Fast K-Dimensional Tree Algorithms for Nearest Neighbor Search with Application to Vector Quantization Encoding

V. Ramasubramanian and Kuldip K. Paliwal, *Member, IEEE*

Abstract—**In this paper, fast search algorithms are proposed and studied for vector quantization encoding using the K-dimensional (K-d) tree structure. Here, the emphasis is on the optimal design of the K-d tree for efficient nearest neighbor search in multidimensional space under a bucket-Voronoi intersection search framework. Efficient optimization criteria and procedures are proposed for designing the K-d tree, for the case when the test data distribution is available (as in vector quantization applications in the form of training data) as well as for the case when the test data distribution is not available and only the Voronoi intersection information is to be used. The proposed optimization criteria and bucket-Voronoi intersection search procedure are studied in the context of vector quantization encoding of speech waveform and are empirically observed to achieve constant search complexity for $O(\log N)$ tree depths. Comparisons are made with other optimization criteria—the maximum product criterion and Friedman *et al.*'s optimization criterion—and the proposed criteria are found to be more efficient in reducing the search complexity. Under the framework used for obtaining the proposed optimization criteria, a geometric interpretation is given for the maximum product criterion explaining the reasons for its inefficiency with respect to the proposed optimization criteria.**

I. Introduction

NEAREST neighbor search consists of determining the closest point to a query point among N points in K-dimensional (K-d) space. This search is widely used in several areas such as pattern classification, nonparametric estimation, and data compression using vector quantization. Reducing the complexity of nearest neighbor search is of considerable interest in these areas, and particularly in vector quantization encoding. In this paper, we discuss fast nearest neighbor search in the context of vector quantization.

Vector quantization is a powerful data compression technique which has become very popular recently in a number of areas such as speech coding, image coding, and speech recognition [1]–[4]. Vector quantization has the potential to achieve coding performance close to the rate-distortion limit with increasing vector dimension. However, the utilization of vector quantizers is severely limited by its encoding complexity which increases exponentially with dimension K [1]–[4]. Vector quantization encoding is the minimum-distortion quantization of a vector $\boldsymbol{x} = (x_1, \cdots, x_K)$ (referred to as the test vector), using a given set of N K-dimensional codevectors (called the codebook $\boldsymbol{C} = \{\boldsymbol{c}_j\}_{j=1,\cdots,N}$, of size N), under some distance measure $d(\boldsymbol{x}, \boldsymbol{y})$. This involves finding the nearest neighbor of $\boldsymbol{x}$ in $\boldsymbol{C}$, given by $\boldsymbol{q}(\boldsymbol{x}) = \boldsymbol{c}_k : d(\boldsymbol{x}, \boldsymbol{c}_k) \leq d(\boldsymbol{x}, \boldsymbol{c}_j), j = 1, \cdots, N$, which requires N vector distance computations $d(\boldsymbol{x}, \boldsymbol{c}_j)$ using the exhaustive full search for a codebook of size N. The codebook size N is related to the dimension K and bit-rate r (bits/sample) as $N = 2^{Kr}$ and the complexity of encoding $\boldsymbol{x}$ increases exponentially with K and r. Thus, the problem of reducing the computational complexity of vector quantization encoding becomes important for realizing the full potential of vector quantization and for rendering it practically useful for real-time applications.

An important approach towards fast nearest neighbor search in K-dimensions is the use of data structures which facilitate fast search of the codebook which is normally unstructured. In this context, the K-d (K-dimensional) tree structure developed by Bentley [5] is a powerful structure which has been used recently for fast nearest neighbor search [6]–[13]. This multidimensional binary tree structure was originally used by Bentley to carry out fast associative searches on multidimensional data for answering a wide range of information retrieval queries such as intersection query, region search, exact match, partial and closest match queries from files with multiple key records. The optimization of the K-d tree during its design is a very important issue which decides the efficiency of the tree in reducing the search complexity. The optimization of the tree involves the choice of the dividing hyperplane for each nonterminal node in the tree such that the resulting tree structure when used for nearest neighbor search using a specified search procedure yields the minimum search time complexity. This issue has been addressed in [6]–[10].

In [6], Friedman *et al.* proposed general prescriptions for optimizing the tree to minimize the expected search time under a backtracking search procedure. The algorithm based on this optimization and backtracking search

Manuscript received May 2, 1990; revised January 17, 1991.

