Title

Unbalanced non-binary tree-structured vector quantizers

Authors

Schmidl, T M
Cosman, P C
Gray, R M

Publication Date

2014-08-06

Peer reviewed

Unbalanced non-binary tree-structured vector quantizers

Timothy M. Schmidl Pamela C. Cosman Robert M. Gray

Information Systems Laboratory, Stanford University, Stanford, CA 94305-4055

Abstract

An established method for developing unbalanced binary tree-structured vector quantizers is greedy growing followed by optimal pruning. These algorithms can be extended to a hybrid binary/quaternary tree structure or to a pure quaternary tree structure. The tradeoff of decreased distortion for increased rate is examined for the split into two or four children at each terminal node. The trees employing quaternary splits have smaller memory requirements for the codebook and provide slightly lower mean-squared-error on the test sequence as compared to a binary tree.

1 Growing and pruning binary trees

Tree-structured vector quantization (TSVQ) is an image compression technique that is rapid for both the encoder and the decoder. A variable rate code can be implemented by an unbalanced tree, obtained either by growing a balanced tree and then pruning it back so that it becomes unbalanced, or by "greedily" growing an unbalanced tree directly [5]. A balanced TSVQ is grown one level at a time using, for example, the splitting method of the generalized Lloyd algorithm (GLA) [3]. This results in a fixed rate code. A large fixed rate tree can be pruned back to form a variable rate tree, but growing a fixed rate tree is unnecessarily constrained if the final code is to be variable rate. Instead, one can grow the tree one node at a time. The algorithm is an extension of a common decision tree design technique [1] to VQ. An "impurity function" which measures the quality or penalty of a particular node in a tree is chosen. For VQ, the impurity function is taken to be an average distortion,

$$d(j) = E[d|j] \equiv E[d(\mathbf{x}, \hat{\mathbf{x}}_j)|j], \tag{1}$$

where $d(.,.)$ is a distortion measure and $\hat{\mathbf{x}}_j$ is the reproduction vector associated with node j. Typically, the distortion measure used is the mean-squared error.

The conditional expectation is computed as a sample average based on a training set, e.g., if $\mathcal{T}_j$ is the set of all training vectors $\mathbf{x}_k$ mapping into node j, then

$$d(j) = \frac{1}{\|\mathcal{T}_j\|} \sum_{k:x_k \in \mathcal{T}_j} d(\mathbf{x}_k, \hat{\mathbf{x}}_j) \tag{2}$$

where $\|\mathcal{T}_j\|$ is the number of vectors in $\mathcal{T}_j$. The "goodness" of a node split is defined as the decrease in node impurity:

$$\Delta d(s, j) = d(j) - p_L d(j_L) - p_R d(j_R). \tag{3}$$

Here, s is a binary test (a nearest neighbor selection in a TSVQ), $d(j)$ is the impurity (distortion) measured at node j of the tree, p_L is the proportion of the samples in node j that go to the left child, and p_R is the proportion that go to the right child. If $p(j), p(j_L)$, and $p(j_R)$ are the probabilities (as estimated by relative frequency on the training sequence) of nodes j, j_L, and j_R respectively, then $p_L = \frac{p(j_L)}{p(j)}$ and $p_R = \frac{p(j_R)}{p(j)}$.

Given a tree T, let $\tilde{T}$ stand for its leaves (or terminal nodes). Assume that we split $j \in \tilde{T}$ into two new leaves j_L and j_R. Let D and R stand for the distortion and rate, respectively, measured by T, and let D' and R' stand for the distortion and rate of the tree after j is split. Let $\Delta D = D' - D$ and $\Delta R = R' - R$ be the change in the distortion and rate, respectively, due to splitting j, and let $l(j)$ be the depth of node j. Then

$$D = p(j)d(j) + \sum_{\substack{i \in \tilde{T} \\ i \neq j}} p(i)d(i) \tag{4}$$

$$R = p(j)l(j) + \sum_{\substack{i \in \tilde{T} \\ i \neq j}} p(i)l(i) \tag{5}$$

$$D' = p(j_L)d(j_L) + p(j_R)d(j_R) + \sum_{\substack{i \in \tilde{T} \\ i \neq j}} p(i)d(i) \tag{6}$$

1058-6393/93 $03.00 © 1993 IEEE

$$R' = p(j_L)l(j_L) + p(j_R)l(j_R) + \sum_{\substack{i \in T \\ i \neq j}} p(i)l(i), \qquad (7)$$

and the ratio of the change in distortion to the change in rate due to splitting leaf j is

$$\lambda = -\frac{\Delta D}{\Delta R} = d(j) - p_L d(j_L) - p_R d(j_R) = \Delta d(s, j), \qquad (8)$$

which is the goodness of split for leaf j.

