[19] 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 Proc.*, vol. 1, pp. 162–169, Apr. 1992.
[20] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering Optimization*. New York: Wiley, 1983.
[21] K. Rose, D. Miller, and A. Gersho, "Entropy-constrained tree-structured vector quantizer design by the minimum cross entropy principle," in *Proc. Data Compress. Conf.*, Snowbird, UT, Mar. 1994.
[22] K. Rose, E. Gurewitz, and G. C. Fox, "Statistical mechanics and phase transitions in clustering," *Physical Review Letters*, vol. 65, pp. 945–948, 1990.
[23] ——, "Vector quantization by deterministic annealing," *IEEE Trans. Inform. Theory*, vol. 38, pp. 1249–1257, July 1992.
[24] J. E. Shore and R. W. Johnson, "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 26–37, 1980.

Fast Tree-Structured Nearest Neighbor Encoding for Vector Quantization

Ioannis Katsavounidis, C.-C. Jay Kuo, and Zhen Zhang

Abstract—**This work examines the nearest neighbor encoding problem with an unstructured codebook of arbitrary size and vector dimension. We propose a new tree-structured nearest neighbor encoding method that significantly reduces the complexity of the full-search method without any performance degradation in terms of distortion. Our method consists of efficient algorithms for constructing a binary tree for the codebook and nearest neighbor encoding by using this tree. Numerical experiments are given to demonstrate the performance of the proposed method.**

I. Introduction

Nearest neighbor (NN) encoding [1] is finding the nearest point for an unknown input point from a set of fixed point—called the codebook in vector quantization (VQ)—in a k-dimensional vector space. Its fundamental role in the VQ area is indicated by the fact that NN encoding is a synonym for vector quantizing. Moreover, it constitutes the major computational task of the generalized Lloyd algorithm (GLA), which is the most commonly used algorithm for VQ codebook design. It is also used extensively in pattern classification and decision-making problems. A straightforward solution to this problem is the full-search method, which involves an exhaustive search of the distances for all available points; in this way, the complexity of encoding each component of a vector point with full search is proportional exponentially to the dimension k and the bit rate r.

Because of its importance, many researchers have looked into this problem in an effort to find ways to accelerate the vector quantization process. We can classify previous work into two groups. The first group consists of methods that do not solve the nearest neighbor problem itself but instead seek a suboptimal solution that is almost as good in the sense of mean squared error (MSE). One such method is to use a tree-structured codebook search. In tree-structured VQ (TSVQ) [1], the search is performed in stages. In each stage, a substantial subset of candidate vectors is eliminated from consideration by a relatively small number of operations. In a binary tree search, the input code vector is compared with two predesigned test vectors at each stage or node of the tree. The nearest test vector determines which of two paths through the tree to select in order to reach the next stage of testing. At each stage, the number of candidate code vectors is reduced to roughly half the previous set of candidates. Efficient TSVQ design often requires simultaneous design of the codebook and tree structure. Various methods for TSVQ codebook design have been proposed, such as splitting [2] and single-node-splitting [3]. Another recent approach is fine-coarse VQ [4] that operates on arbitrary unstructured codebooks, claiming only a slight increase in distortion over full search but with a substantial improvement in speed.

The second group addresses an exact solution of the nearest neighbor encoding problem with less computation than that of exhaustive search. A very simple yet effective method is the *partial distortion calculation* reported by Bei and Gray in [5]. It is important to note that this method requires no memory overhead, but only provides moderate acceleration—about four times over full exhaustive search. Other methods include the *projection method* [6] and its variants [7], [8]. Binary hyperplane testing [9] and its more general form, i.e., $K-d$ trees [10], [11] have also been widely used for fast search. The design of optimal search trees suffers the "curse of dimensionality," which in this problem is expressed in the form of extremely high computational complexity. Thus, their applicability has been limited to small vector dimension and high-resolution cases. A recent work by Ramasubramanian and Paliwal [12] on the optimization of $K-d$ trees provides a family of search algorithms that are quite promising. Unfortunately, they do not guarantee the nearest neighbor classification of an arbitrary vector. Other work on this problem is centered around the use of the triangle inequality property of metric spaces. For details about fast NN-encoding algorithms built on this idea, see [13]–[15].

In this research, we focus on a fast tree-structured nearest neighbor encoder that significantly reduces the complexity of the full-search method without any performance degradation in terms of distortion. Both the construction of a binary tree-structured codebook and nearest neighbor encoding with such a tree-structured codebook are examined. The execution time of these two tasks is so small that it allows the application of the proposed method to slowly adaptive VQ schemes, where adaptation takes place after a number of vectors have been coded. One such case is the GLA, where a very large number of training vectors must be quantized for every iteration, while the update of the codebook only takes place once after every iteration.

This work is organized as follows. After introducing some basic geometrical properties in Section II, we present a new method where a binary search tree with respect to an arbitrary codebook can be obtained, and the nearest neighbor of an input vector can be located effectively in Section III. The performance of the proposed method is reported in Section IV.

II. Basic Geometrical Properties

Let C be a subset of the k-dimensional Euclidean space R^k with cardinality $|C| = N$. We denote the elements of C by c_i,

Manuscript received June 30, 1994; revised June 19, 1995. This work was supported by the National Science Foundation Presidential Faculty Fellow (PFF) Award ASC-9350309.

I. Katsavounidis and C.-C. J. Kuo are with the Signal and Image Processing Institute and the Department of Electrical Engineering Systems, University of Southern California, Los Angeles, CA 90089-2564 USA.

Z. Zhang is with the Communication Science Institute and the Department of Electrical Engineering Systems, University of Southern California, Los Angeles, CA 90089-2565 USA.

Publisher Item Identifier S 1057-7149(96)01308-5.