V. Ramasubramanian is with the Computer Systems and Communications Group, Tata Institute of Fundamental Research, Bombay-400 005, India.

K. K. Paliwal is with AT&T Bell Laboratories, Murray Hill, NJ 07974, on leave from the Tata Institute of Fundamental Research, Bombay, India.

IEEE Log Number 9105654.

has an $O(\log N)$ average complexity performance. However, the back-tracking search has high computational overhead and the average complexity bound has a 2^K dependence. In many practical applications such as real-time vector quantization encoding, the worst case complexity is also of considerable importance, in addition to average computational complexity. In particular, in vector quantization applications where it is of interest to use large values of K for $r \leq 1$ b/sample, the codebook size is $N \leq 2^K$ and the backtracking search with a performance bound of the order of 2^K will offer a bad worst case performance. Moreover, the optimization was developed under the restriction that no knowledge of the test data distribution is available and, as a result, does not truly minimize the expected search time for any particular distribution. The approach also does not provide any means of optimization for a given distribution even if it is known *a priori*.

An alternate approach to the use of the K-d tree structure for fast nearest neighbor search is the bucket-Voronoi intersection framework. Here each leaf (or bucket) of the tree is associated with a set of codevectors whose Voronoi regions intersect with the region defined by the bucket region. The search involves first locating the bucket containing the test vector and subsequently performing a full search among the set of codevectors associated with the bucket. The bucket-Voronoi intersection based search procedure is simple and direct and can offer significant complexity reduction over the algorithm based on a backtracking search which has high computational overheads and worst case complexity. The design of the K-d tree under this framework has to consider the Voronoi information explicitly and it is important to obtain efficient optimization procedures for this. The fast search procedures reported in [7]–[10] fall within this framework with particular emphasis on the optimization of the K-d tree. In [11]–[13], the optimization prescribed by Friedman *et al.* [6] is used to organize the training data into buckets for performing fast agglomerative clustering and the emphasis here was more on exploiting the partitioning and space localization offered by the K-d tree in the form of buckets.

In the paper, we address the issue of optimizing the K-d tree under the bucket-Voronoi intersection search in detail, with particular reference to the optimization procedures proposed by us in the preliminary reports [9], [10]. The paper is organized as follows. In Section II, we describe the basic K-d tree structure. In Section III, we briefly describe the optimization and backtracking search procedure of Friedman *et al.* [6]. Section IV describes the bucket-Voronoi intersection framework. In Section V, we consider the optimization of the K-d tree under the bucket-Voronoi intersection in detail. Section V-A addresses the optimization given the test data distribution and presents the exact optimization criterion (EOC) proposed by us in [9]. Here, the optimization involves minimizing the expected search complexity using both the statistical and geometric information available in the form of test data distribution and Voronoi regions of the given codevectors. This results in the most direct optimization possible in the minimization of the expected search time locally at every node of the K-d tree when the test data distribution is known. In the case of the vector quantization, information about the test data distribution is available in the form of training data on which the quantizer is designed by using algorithms such as the Linde–Buzo–Gray algorithm [15]. This training data is used for the optimization of the tree. However, in most other practical cases where *a priori* knowledge of the test data distribution is not available, this procedure cannot be applied and there is a need for good generalized optimization criteria which do not require the test data distribution. Section V-B addresses this problem of efficient optimization of the tree for the general case when the test data distribution is not known. Here, we present the generalized optimization criterion (GOC) proposed in [10] which uses only the Voronoi information. This criterion is obtained based on a geometric interpretation of the optimization problem using a direct characterization of the number of Voronoi intersections in the left and right partitions as a function of the partition location. In Section VI, we consider the maximum product criterion (MPC) used in [7], [8]. We give a geometric interpretation of the MPC and explain the reasons for its inefficiency with respect to the GOC optimization criterion. In Section VII, we describe the experiments and results obtained in the context of vector quantization of speech waveform. The conclusions are reported in Section VIII.