As in decision tree design, we can design a TSVQ one node at a time, always splitting the node with the largest λ. We take the binary test s to be a nearest neighbor selection of node labels designed by the GLA, although other tests could be used. The algorithm is "greedy" in the sense that each node is split without considering its later effect on the tree. This method results in an unbalanced tree, since the node that is split can be at any depth. There will be more codewords available to code high distortion events; this is where the tree will have been split the most.

The growing method optimally trades off rate and distortion for each new node in a greedy fashion. The resulting tree can then be pruned with the generalized Breiman, Friedman, Olshen, and Stone (BFOS) algorithm [1, 2] an extension of an idea from classification tree design to coding. One can achieve a lower distortion for a given average rate by optimally pruning the tree with the generalized BFOS algorithm rather than by removing the nodes in the reverse order in which they were added. This is because the growing algorithm is greedy, whereas the BFOS pruning algorithm removes nodes optimally. Unbalanced trees are also able to code high distortion events at a higher resolution than can balanced trees which are limited by their initial depth.

2 Extension to non-binary trees

These growing and pruning algorithms can be directly extended to m-ary trees, such as quaternary, or to hybrid trees, such as a mixed binary/quaternary structure. The idea of a hybrid tree is to allow the greedy algorithm to choose the type of split that provides the best trade-off of rate and distortion at each node. For each terminal node in the tree, we first run the GLA to determine how that node would split into 2 children. We can calculate

$$\lambda_2 = -\frac{(\Delta D)_2}{(\Delta R)_2} = \frac{(\Delta D)_2}{1} \qquad (9)$$

for that split. The GLA can be run again to determine the split into 4 children, and we calculate

$$\lambda_4 = -\frac{(\Delta D)_4}{(\Delta R)_4} = \frac{(\Delta D)_4}{2} \qquad (10)$$

for the quaternary split. For a hybrid bi/quat tree, the larger of the two ratios provides the candidate split for that node; it is compared against the candidate splits for all the other current terminal nodes in order to choose the next node for splitting. The extension of the pruning algorithm is analogous. For the pure quaternary tree, the candidate split for a node is simply taken to be λ_4. Because the GLA must be run twice for each split, this algorithm for a hybrid tree requires more time to develop a tree than does the pure binary or pure quaternary algorithm. However, this is not considered to be a problem since the codebook development is performed only once, off-line.

Binary, quaternary, and hybrid binary/quaternary trees were grown to a depth of 0.75 bpp on a training sequence of five 512 × 512 images from the USC database blocked into 4 × 4 vectors. The trees were pruned back to several different bit rates between 0 and 0.75 bpp. On each pruned subtree, the distortion was measured on two test images ("Lena" and "LAX") not in the training set. The results are shown in Figures 1–2. In these figures, the results for the binary tree are shown as a solid line, those for the hybrid tree are shown as a dashed line, and those for the quaternary tree are shown as dash-dot. At the low bit rates, the quaternary performs best, and the binary is identical to the hybrid because the hybrid tree does not choose to make any quaternary splits. At intermediate rates, the hybrid and quaternary trees outperform the binary tree by up to 10% on the test images. We also examined octonary trees and hybrid bi/quat/oct trees. The trees involving octonary splits were not significantly different from those involving quaternary splits, and the results are not shown.