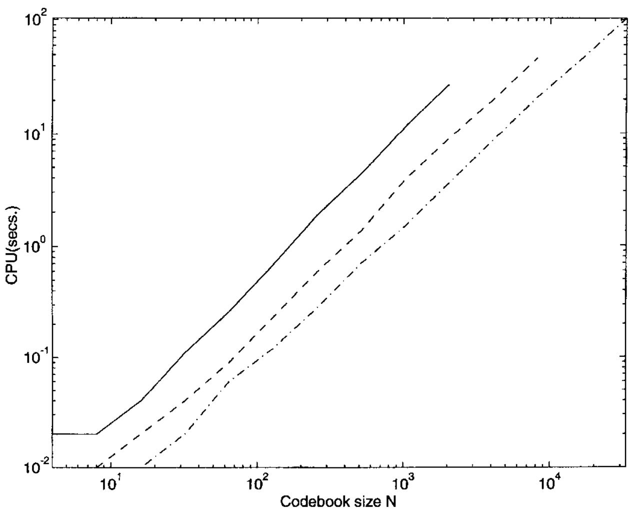

Fig. 1. Plot of tree creation CPU time versus codebook size (solid line: 8×8 block size; dashed line: 4×4 block size; dashed-dotted line: 2×2 block size).

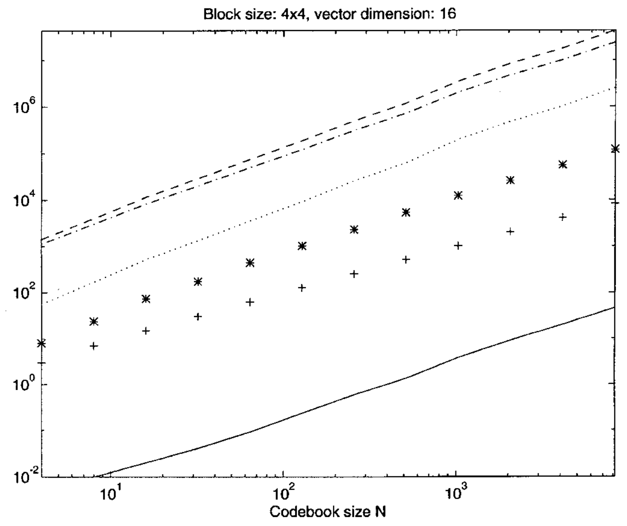

Fig. 2. Plot of tree creation MAC's versus codebook size (solid line: CPU time; dashed-dotted line: multiplications; dashed line: additions; dotted line: comparisons; +: square roots; *: quick-sorts).

$i \in \{1, \cdots, N\}$ and the individual components of the vectors by c_i^j, $j \in \{1, \cdots, k\}$. The nearest neighbor (NN) encoding problem can be stated as follows. For an arbitrary vector $x \in R^k$, find the vector c_i such that

$$\forall c_j \in C, \quad j \neq i \Rightarrow d(x, c_j) \geq d(x, c_i)$$

where $d(x, y)$ can be any distance function between two vectors in R^k. We focus on the Euclidean distance in this work.

Let $c_1, c_2 \in R^k$, $c_1 \neq c_2$, and define

$$w^0 = \frac{1}{2d(c_1, c_2)} \sum_{j=1}^{k} [(c_1^j)^2 - (c_2^j)^2]$$

and

$$w^j = \frac{c_2^j - c_1^j}{d(c_1, c_2)}, \quad 1 \leq j \leq k. \tag{1}$$

By augmenting vectors x and w with one additional component, we can define vectors of dimension $k + 1$:

$$\tilde{x} = (x^0, x^1, x^2, \cdots, x^k),^T \quad \text{where} \quad x^0 = 1,$$

and

$$\tilde{w} = (w^0, w^1, w^2, \cdots, w^k).^T$$

It is easy to verify the following nearest neighbor encoding rule:

$$d(x, c_1) < d(x, c_2) \Longleftrightarrow \langle \tilde{x}, \tilde{w} \rangle < 0. \tag{2}$$

Thus, the problem of finding the closest distance between an arbitrary input vector x and two given vectors c_1 and c_2 involves the calculation of just one inner product instead of two. In fact, the $k+1$ dimensional vector $\tilde{w}$ defines a hyperplane in R^k

$$H(\tilde{w}) = \{x \in R^k : \langle \tilde{x}, \tilde{w} \rangle = 0\}$$

which partitions the space R^k into two regions, where $d(x, c_1) < d(x, c_2)$ in one and $d(x, c_1) > d(x, c_2)$ in the other. The above derivation is a well-known fact and has been used in many decision problems.

Furthermore, we can obtain the minimum distance between an arbitrary vector x and the hyperplane $H(\tilde{w})$ as

$$d[x, H(\tilde{w})] = |\langle \tilde{x}, \tilde{w} \rangle|.$$

Motivated by the above equations, we define the *signed distance* from a point x to a hyperplane $H(\tilde{w})$ as

$$d_s[x, H(\tilde{w})] = \langle \tilde{x}, \tilde{w} \rangle.$$

Note that the signed distance has the same magnitude as that of the regular distance, and takes the minus (or plus) sign if x is closer to c_1 (or c_2) as indicated in (2).

Next, we present a theorem that will be needed in the tree-search algorithm given in Section III-B.