II. The K-Dimensional (K-d) Tree

The K-d tree is a generalization of the simple one-dimensional binary tree. In the general case of K-dimensions, the $\Re^K$ space is split into two half spaces by means of a hyperplane orthogonal to one of the K coordinate axes. Such a hyperplane $\boldsymbol{H}$, represented in general by $\boldsymbol{H} = \{x \in \Re^K : x_j = h\}$, defines the two half spaces, R_L and R_R as $R_L = \{x \in \Re^K : x_j \leq h\}$ and $R_R = \{x \in \Re^K : x_j \geq h\}$. This partitioning hyperplane is represented by just two scalar quantities: 1) j, the index to the coordinate axis orthogonal to the plane, and 2) h, the location of the plane on this axis. Any vector point x can now be located with respect to the dividing plane $\boldsymbol{H}$ by a single scalar comparison of the form $x_j \leq h$, i.e., the vector's jth component value with the partition value h. The initial region corresponds to the root of the tree at layer 1 and the two subregions R_L and R_R obtained by the division correspond to the left and right sons at layer 2. Each of these two half spaces are successively divided by hyperplanes orthogonal to the coordinate axes and d such successive divisions starting with the initial region as the root at layer 1 creates a tree of depth d with 2^d terminal regions termed "buckets" at the $(d+1)$th layer. Every nonterminal node is associated with a region and a partitioning hyperplane of the form $x : x_j = h$, which needs storage of just two scalar quantities (j, h) at each node. Given any vector in $\Re^K$, a sequence of d scalar comparisons of the vector's jth component value with the partitioning hyperplane (j, h) at

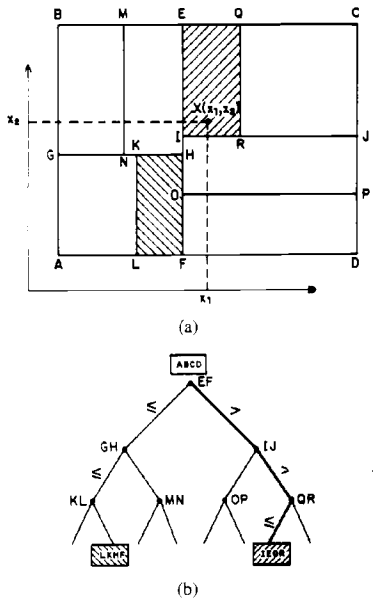


Fig. 1. Example of K-d tree partition.

that node leads to the leaf (or bucket) containing the vector. Thus, a K-d tree structure of depth d partitions the $\Re^K$ space into 2^d disjoint rectangular regions (buckets) and allows identification of the bucket containing a given vector $\boldsymbol{x}$ in just d scalar comparisons.

The basic structure of K-d tree is illustrated in Fig. 1 for a planar case. The root region *ABCD* is divided by hyperplane *EF* into two halves. Fig. 1(b) shows the corresponding tree of depth 3 generated by divisions of these two regions by various hyperplanes orthogonal to one of the coordinate axes x_1 or x_2. A vector $\boldsymbol{x} = \{x_1, x_2\}$ is located to be in the bucket region *IEQR* after 3 scalar comparisons: x_1 with *EF*, x_2 with *IJ*, and x_1 with *QR*. Similarly, the path corresponding to a vector located in bucket region *LKHF* will consist of a comparison sequence with the hyperplanes *EF*, *GH*, and *KL*.

III. Friedman-Bentley-Finkel (FBF) Optimization and Backtracking Search Procedure

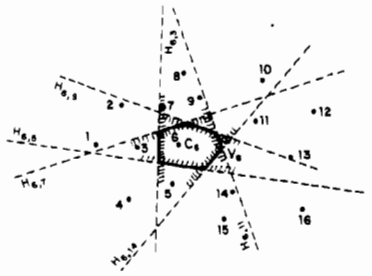
Here we briefly describe the optimization and backtracking search procedure proposed by Friedman *et al.* [6]. The prescription for the choice of the partitioning hyperplane at every nonterminal node of the tree considers the codevectors lying within the region represented by the node to be partitioned and constitutes a local optimization. The direct application of the FBF optimization for the case of vector quantization encoding will be as follows: i) the axis along which the corresponding codevector components have the maximum variance is chosen as the discriminant axis and ii) the median of the corresponding codevector component distribution on the chosen axis is chosen as the partition value. The variance and median are computed using codevectors within the region to be divided. The optimization results in a balanced binary tree, with each bucket containing equal number of codevectors.

The general nature of the search consists in first finding a tentative (current) nearest neighbor of the given test vector $\boldsymbol{x}$ from among the small set of codevectors within the bucket containing $\boldsymbol{x}$ and then in determining the actual nearest neighbor from among other buckets which overlap with the current nearest neighbor ball. The overall search is carried out by a recursive procedure which implicitly performs a backtracking to move from one overlapping bucket to another, the overlap being detected by a bounds-overlap-ball test. The termination is checked by a ball-within-bounds test at the root of every subtree that has been examined completely, i.e., each of the buckets within this region has either been searched, or ignored after verifying that the nearest neighbor ball does not overlap with it.