The quaternary splits require more time for encoding than the binary splits. If implemented with distortion calculations, they are equivalent: to descend 2 levels in a binary tree, one must make a total of 4 distortion calculations (2 at the first level, and 2 at the next level). To descend one level in a quaternary tree, one makes 4 distortion calculations. Thus in either case, there is the same cost of 2 distortion calculations per 1 bit output. However, encoding for the binary split can be implemented as a hyperplane test, which would require only half the time, since the dot product calculation is approximately as time-consuming as one distortion computation. Although

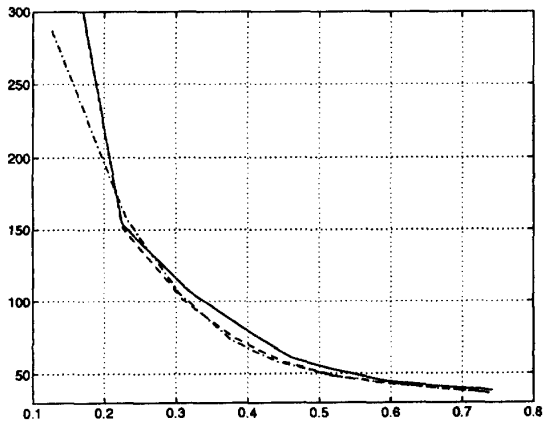

Figure 1: Distortion (MSE) vs. bit rate (bpp) on "Lena" test image

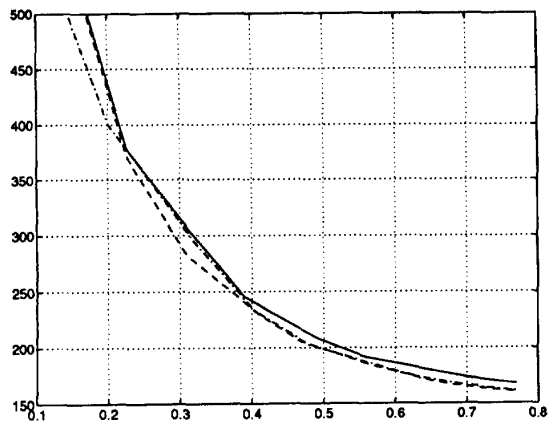

Figure 2: Distortion (MSE) vs. bit rate (bpp) on "LAX" test image

with distortion computations, the binary and quaternary splits are equivalent, this is no longer the case with an octonary tree, where one makes 8 calculations in order to put out 3 bits, for a cost of 8/3 calculations per bit. With a full search VQ, the cost per bit depends on the codebook size, but one can, for example, make 256 distortion calculations in order to put out 8 bits, for a complexity of 32 calculations per bit output. So the encoding complexity increases along the spectrum of m-ary trees as they approach full search.

An additional advantage comes from the memory savings. Tree-structured codebooks occupy considerably more memory space than do full search codebooks because all the internal nodes must be stored as well as the terminal nodes. In this respect, quaternary splits are superior to binary ones, because a 2-level balanced binary tree requires storage of the two internal nodes in addition to the root node and the 4 terminal nodes, whereas a quaternary tree of the same rate has only the root node and the 4 terminal nodes. Figure 3 plots the total number of nodes (terminal + internal) in a tree as a function of bit rate (binary: solid line, hybrid: dashed line, quaternary: dash-dot line). The binary tree has about 30% more nodes over a wide range of rates. If the binary search is implemented as a hyperplane test, then the actual vectors corresponding to the nodes do not need to be stored at the encoder. However, the vectors would need to be stored at the decoder, and if the system is to be used progressively, then the vectors corresponding to the internal nodes would need to be stored as well as those for the terminal nodes.

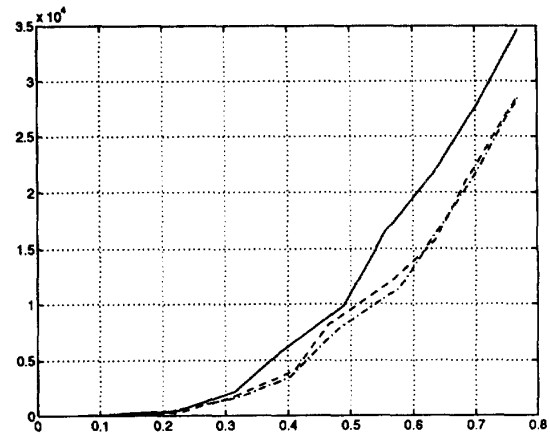

Figure 3: Total number of tree nodes vs. bit rate (bpp).