Theorem 1: Let $x, y \in R^k$ and $H(\tilde{w})$ be defined as above. Then we have

$$d(x, y) \geq |d_s[x, H(\tilde{w})] - d_s[y, H(\tilde{w})]|.$$

Proof: By definition, we have

$$d_s[x, H(\tilde{w})] = \langle \tilde{x}, \tilde{w} \rangle$$

and

$$d_s[y, H(\tilde{w})] = \langle \tilde{y}, \tilde{w} \rangle$$

so that

$$\begin{aligned} d_s[x, H(\tilde{w})] - d_s[y, H(\tilde{w})] &= \langle \tilde{x}, \tilde{w} \rangle - \langle \tilde{y}, \tilde{w} \rangle \\ &= \sum_{j=1}^{k} x^j w^j - \sum_{j=1}^{k} y^j w^j. \end{aligned}$$

By squaring the above expression, we obtain

$$\begin{aligned} &\{d_s[x, H(\tilde{w})] - d_s[y, H(\tilde{w})]\}^2 \\ &= \left[\sum_{j=1}^{k} (x^j - y^j) w^j\right]^2 \\ &= \frac{\left[\sum_{j=1}^{k} (x^j - y^j)(c_2^j - c_1^j)\right]^2}{d^2(c_1, c_2)}. \end{aligned} \tag{3}$$

By applying Schwartz's inequality to (3) we get

$$\begin{aligned}\{d_s[x, H(\tilde{w})] - d_s[y, H(\tilde{w})]\}^2 &\le \frac{\sum_{j=1}^{k}(x^j - y^j)^2 \sum_{j=1}^{k}(c_2^j - c_1^j)^2}{d^2(c_1, c_2)} \\ &= \frac{d^2(x,y)d^2(c_1,c_2)}{d^2(c_1,c_2)} \\ &= d^2(x, y),\end{aligned}$$

and the proof is completed. ■

III. Fast Nearest Neighbor Encoding

A. Binary Tree-Structured Codebook Design

The most commonly used procedure for designing vector codebooks from a set of training vectors is the so-called GLA, also referred to as the LBG algorithm after [2]. It is a two-step optimization algorithm that iteratively improves (in the sense of MSE) some initial codebook until it reaches a local minimum.

Most researchers have applied the GLA in codebook design. In this work, we use it as the main tool for building a binary tree-structured codebook from a given unstructured codebook. It is important to note that the first step in the GLA is nothing but NN-encoding of all the training vectors with respect to the existing codebook. It is for this reason that the GLA is probably the perfect testbed experiment for all fast NN-encoding methods. It is also clear that the codebook changes at every iteration. Thus, in order to apply a fast NN-encoding algorithm to the GLA, the preprocessing time (i.e., the execution time needed by the algorithm to set up all necessary data structures for a given codebook) should be relatively small.

Our tree construction algorithm is stated below.

Algorithm 1—Tree-Construction Algorithm:

1) Initialization: Store the entire codebook at the root of a binary tree.
2) With the codebook stored at the node, run the GLA to obtain the two centroids C_1 and C_2 that best represent it; determine the hyperplane $H(\tilde{w})$ that separates the two Voronoi cells; and calculate the signed distances between all the codewords and the hyperplane $d_s[c_i, H(\tilde{w})]$, $i = 1, \cdots, N$. Create two child nodes. Store the indexes of the codewords that have negative distance from the hyperplane at the left node; store those with positive distance at the right node.
3) We repeat the process in Step 2 until each leaf contains only one codeword.

It is evident that the above algorithm leads to a binary tree that can be balanced or unbalanced depending on the input codebook. Our experiments have shown that even though the resulting tree is almost always unbalanced, it does not have a high degree of nonuniformity. A balanced binary tree of N items has a depth of $\log_2 N + 1$ while the averaged depth with our algorithm is approximately $\log_2 N + 3$. Thus, the computation time required to search this binary tree is still low. Since the tree is dependent on the input codebook, computational complexity is random and we can only consider average performance. Moreover, each step involves a call to the GLA, which is an iterative optimization procedure with unknown number of iterations, so that the problem of determining complexity becomes even more difficult. Fortunately, the GLA for the case of $N = 2$ requires significantly fewer iterations to converge than any other case. With conjunction to the maximum distance initialization technique [16] we observed that no more than five iterations are

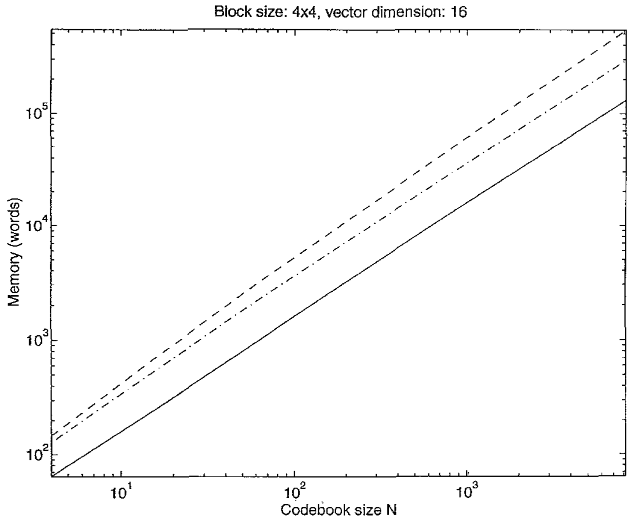

Fig. 3. Plot of memory requirements versus codebook size (solid line: exhaustive VQ; dashed-dotted line: TSVQ; dashed line: the new algorithm).

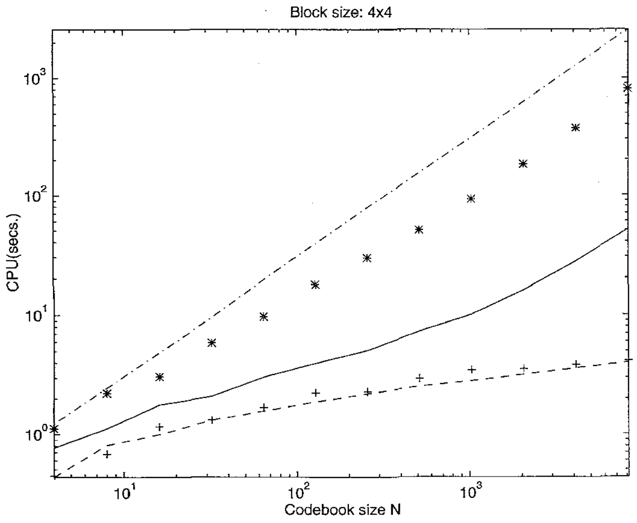

Fig. 4. Performance comparison for Couple image (solid line: the new method; dashed line: TSVQ; dashed-dotted line: full-distortion exhaustive search; *: partial-distortion exhaustive VQ; +: structured VQ—"depth only").

necessary for convergence. Thus, computational complexity of the algorithm can be estimated as $O(kN \log_2 N)$. Similarly, the memory requirements of the algorithm are $O(N \log_2 N)$, since for every level of the binary tree we need to store the index and the signed distance of each codeword. More details on the computational complexity and memory requirements of the tree-structuring algorithm are given in Section IV.