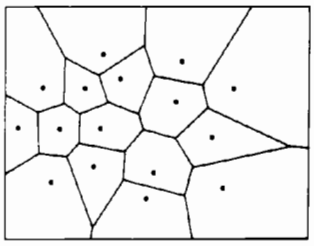
The backtracking procedure has a high computational overhead, dominated by the bounds-overlap-ball test, which is essentially a vector distance computation. Under this search strategy, the expected search time minimization was formulated in the form of minimizing the average number of buckets which overlap with the current nearest neighbor ball. The exact analysis of such an optimization formulation was intrinsically difficult due to the limitation that the search was of a backtracking nature, in addition to the restriction that the test vector distribution is unknown. This rendered the performance analysis obtuse and indirect. Moreover, the analysis did not directly yield the main prescriptions for the optimal division of a region and these were provided by means of qualitative considerations.

IV. Bucket-Voronoi Intersection Framework Using K-d Trees

The implicit geometric interpretation of the nearest neighbor search in terms of the Voronoi regions of a given set of codevectors is an important paradigm for structure based fast search. Given a set of N codevectors along with a specified distance measure, the entire space is partitioned into N disjoint regions, known as Voronoi regions, with each codevector associated with one region. The Voronoi region V_j associated with a codevector $\boldsymbol{c}_j$ contains all points in $\Re^K$ nearer to $\boldsymbol{c}_j$ than any other codevector and is the nearest neighbor locus region of $\boldsymbol{c}_j$. The Voronoi region V_j is defined as $V_j = \{\boldsymbol{x} \in \Re^K: \boldsymbol{q}(\boldsymbol{x}) = \boldsymbol{c}_j\}$ or $V_j = \{\boldsymbol{x} \in \Re^K: d(\boldsymbol{x}, \boldsymbol{c}_j) \leq d(\boldsymbol{x}, \boldsymbol{c}_i),\ i = 1, \cdots, N\}$. The Voronoi region V_j is a convex region formed by the intersection of the halfspaces $\{H(\boldsymbol{c}_j, \boldsymbol{c}_i),\ i = 1, \cdots, N,\ j \neq i\}$, given by $V_j = \cap_{j \neq i} H(\boldsymbol{c}_j, \boldsymbol{c}_i)$, where $H(\boldsymbol{c}_j, \boldsymbol{c}_i)$ is the set of points closer to $\boldsymbol{c}_j$ than $\boldsymbol{c}_i$. For the Euclidean distance, $H(\boldsymbol{c}_j, \boldsymbol{c}_i)$ is the half-space containing $\boldsymbol{c}_j$ formed by the perpendicular bisector plane of $\overline{\boldsymbol{c}_j \boldsymbol{c}_i}$, the line connecting $\boldsymbol{c}_j$ and $\boldsymbol{c}_i$. Fig. 2(a) illustrates the idea of the Voronoi region associated with a point in the plane ($\Re^2$) for the Euclidean distance and Fig. 2(b) shows the Voronoi partition for the given set of points.



(a)



(b)

Fig. 2. Example of Voronoi partition. (a) Illustrates the idea of the Voronoi region associated with a point on the plane ($\Re^2$) for the Euclidean distance. (b) The Voronoi partition for the given set of points.

Thus if the test vector $\boldsymbol{x}$ is contained in a Voronoi region $\boldsymbol{V}_j$, the associated codevector $\boldsymbol{c}_j$ will be the nearest neighbor of $\boldsymbol{x}$. In order to determine the nearest neighbor of the test vector it is thus sufficient to determine the Voronoi region containing it, with the associated codevector being its nearest neighbor. Direct approaches in determining the region containing $\boldsymbol{x}$ are complex and do not provide any computational saving over the distance-based exhaustive full search. The problem of fast identification of the Voronoi region containing a test vector in multidimensional space thus becomes important. An attempt made by Cheng *et al.* [14] to apply this approach using Voronoi projections showed considerable promise, but required large storage and high computation and memory access overheads.