3 Initialization of quaternary splits

In the original binary greedy growing algorithm, the Lloyd algorithm for each split was initialized by setting one child's codeword to be equal to the parent centroid $\vec{v}$, and choosing the other child's codeword to be $(1+\epsilon)\vec{v}$, where ϵ is some small number. The Lloyd algorithm is then run to improve these two codewords. By setting one child's codeword to be equal to the parent's centroid, one ensures that the children will provide at least no worse a representation for the training vectors than the parent node did. The choice of the other child's codeword as a slight perturbation of $\vec{v}$ in the direction of $\vec{v}$ was arbitrary.

Other researchers tried splitting in the principal axis direction [4, 7], and it was found that initializing the Lloyd algorithm with a perturbation in the principal axis direction produced better splits than a random perturbation. Because neighboring pixels in a block are highly correlated, the data set of training vectors is strongly "cigar-shaped" along the $(1, 1, \ldots, 1)$ direction for most of the early nodes of the tree, and principal axis splitting capitalizes on that correlation. In work that attempted to jointly optimize many splits of the tree at once, it was noted that the principal axes of the split subsets remain approximately the same for about 30 splits, for many different types of images [6].

Thus it turned out that the original arbitrary choice of splitting in the direction of $\vec{v}$ was a good one. For the early nodes of the tree, the centroids lie nearly along the $(1, 1, \ldots, 1)$ direction, and so do the principal axes of the clusters. So perturbing in the direction of $\vec{v}$ is nearly identical to perturbing in the principal axis direction. Towards the bottom of a large tree, the clusters are more likely to be spherical rather than strongly cigar-shaped, so the perturbation in the $\vec{v}$ direction would tend not to be particularly worse or better than any other.

How does this discussion extend to the quaternary case, in which one needs to choose three perturbation directions? We tried a method based on principal axes, as well as several *ad hoc* methods which attempt to "divide up" the $(1, 1, \ldots, 1)$ or the $\vec{v}$ direction between them. In all cases, one of the four children nodes was initialized to equal the centroid $\vec{v}$ of the parent node. The other 3 children were initialized to $\epsilon_1 + \vec{v}$, $\epsilon_2 + \vec{v}$, and $\epsilon_3 + \vec{v}$, where ϵ_1, ϵ_2, and ϵ_3 are small vectors whose directions are given in Table 1 for the 6 different methods tried. As shown in Figure 4, when a pure quaternary tree is grown using these different initialization methods, the difference in performance is very slight. The last method listed, better than the others by a small amount for this test image, is shown as a dashed line.

ϵ_1	ϵ_2	ϵ_3
(1...1)	(1 -1 ...)	(-1 1 ...)
a_1	a_2	a_3
(111110...0)$\vec{v}$	(0...0111110...0)$\vec{v}$	(0...0111111)$\vec{v}$
(1...10)$\vec{v}$	(1...101)$\vec{v}$	(1...1011)$\vec{v}$
(1...)$\vec{v}$	(-1...-1)$\vec{v}$	(1 -1 1 -1...)$\vec{v}$
(1...1)$\vec{v}$	(1 -1 ...)$\vec{v}$	(-1 1 ...)$\vec{v}$

Table 1: Initialization methods: a_1, a_2, a_3 denote the first three principal axis directions.

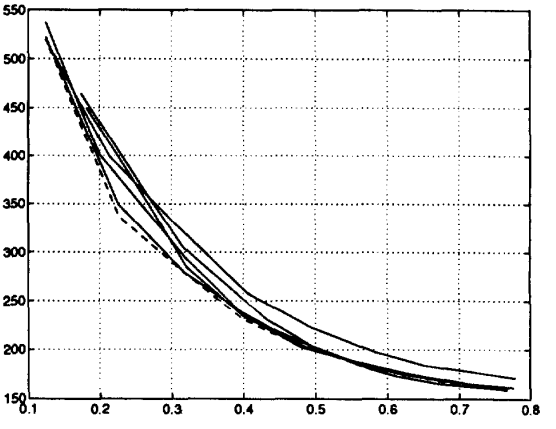

Figure 4: Distortion (MSE) vs. bit rate (bpp) on "LAX" test image: comparison of initialization methods for quaternary splits