B. Nearest Neighbor Encoding with Tree-Structured Codebook

We are now ready to present our nearest neighbor encoding based on the tree-structured codebook constructed in Section III-A. For a given input vector x, we do the following.

Algorithm 2—Tree-Search Algorithm:

1) *Perform a greedy binary search from root to leaf.* Compute the signed distance of the input vector x to the hyperplane $H(\tilde{w})$ associated with the node. If the signed distance $d_s[x, H(\tilde{w})] \le$

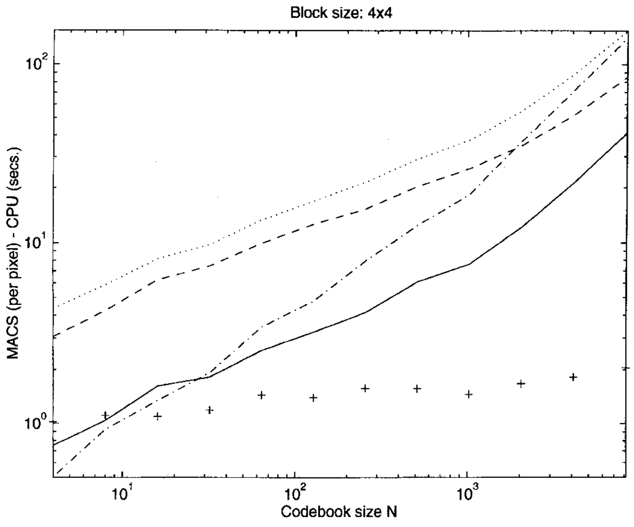

Fig. 5. Performance comparison for Elaine image with block size 4×4 with the new method (solid line: execution time; dashed line: multiplications; dotted line: additions; dashed-dotted line: comparisons; +: square roots per vector).

TABLE I
PERFORMANCE OF FULL-SEARCH, PARTIAL-DISTORTION VQ ON COUPLE.512

Block size	Code. size	Mult. (per pixel)	Add. (per pixel)	Comparisons (per pixel)	Sqrts. (per vector)
8×8	256	$7.225 \cdot 10^{+01}$	$1.444 \cdot 10^{+02}$	$7.906 \cdot 10^{+01}$	0
	512	$1.385 \cdot 10^{+02}$	$2.770 \cdot 10^{+02}$	$1.533 \cdot 10^{+02}$	0
	1024	$2.545 \cdot 10^{+02}$	$5.089 \cdot 10^{+02}$	$2.852 \cdot 10^{+02}$	0
	2048	$5.789 \cdot 10^{+02}$	$1.158 \cdot 10^{+03}$	$6.416 \cdot 10^{+02}$	0
4×4	1024	$1.867 \cdot 10^{+02}$	$3.734 \cdot 10^{+02}$	$3.129 \cdot 10^{+02}$	0
	2048	$3.661 \cdot 10^{+02}$	$7.322 \cdot 10^{+02}$	$6.202 \cdot 10^{+02}$	0
	4096	$7.221 \cdot 10^{+02}$	$1.444 \cdot 10^{+03}$	$1.232 \cdot 10^{+03}$	0
	8192	$1.518 \cdot 10^{+03}$	$3.036 \cdot 10^{+03}$	$2.540 \cdot 10^{+03}$	0
2×2	4096	$1.147 \cdot 10^{+03}$	$2.295 \cdot 10^{+03}$	$3.191 \cdot 10^{+03}$	0
	8192	$2.265 \cdot 10^{+03}$	$4.529 \cdot 10^{+03}$	$6.356 \cdot 10^{+03}$	0
	16384	$4.471 \cdot 10^{+03}$	$8.942 \cdot 10^{+03}$	$1.266 \cdot 10^{+04}$	0
	32768	$8.846 \cdot 10^{+03}$	$1.769 \cdot 10^{+04}$	$2.522 \cdot 10^{+04}$	0

0, we move to the left node; otherwise, we move to the right. The same process repeats until a leaf with only one codeword c_i is reached. The candidate codeword c_i and its distance $d_{\min}$ from x are reported to the parent node. We move from the leaf node to its parent node.

2) *Search the best candidate codeword from leaf to root.* At a given node, we search all codewords associated with the node that have a signed distance from the hyperplane with an opposite sign to that of $d_s[x, H(\tilde{w})]$, starting with the one that has the shortest distance from the hyperplane and considering only those code vectors c_j satisfying $|d_s[x, H(\tilde{w})] - d_s[c_j, H(\tilde{w})]| < d_{\min}$. For each eligible candidate, apply the partial distance calculation method, i.e., complete the distortion calculation for as long as the partial sums D_l satisfy $D_l = \sum_{m=1}^{l} (c_j^m - x^m)^2 < d_{\min}^2$ to determine whether it is the nearest neighbor of x. By doing so, we can find the best candidate codeword with respect to this node. Then we move to its parent node and repeat the process until the root is reached.