In this context, the K-d tree has a very good space localization property, allowing very fast identification of the test vector in localized regions with very low overheads. The buckets and the Voronoi regions provide two independent partitionings of the same space using disjoint regions. Under the condition that the test vector is contained within the bounds of a particular bucket region, the test vector can be present only in one of the Voronoi regions having a nonempty intersection with the bucket. A search among the codevectors associated with these Voronoi regions is sufficient to determine the actual nearest neighbor. As the size of the bucket becomes smaller, the number of Voronoi regions which intersect with the bucket also reduces, thus lessening the search complexity considerably. Fig. 3 illustrates the bucket-Voronoi intersections obtained by the intersection of the K-d tree partitioning shown in Fig. 1 and the Voronoi partitioning shown in Fig. 2. For a test vector $\boldsymbol{x}$ located in the bucket *IEQR*, it is sufficient to search among the codevectors

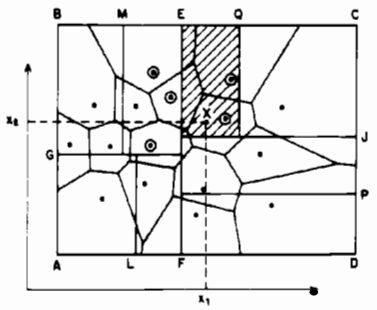


Fig. 3. Bucket-Voronoi intersections corresponding to example.

(marked in circles) whose Voronoi regions intersect with the region *IEQR*. These codevectors are associated with the bucket *IEQR* and for test vectors in this bucket, the search cost is reduced from a full-search cost of 16 to 5, subsequent to just 3 scalar comparisons. It can be observed that the complexity can be reduced further by increasing the tree depth to reduce the bucket sizes with a consequent reduction in the number of Voronoi regions intersecting with a bucket. The location of the hyperplanes is the most important issue here as this determines the extent of Voronoi intersections with the buckets formed by the tree. This forms the main issue of the optimization of the tree under the bucket-Voronoi intersection framework and is discussed in the following sections.

V. Optimization of the K-d Tree for Bucket-Voronoi Intersection Search

For a K-d tree of depth d, the root region is partitioned into $M(=2^d)$ disjoint rectangular regions, $\{B_i,\ i = 1, \cdots, M\}$. Let $p(\boldsymbol{x})$ be the joint probability density of $\boldsymbol{x}$ in $\Re^K$. Let $p_i = p(\boldsymbol{x} \in B_i)$. Let n_i be the number of codevectors whose corresponding Voronoi regions have a nonempty intersection with B_i. Thus, given $\boldsymbol{x} \in B_i$, a search among these n_i codevectors yields the nearest neighbor of $\boldsymbol{x}$; the cost of search, then, given $\boldsymbol{x} \in B_i$, is n_i. The average cost of search for bucket B_i is $p_i n_i$. The expected cost of search over all $\boldsymbol{x}$ is $E = \sum_{i=1}^{M} p_i n_i$. Minimization of E corresponds to a global optimization problem involving the variables p_i and n_i, $i = 1, \cdots, M$, which are functions of the partition choice at every nonterminal node in the tree. For a tree of depth d, the global optimization thus involves a joint optimization of the $(2^d - 1)$ variable pairs (j, h). The axis choice j has an integer range of 1 to K and the partition value h has a continuous real range within the bounds of the region to be divided, which, in turn, are determined by the partition choice at the nodes above it in the tree. The global optimization problem is thus extremely complex and, therefore, it becomes necessary to employ means for constructing a near optimal (in the sense of minimum expected search time) tree. One such approach is to minimize the expected search at every node in an independent way and this results in a local optimization of the individual binary divisions at every node. In the local optimization of the K-

d tree, the search complexity is minimized as much as possible at every nonterminal node division of the tree. We consider the local optimization problem in detail in the following.