4 Using lookahead

For both the USC training set discussed above and a training set of magnetic resonance brain scans, the hybrid TSVQ made its first few splits binary. Quaternary splits started occasionally winning out over binary ones only after the first few tree levels. One can speculate that binary splits are chosen over quaternary ones at the low levels of the tree because of this tendency of the subsets to be strongly "cigar-shaped" along the principal axis direction for the early splits. However, it is known that a full search VQ tends to outperform a balanced TSVQ of the same rate. Therefore, a quaternary split of the root node will often be an improvement over 2 levels of binary splits of the root node. One could know if it is an improvement by using lookahead to determine the future binary

splits of the initial binary split, and then comparing the 2-level tradeoff of distortion for rate against the quaternary case. The difficulty of imposing such a seemingly superior quaternary split on the tree is that the binary greedy growing algorithm, after splitting the root node and the preferred one of its two children, may decide *never* to return to the other child for splitting. If the other child has a sufficiently low distortion, then the $\frac{(\Delta D)_2}{(\Delta R)_2}$ tradeoff might never be good enough to cause that node to be chosen. In that case, having started the tree with a quaternary split will be inferior to the binary choice. The comments here about the root node apply to all subsequent nodes as well. One way of guaranteeing that at each node j the tree always has the better of the two choices (quaternary split or 2-level balanced binary split) would be to run the greedy growing algorithm as discussed above, choosing the better of the quaternary or 1-level binary split, and then, *if* the growing algorithm at some later point chooses to come back and split the other child of node j, then the 2-level balanced binary tree descending from node j could be compared against the quaternary split of that node and replaced if necessary. If the quaternary split did replace it, all the splitting that occurred between the first and second visitations of node j's children would have to be redone. As it was felt this backtracking approach would be too time-intensive, we implemented a simple lookahead step instead, in which the quaternary split was chosen over the 2-level balanced binary in those cases where it provided a superior trade-off. The resulting tree performed slightly better than the hybrid trees grown without lookahead.

These remarks apply even more strongly to the bi/quat/oct hybrid tree structure. An octonary split is less likely to be chosen than a binary split, and it should be compared against 3 levels of binary splits. There is a greater need for backtracking or lookahead, and we have not yet implemented these, nor have we varied the octonary initialization. This perhaps explains the preliminary results that a pure octonary tree and the bi/quat/oct hybrid tree do not outperform the trees involving only binary and quaternary splits.

5 Conclusion

Greedy growing and optimal pruning algorithms for binary tree structures can be directly extended to produce m-ary trees and hybrid trees that mix different types of splits. The quaternary and hybrid binary/quaternary trees provide a slight decrease in distortion and a significant decrease in codebook storage requirements, at the expense of an increase in encoding time. Additional refinements to this algorithm, including substituting quaternary splits for 2-level balanced binary splits when appropriate and examining different choices for the initialization of the quaternary Lloyd splits provided small additional improvements. By examining binary, quaternary and octonary trees and hybrid mixtures of them, this study is a preliminary attempt to understand the tradeoffs in performance, storage requirements, and encoding time that lie along the spectrum of unbalanced tree structures from binary TSVQ to full search VQ.

6 Acknowledgments

This work was supported in part by the National Institutes of Health under Grant CA49697-02, by the National Science Foundation under Grant MIP-9016974, and by a Graduate Fellowship from the National Science Foundation.

References

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[2] P. A. Chou, T. Lookabaugh, and R. M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Trans. Inform. Theory*, 35(2):299–315, March 1989.

[3] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Comm.*, COM-28:84–95, Jan. 1980.

[4] M.T. Orchard and C.A. Bouman. Color quantization of images. *IEEE Trans. Signal Process.*, 39:2677–2690, Dec. 1991.

[5] E. A. Riskin and R. M. Gray. A greedy tree growing algorithm for the design of variable rate vector quantizers. *IEEE Trans. Signal Process.*, 39:2500–2507, Nov. 1991.

[6] X. Wu. Vector quantizer design by constrained global optimization. In J. A. Storer and M. Cohn, editors, *Proceedings Data Compression Conference*, pages 132–141, Snowbird, Utah, March 1992. IEEE Computer Society Press.

[7] X. Wu and K. Zhang. A better tree-structured vector quantizer. In J. A. Storer and J. H. Reif, editors, *Proceedings Data Compression Conference*, pages 392–401, Snowbird, Utah, April 1991. IEEE Computer Society Press.