TABLE II
PERFORMANCE OF TREE-STRUCTURED VQ ON HEATHER.512

Block size	Code. size	Mult. (per pixel)	Add. (per pixel)	Comparisons (per pixel)	Sqrts. (per vector)
8×8	256	$9.518 \cdot 10^{+00}$	$9.667 \cdot 10^{+00}$	$3.131 \cdot 10^{-01}$	0
	512	$1.117 \cdot 10^{+01}$	$1.134 \cdot 10^{+01}$	$3.646 \cdot 10^{-01}$	0
	1024	$1.433 \cdot 10^{+01}$	$1.455 \cdot 10^{+01}$	$4.633 \cdot 10^{-01}$	0
	2048	$1.617 \cdot 10^{+01}$	$1.642 \cdot 10^{+01}$	$5.210 \cdot 10^{-01}$	0
4×4	1024	$1.288 \cdot 10^{+01}$	$1.369 \cdot 10^{+01}$	$1.673 \cdot 10^{+00}$	0
	2048	$1.440 \cdot 10^{+01}$	$1.530 \cdot 10^{+01}$	$1.863 \cdot 10^{+00}$	0
	4096	$1.609 \cdot 10^{+01}$	$1.710 \cdot 10^{+01}$	$2.074 \cdot 10^{+00}$	0
	8192	$1.804 \cdot 10^{+01}$	$1.917 \cdot 10^{+01}$	$2.317 \cdot 10^{+00}$	0
2×2	4096	$1.398 \cdot 10^{+01}$	$1.748 \cdot 10^{+01}$	$7.241 \cdot 10^{+00}$	0
	8192	$1.507 \cdot 10^{+01}$	$1.884 \cdot 10^{+01}$	$7.786 \cdot 10^{+00}$	0
	16384	$1.631 \cdot 10^{+01}$	$2.039 \cdot 10^{+01}$	$8.407 \cdot 10^{+00}$	0
	32768	$1.736 \cdot 10^{+01}$	$2.171 \cdot 10^{+01}$	$8.932 \cdot 10^{+00}$	0

The validity of the algorithm just presented is obvious. The nearest neighbor is chosen via two elimination processes. First, since we have $d(x, c) \geq |d_s[x, H(\tilde{w})] - d_s[c, H(\tilde{w})]|$ from Theorem 1, if a codeword c satisfies the relation $|d_s[x, H(\tilde{w})] - d_s[c, H(\tilde{w})]| \geq d(x, c_i)$, we can eliminate c as a candidate for the nearest neighbor codeword. The second elimination process comes from the partial-distance search method that is self evident. Note that the encoding algorithm can be implemented very efficiently by having the signed distances of the codewords precalculated, ordered, and stored at each node when the binary tree is constructed in algorithm 1. It should be noted that the first step of the search algorithm is identical to tree-structured VQ encoding. One can thus terminate the search algorithm at this point, obtaining a suboptimal solution to the nearest neighbor problem. In the following, we will refer to this variant of the search algorithm as the "depth-only" structured VQ algorithm in order to differentiate it from TSVQ, since the former applies to arbitrary unstructured codebooks while the later requires the parallel design of the tree structure and the vector codebook.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Experimental Results

To test our method, we used the "basic" VQ scheme, i.e., the simplest VQ scheme applied to image blocks where each component of the vectors is the intensity of an image pixel. It is, however, important to point out that our method can be applied to any VQ scheme such as predictive VQ, adaptive VQ, transform VQ, etc.

We first used the GLA to obtain a number of codebooks of various dimensions (k) and sizes (N). The training vectors were obtained from the Baboon, Lenna, and Boat images taken from the USC image database. All these are monochrome images of the luminance (NTSC-Y) component. They have size 512×512 (in pixels) with 8 b/pixel (256 grey levels). We partitioned these images into blocks of dimension 2×2, 4×4, and 8×8, and generated training vectors of dimensions $k = 4$, 16, and 64, correspondingly. In the codebook design, we adopted the maximum distance method [16] for its initialization, performed a number of GLA iterations, and stopped the iterations when there was no significant reduction in the mean squared error (MSE) values. We chose codebook sizes of the form $N = 2^n$ with integer n in the experiments. The resulting codebook sizes are up to 2048, 8192, and 32 768 for the cases of

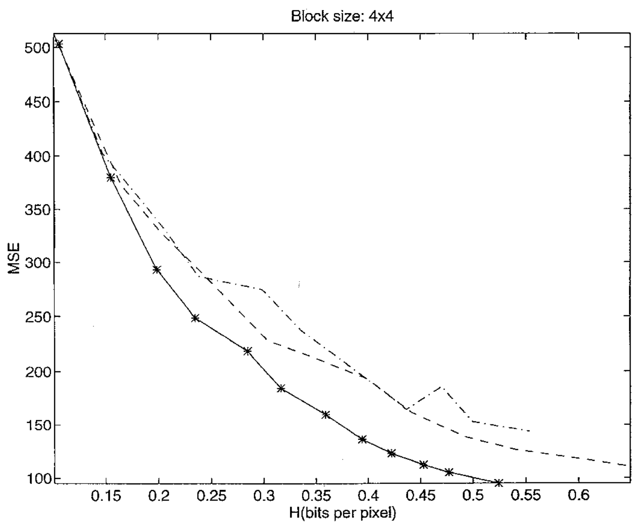

Fig. 6. Empirical rate-distortion function for the Couple image (solid line: the new method; dashed line: TSVQ; dashed-dotted line: Structured-VQ "depth only;" *: full-distortion exhaustive VQ; +: partial-distortion exhaustive VQ).

block-size 8×8, 4×4, and 2×2, respectively. In order to compare our algorithm with tree-structured VQ, we also ran the single-node splitting LBG algorithm [3] on the same training sequences and obtained tree-structured codebooks of the same sizes and dimensions.

We implemented the following five VQ encoding options.

1) Exhaustive full-distortion search VQ
2) Exhaustive partial-distortion search VQ
3) Tree-structured VQ (TSVQ)
4) Fast codebook structuring and nearest neighbor (NN) searching VQ (the new algorithm)
5) Fast codebook structuring and depth-only searching VQ (the new algorithm terminated just after the first leaf of the tree is reached)

In what follows, we maintain the following notation:

u	input (original) image
$\tilde{u}$	output (quantized) image
p_i	empirical distribution

$$p_i = \frac{\text{no. of vectors classified to codeword } c_i}{\text{total no. of input vectors}}.$$

For every experiment, we reported the following quantities:

1) Resulting average distortion (MSE) per pixel and entropy of the encoded image (H) in bits per pixel based on the empirical distribution of the code vectors. These quantities are calculated as follows.

$$\text{MSE} = \frac{1}{512^2} \sum_{i=1}^{512} \sum_{j=1}^{512} [u(i, j) - \tilde{u}(i, j)]^2$$

and

$$\text{H} = -\frac{1}{k} \sum_{i=1}^{N} p_i \log_2 (p_i)$$

where N is the codebook size and k is the vector dimension.