A. Optimization with Test Data Distribution—Exact Optimization Criterion (EOC)

Let R be a bounded region in $\Re^K$, defined as $R = \{x \in \Re^K : a_j \le x_j \le b_j, j = 1, \cdots, K\}$, to be divided into two subregions R_L and R_R by a hyperplane normal to one of the coordinate axis. For some such hyperplane $\{x : x_j = h; a_j \le h \le b_j\}$ (represented as (j, h) henceforth), the subregions are defined as $R_L(j, h) = \{x \in R : x_j \le h\}$ and $R_R(j, h) = \{x \in R : x_j > h\}$. This is illustrated in Fig. 4. Conditioned by $x \in R$, let $p_L = p(x \in R_L)/p(x \in R)$ and $p_R = p(x \in R_R)/p(x \in R)$ with $p_L + p_R = 1$. Let n, n_L, and n_R be the number of Voronoi regions having a nonempty intersection with R, R_L, and R_R, respectively. p_L, p_R, n_L, and n_R are functions of (j, h) with $n_L \le n$ and $n_R \le n$ and $n \le n_L + n_R \le 2n$. For $x \in R$, given $x \in R_L$ or $x \in R_R$, the search complexity is reduced from n to n_L or n_R. Given $x \in R$, the determination of whether $x \in R_L$ or $x \in R_R$ requires only one comparison of the form $x_j \le h$, the division thus resulting in a complexity reduction after just one scalar comparison. The expected search complexity for $x \in R$, given only $x \in R$ is $E(R) = n$ and the expected search complexity using the division (j, h) is

$$E(R, j, h) = p_L(j, h)n_L(j, h) + p_R(j, h)n_R(j, h).$$

For an ideal division (j, h) such that $p_L = p_R = 1/2$ and $n_L = n_R = n/2$ we get $E(R, j, h) = n/2$ and the expected search time using the division will be thus half of that needed for a search without the division. The ideal division considered above is practically unlikely as the inevitable splitting of Voronoi cells rules out the possibility of $n_L = n_R = n/2$. Moreover, p_L, n_L, and n_R are functions of h (for any j) independently and it is highly unlikely for them to obtain these values for the same h. Therefore, it is necessary to consider these functions jointly in determining the optimal division (j, h) and the optimal choice of the partitioning hyperplane $(j, h)^*$ can be chosen as the one which minimizes $E(R, j, h)$ over all possible $(j, h): 1 \le j \le K$ and $a_j \le h \le b_j$. This criterion, which we refer to here as the exact optimization criterion (EOC), is then given by

$$(j, h)^* = \arg \min_{\substack{1 \le j \le K \\ a_j \le h \le b_j}} E(R, j, h).$$

The functional dependence of p_L, n_L, and n_R on (j, h) is completely determined by the distribution of the test vectors and the Voronoi intersection regions within the region of interest. The above optimization has to be carried out by an exhaustive search over all possible (j, h) using the functions $p_L(j, h)$, $n_L(j, h)$, and $n_R(j, h)$ precomputed for the region. It is thus necessary to find p_L, n_L, and n_R for any (j, h). $p_L(j, h)$ is obtained as

$$p_L(j, h) = p(x \in R : x_j < h)$$

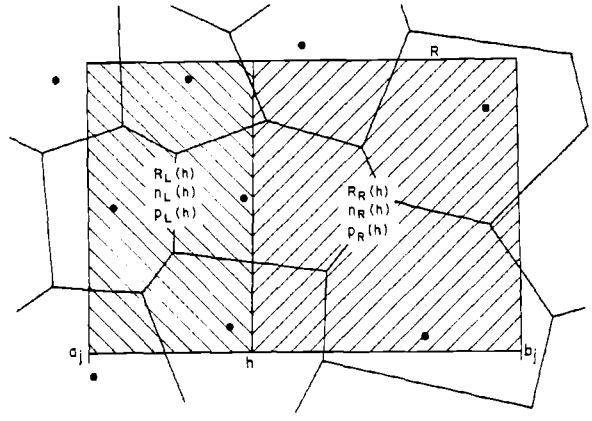


Fig. 4. Typical binary partitioning of a region at a nonterminal node.

from a large training data set, by first forming separate histograms for each of the component axis using the data belonging to the region R. For any nonterminal node the test vectors lying within the region to be divided are found by using the tree built so far above the node. n_L and n_R can be determined for any value of (j, h) using the projections of the Voronoi intersection regions on the coordinate axes. For the n Voronoi intersections having a nonempty intersection with the region R, their n projections on to a coordinate axis j correspond to n overlapping intervals $(P_1^j, P_2^j, \cdots, P_n^j)$ (Fig. 5). The lower and upper boundary points $(P_{i,L}^j, P_{i,U}^j)$ of the projection interval P_i^j of the Voronoi intersection region V_i inside the region R are given by

$$P_{i,L}^j = \min_{x \in V_i} x_j = \min_{x \in R} x_j : q(x) = c_i$$

and

$$P_{i,U}^j = \max_{x \in V_i} x_j = \max_{x \in R} x_j : q(x) = c_i.$$

For any h, $n_L(h)$ and $n_R(h)$ could be computed using these projection boundaries: $n_L(h)$ is the number of lower boundaries less than h and $n_R(h)$ is the number of upper boundaries greater than h. The projection estimates for each region to be partitioned can be obtained either analytically or through a Monte Carlo approach by encoding a large amount of training data or uniformly distributed points falling within the region as adopted in Cheng *et al.* [14] or Cheng and Gersho [7]. The projection estimates are obtained for each region to be divided by encoding the points that fall within the region. The estimates so obtained will be approximate, but have been found to be quite acceptable when obtained using sufficiently large data. The process of histogram formation and projection estimation can be combined together, thus requiring just one pass through the training data being used to carry out the optimal division at each node.

