begin{align*}
\quad u(R) &= u(S') + [u(R_i) - u(i)] = u(S') + \Delta \\
\quad u(S) &= u(R') + [u(S_i) - u(i)] = u(R') + \Delta 
\end{align*}
\]

If \( R \) did not lie on the face, then \( R' \) would be outside the convex hull, and vice versa, as illustrated in Fig. 17. Hence they must both lie on the face of the convex hull.

**REFERENCES**