2) Execution time of the various algorithms in terms of seconds of CPU on a Hewlett-Packard Apollo Series 700 workstation.
3) Number of multiplications, additions, and comparisons (MAC's) per pixel; also the number of square root operations per vector, since it is needed by the new algorithm.
4) Total memory requirements of the algorithms.

TABLE III
PERFORMANCE OF FAST NEAREST NEIGHBOR TREE-STRUCTURED VQ ON ELAINE.512

Block size	Code. size	Mult. (per pixel)	Add. (per pixel)	Comparisons (per pixel)	Sqrts. (per vector)
8×8	256	$1.934 \cdot 10^{+01}$	$2.624 \cdot 10^{+01}$	$6.964 \cdot 10^{+00}$	$1.479 \cdot 10^{+00}$
	512	$2.593 \cdot 10^{+01}$	$3.800 \cdot 10^{+01}$	$1.340 \cdot 10^{+01}$	$1.636 \cdot 10^{+00}$
	1024	$3.432 \cdot 10^{+01}$	$5.321 \cdot 10^{+01}$	$2.254 \cdot 10^{+01}$	$1.775 \cdot 10^{+00}$
	2048	$5.142 \cdot 10^{+01}$	$8.233 \cdot 10^{+01}$	$3.861 \cdot 10^{+01}$	$1.875 \cdot 10^{+00}$
4×4	1024	$2.565 \cdot 10^{+01}$	$3.740 \cdot 10^{+01}$	$1.833 \cdot 10^{+01}$	$1.460 \cdot 10^{+00}$
	2048	$3.478 \cdot 10^{+01}$	$5.502 \cdot 10^{+01}$	$3.686 \cdot 10^{+01}$	$1.672 \cdot 10^{+00}$
	4096	$5.218 \cdot 10^{+01}$	$8.853 \cdot 10^{+01}$	$7.210 \cdot 10^{+01}$	$1.816 \cdot 10^{+00}$
	8192	$8.493 \cdot 10^{+01}$	$1.526 \cdot 10^{+02}$	$1.392 \cdot 10^{+02}$	$2.048 \cdot 10^{+00}$
2×2	4096	$2.654 \cdot 10^{+01}$	$4.568 \cdot 10^{+01}$	$4.126 \cdot 10^{+01}$	$1.769 \cdot 10^{+00}$
	8192	$3.427 \cdot 10^{+01}$	$6.099 \cdot 10^{+01}$	$6.714 \cdot 10^{+01}$	$1.807 \cdot 10^{+00}$
	16384	$4.475 \cdot 10^{+01}$	$8.175 \cdot 10^{+01}$	$1.030 \cdot 10^{+02}$	$1.852 \cdot 10^{+00}$
	32768	$5.594 \cdot 10^{+01}$	$1.036 \cdot 10^{+02}$	$1.397 \cdot 10^{+02}$	$1.874 \cdot 10^{+00}$

After obtaining various codebooks as stated above, we ran the tree-construction algorithm, i.e., algorithm 1, to build a binary tree for each codebook. Execution time versus codebook size for various vector dimensions is plotted in Fig. 1. One can verify the $O(kN \log_2 N)$ behavior as claimed in Section III-A. We also present overall performance, i.e., multiplications, additions, comparisons (MAC's), square-root operations, and quick-sort counts (needed for the sorting of the code-vector distances from the hyperplanes) in Fig. 2. They demonstrate the same relationship. It is worthwhile to point out that the execution time for the largest experiment, which has a block size 2×2 and a codebook size $N = 32\,768 = 2^{15}$, required approximately 100 seconds of CPU time to build the binary search tree. This is significantly less than the time required by existing algorithms that build a binary tree from an arbitrary codebook using the Monte-Carlo techniques by an order of magnitude.

The plot of the overall memory requirements of the various algorithms is presented in Fig. 3. The $O[N \log_2 (N)]$ overhead can be observed as predicted in Section III-A. Furthermore, it can be seen that the memory overhead of both the new method and that of tree-structured VQ is decreasing in comparison to the memory needed for storage of the codebook.

Next, we tested all five algorithms by encoding three different monochrome images: Couple, Elaine, and Heather of size 512×512. The execution times obtained for the Couple image is given in Fig. 4 as functions of the codebook size N with a vector dimension equal to 16. A similar performance is also observed for the other two images. In addition, the MAC's for various algorithms are listed on Tables I, II, and III. To show a general trend, we plot the data for the case of the Elaine image in Fig. 5. Based on these execution times, we list the speed-up factor, which is defined as the ratio of CPU time needed by exhaustive full-distortion VQ over CPU time needed by the other four algorithms in Table IV for various vector dimensions and codebook sizes. We see a similar general behavior from this table.