B. Optimization Without Test Data Distribution—Generalized Optimization Criterion (GOC)

Under the condition that the test data distribution $p(x)$ is not known, it is not possible to minimize the expected

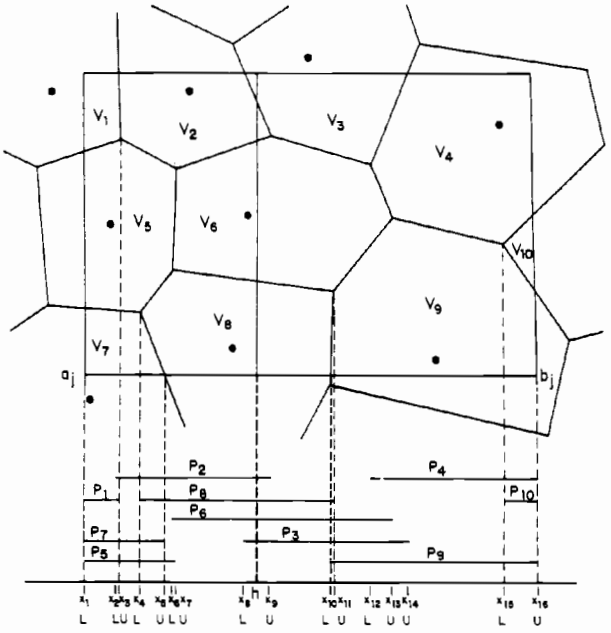


Fig. 5. Voronoi intersection regions and their projections (dashed lines show projection of region V_i to interval P_i; only some projection lines are shown to avoid cluttering).

search complexity using EOC described above. The Voronoi intersection numbers $n_L(h)$ and $n_R(h)$ are the only quantities that could be used in the optimization, and the objective of a good criterion would be to find a partition such that both n_L and n_R are as small as possible. The problem of obtaining a good criterion which represents the net complexity reduction due to a division in this case requires careful consideration due to the inevitable splitting of Voronoi regions for any division. In this section, we consider the division problem by characterizing the behavior of $n_L(h)$, $n_R(h)$ as a function of h, as h varies from a_j to b_j. Then, using a geometric interpretation of this in the n_L-n_R plane we obtain an efficient optimization criterion which uses only $n_L(h)$ and $n_R(h)$.

Given that n Voronoi regions have a nonempty intersection with the region R, their n projections on to a coordinate axis correspond to n overlapping intervals (P_1, P_2, $\cdots$, P_n) (Fig. 6(a)). These are equivalent to $2n$ projection boundaries (x_1, x_2, $\cdots$, x_{2n}) with a natural ordering ($x_1 < x_2 < \cdots < x_{2n}$). Each of these boundaries could be labeled as a lower (L) or upper (U) boundary according to whether it corresponds to the lower or upper edge of a Voronoi projection. We assume these boundaries to be distinct. These boundaries then divide the (a_j, b_j) range into ($2n - 1$) continguous intervals (I_1, I_2, $\cdots$, I_{2n-1}).

The behavior of $n_L(h)$ and $n_R(h)$ as h varies from a_j to b_j is as follows (Fig. 6(b)): At $h = a_j$, $n_L(h) = 0$ and $n_R(h) = n$; at $h = a_j + \epsilon$, $n_L(h) = 1$ and $n_R(h) = n$. As h varies across an interval boundary x_i, if x_i is a lower boundary (L), a new Voronoi region gets included in the left region $R_L(h)$ and $n_L(h)$ steps up by 1; there is no