B. Discussion

It is important to note the degradation in performance (in terms of MSE and entropy) when using TSVQ instead of full-search VQ presented in Fig. 6 in the form of operational rate-distortion function for various methods. This is the cost paid to gain the advantage of the very low complexity of TSVQ. On the other hand, we can immediately see from the corresponding plots and tables

TABLE IV
SPEED-UP FACTOR OVER EXHAUSTIVE, FULL-DISTORTION VQ FOR VARIOUS METHODS: P = PARTIAL-DISTORTION FULL-SEARCH VQ; T = TREE-STRUCTURED VQ; S = STRUCTURED FULL-SEARCH VQ; D = DEPTH ONLY—FIRST STAGE OF STRUCTURED VQ

Vector	Code.	Couple				Elaine				Heather			
dim.	size	P	T	S	D	P	T	S	D	P	T	S	D
8 × 8	256	2.61	36.6	10.46	27.0	3.16	38.1	16.7	30.8	4.02	43.2	21.4	27.0
	512	2.73	64.4	13.80	53.4	3.37	64.9	23.1	55.8	3.89	72.6	35.9	52.6
	1024	2.98	109.7	18.17	102.6	3.64	114.9	32.9	101.7	4.15	115.4	48.4	90.9
	2048	2.62	194.2	21.66	157.4	3.15	201.6	41.4	154.9	3.41	207.2	68.8	121.9
4 × 4	1024	3.37	110.83	30.94	90.0	3.60	113.3	40.54	94.0	4.11	117.4	58.7	86.3
	2048	3.42	197.74	39.20	176.5	3.67	200.8	50.74	177.8	3.99	204.0	86.3	159.4
	4096	3.41	355.00	46.64	331.8	3.61	357.9	59.14	330.7	3.89	376.2	127.8	319.9
	8192	3.20	643.90	50.79	625.2	3.41	656.8	62.63	594.6	3.59	681.2	125.0	632.8
2 × 2	4096	1.83	315	103.1	296	1.83	319	105.9	295	1.85	328	137.8	297
	8192	1.83	579	137.6	584	1.83	597	146.8	589	1.85	608	212.4	599
	16384	1.77	1068	162.0	1131	1.77	1034	193.6	1133	1.79	1084	250.7	1186
	32768	1.77	1966	200.6	1958	1.77	1884	266.1	2025	1.80	2187	334.8	2134

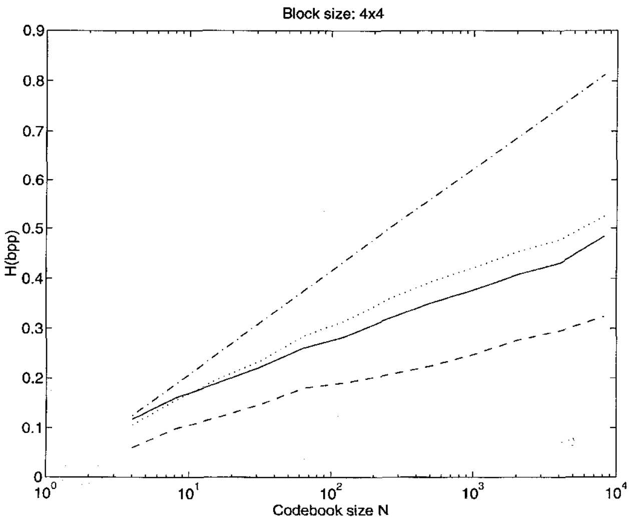

Fig. 7. Entropy versus codebook size (solid line: Elaine; dotted line: Couple; dashed line: Heather).

that both TSVQ and the depth-only version of the new algorithm are much faster than exhaustive full-distortion and partial-distortion VQ and also faster than the proposed NN-encoding algorithm. This is expected, since it is known that the complexity for both these algorithms is proportional to the average depth of the binary tree, which is essentially $O(\log N)$. These two suboptimal algorithms are comparable in terms of performance (MSE,H), having almost identical memory and CPU requirements. Thus, the tree-structuring algorithm (algorithm 1) provides an alternative to TSVQ design, with the additional advantage that it is applicable to any unstructured VQ codebook.

Another important observation is the significant variance in the execution time of the proposed algorithm for the same codebook size and vector dimension among different images. This variance in execution time is coupled by the variance we observe on the performance of the algorithm in terms of (MSE,H). This can be clearly seen in Fig. 7 where we plotted the entropy of the output image (measured in bits per pixel) versus codebook size for the three images. Each plot shows the entropy obtained for the three different images and their relationship to the theoretical vector quantizer resolution, defined as $r = \log_2 N/k$. Fig. 8 presents encoding time as a function of codebook size for the three images with block size 4×4 by using our algorithm. The resemblance is apparent; based on this, we

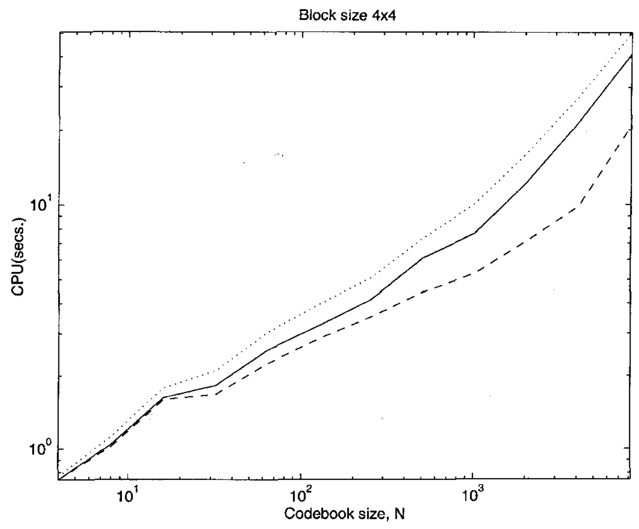

Fig. 8. Execution time versus codebook size (solid line: Elaine; dotted line: Couple; dashed line: Heather).

TABLE V
PARAMETERS α AND C FOR THE THREE IMAGES AND VARIOUS VQ METHODS

Image: Couple.512						
	8×8		4×4		2×2	
Method	α	C	α	C	α	C
FSVQ	3.3990	0.1014	2.9927	0.1241	2.6059	0.2903
PDVQ	2.7590	0.1307	2.6441	0.1537	2.4577	0.2733
TSVQ	1.3779	0.2673	1.2607	0.4415	1.1828	0.8838
STRUCT-VQ	2.0333	0.1590	1.8019	0.2345	1.4311	0.7527
STRUCT-VQ-TSVQ	1.4918	0.2134	1.3559	0.3627	1.2171	0.8452
Image: Elaine.512						
	8×8		4×4		2×2	
Method	α	C	α	C	α	C
FSVQ	4.2439	0.0486	3.9030	0.0792	2.4897	0.3393
PDVQ	3.3177	0.0587	3.1972	0.0837	2.3569	0.2998
TSVQ	1.4065	0.2366	1.2819	0.4059	1.1825	0.8346
STRUCT-VQ	2.0071	0.1367	1.9293	0.1822	1.3868	0.8270
STRUCT-VQ-TSVQ	1.8668	0.1603	1.8223	0.2056	1.3883	0.8039
Image: Heather.512						
	8×8		4×4		2×2	
Method	α	C	α	C	α	C
FSVQ	8.5659	0.0804	6.8074	0.1269	3.0245	0.6859
PDVQ	6.0339	0.0729	5.0997	0.1183	2.8854	0.5243
TSVQ	1.5779	0.3036	1.3709	0.4846	1.2120	0.9764
STRUCT-VQ	2.4272	0.2042	2.0783	0.3237	1.4332	1.0752
STRUCT-VQ-TSVQ	2.1381	0.1769	1.6185	2.6094	1.2389	1.0404

conclude that execution time relates to entropy H of the output more than it does to the codebook size N.

The shape of the graphs motivates us to model the logarithm of the execution time linearly with respect to entropy. By performing a simple linear regression, we can model execution time as a function of entropy as

$$T = C\alpha^{Hk}$$

where the parameters C and α are listed in Table V. The table clearly shows the gain provided by our new algorithm. While the encoding complexity of full search grows as a function $O(\alpha_1^{Hk})$, the proposed algorithm grows as $O(\alpha_2^{Hk})$, where the ratio of α_1 and α_2 is in the range $1.7 \leq \alpha_1/\alpha_2 \leq 3.5$. Thus, the proposed method performs increasingly faster than full search as codebook size increases. It provides a clear advantage in satisfying the need for large codebook sizes and vector dimensions leading to better compression.

V. Conclusion and Extension

In this work, we proposed a tree-construction algorithm and a tree-search algorithm to achieve fast nearest neighbor encoding. The superior performance of the proposed method was demonstrated by a set of numerical experiments.

We would like to mention some related research work to be carried out in the future. The study of existing methods has so far been very enlightening as to various alternatives for the partial-distortion elimination of codewords, which at this point appears to be the computational bottleneck of our encoding algorithm. Preliminary results, reported in [17], are quite promising. We can also augment existing tree codebook structures with the indexes and the signed distances from the hyperplanes as described by algorithm 1, and use the new algorithm as a refinement to TSVQ for cases where the improvement in the (MSE,H) substantiates increased memory and CPU complexity. In addition, we believe that there is much more to the relationship between entropy, distortion, computational complexity, and memory capacity than this work is able to address. A more complete analysis that provides the operational rate-distortion function, i.e., the achievable rate-distortion pair under some constraints of bounded computational complexity and/or memory capacity should be of great interest.

References

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

[2] Y. Linde, A. Buzo, and R. M. Gray, “An algorithm for vector quantizer design,” *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, Jan. 1980.

[3] J. Makhoul, S. Roucos, and H. Gish, “Vector quantization in speech coding,” in *Proc. IEEE*, vol. 43, no. 11, pp. 1551–1587, Nov. 1985.

[4] N. Moayeri, D. L. Neuhoff, and W. E. Stark, “Fine-coarse vector quantization,” *IEEE Trans. Signal Processing*, vol. 39, no. 7, pp. 1503–1515, July 1991.

[5] C.-D. Bei and R. M. Gray, “An improvement of the minimum distortion encoding algorithm for vector quantization,” *IEEE Trans. Commun.*, vol. COM-33, no. 10, pp. 1132–1133, Oct. 1985.

[6] D. Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, “Fast search algorithms for vector quantization and pattern matching,” in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, San Diego, Mar. 1984, pp. 9.11.1–9.11.4.

[7] J. H. Friedman, F. Baskett, and L. J. Shustek, “An algorithm for finding nearest neighbors,” *IEEE Trans. Comput.*, vol. C-24, no. 10, pp. 1000–1006, Oct. 1975.

[8] S.-W. Ra and J.-K. Kim, “A fast mean-distance-ordered partial codebook search algorithm for image vector quantization,” *IEEE Trans. Circuits Syst. II*, vol. 40, no. 9, pp. 576–579, Sept. 1993.

[9] D. Y. Cheng and A. Gersho, “A fast codebook search algorithm for nearest-neighbor pattern matching,” in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Tokyo, vol. 1, Apr. 1986, pp. 265–268.

[10] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Comm. ACM*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[11] A. Lowry, S. Hossain, and W. Millar, “Binary search trees for vector quantization,” in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Dallas, 1987, pp. 2205–2208.

[12] V. Ramasubramanian and K. K. Paliwal, “Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding,” *IEEE Trans. Signal Processing*, vol. 40, no. 3, pp. 518–531, Mar. 1992.

[13] M. Orchard, “A fast nearest neighbor search algorithm,” in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Toronto, Canada, May 1991, pp. 2297–2300.

[14] M. R. Soleymani and S. D. Morgera, “A fast mmse encoding technique for vector quantization,” *IEEE Trans. Commun.*, vol. 37, no. 6, pp. 656–659, June 1989.

[15] L. Torres and J. Huguet, “An improvement on codebook search for vector quantization,” *IEEE Trans. Commun.*, vol. 42, nos. 2–4, pp. 208–210, 1994.

[16] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, “A new initialization technique for generalized lloyd iteration,” *IEEE Signal Processing Lett.*, vol. 1, no. 10, pp. 144–146, Oct. 1994.

[17] ——, “Fast generalized lloyd iteration for vq codebook design,” in *1995 IS&T/SPIE Symp. Electron. Imaging: Sci. Technol.*, San Jose, CA, Feb. 1995.