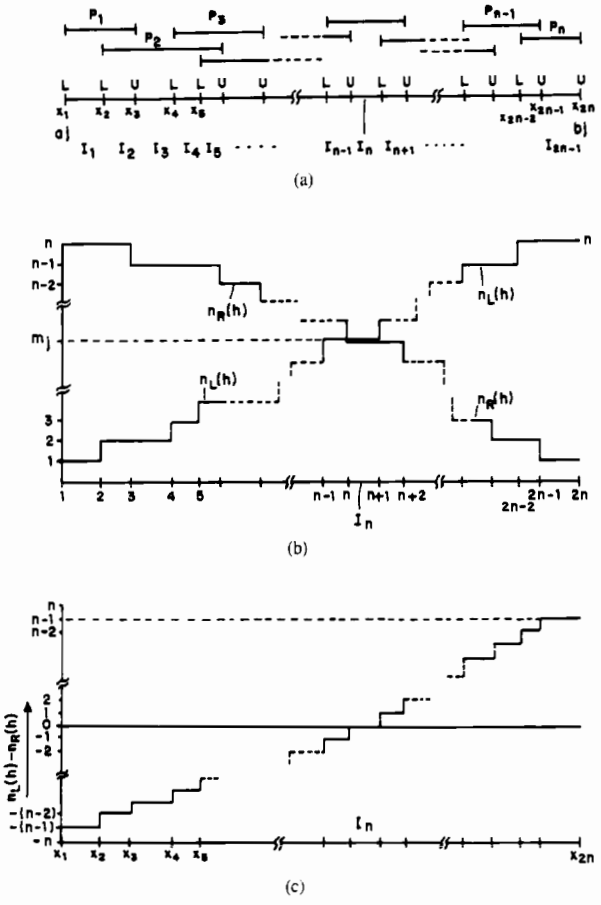


Fig. 6. Behavior of n_L, n_R and $(n_L - n_R)$ are a function of partition location h.

change in the status of the number of Voronoi regions in the right region $R_R(h)$ and hence $n_R(h)$ does not change. If x_i is an upper boundary (U), a Voronoi region gets excluded from the right region $R_R(h)$ and $n_R(h)$ steps down by 1; $n_L(h)$ does not change. When h varies between two adjacent interval boundaries (x_i, x_{i+1}) (i.e., within an interval I_i), there is no change in the status of the number of Voronoi intersections in both the left and right regions and hence $n_L(h)$ and $n_R(h)$ remain constant. At $h = b_j - \epsilon$, $n_L(h) = n$ and $n_R(h) = 1$; at $h = b_j$, $n_L(h) = n$ and $n_R(h) = 0$.

The function $(n_L(h) - n_R(h))$ behaves as follows as h varies from a_j to b_j (Fig. 6(c)): At $h = a_j + \epsilon$, $(n_L(h) - n_R(h)) = -(n - 1)$ and at $h = b_j - \epsilon$, $(n_L(h) - n_R(h)) = (n - 1)$. As h varies across a lower boundary, $n_L(h)$ increases by 1 while $n_R(h)$ remains constant and hence $(n_L(h) - n_R(h))$ increases by 1. As h varies across an upper boundary, $n_R(h)$ decreases by 1 while $n_L(h)$ remains constant and hence $(n_L(h) - n_R(h))$ increases by 1 again. Thus when h crosses an interval boundary, irrespective of whether it is a lower or upper boundary, $(n_L(h) - n_R(h))$ increases by 1. Within an interval, as $n_L(h)$ and $n_R(h)$ both remain constant, $(n_L(h) - n_R(h))$ also remains constant.

Thus, starting from $-(n-1)$ at I_1, $(n_L(h) - n_R(h))$ increases in steps of 1 as h crosses every interval boundary, finally reaching a value of $(n-1)$ at I_{2n-1}. At any interval I_l, $(n_L(h) - n_R(h)) = -(n-1) + (l-1)$. At $l = n$, i.e., at the interval I_n, $(n_L(h) - n_R(h)) = 0$ and let $n_L(h) = n_R(h) = m_j$. Thus for any $h \in I_n$, the division corresponds to a balanced division with equal number of Voronoi regions intersecting with both the left and right regions. Moreover, for any $h \in I_n$, the expected search complexity $E(h) = m_j$ as can be seen by using $n_L(h) = n_R(h) = m_j$ in $E(h) = p(h)n_L(h) + (1 - p(h))n_R(h)$. (Since no information about $p(h)$ has been used, this value of $E(h)$ is not the minimum possible on the axis j; this might lie at some other location of h.) It can also be noted that for the division $h \in I_n$, since $n_L(h) = n_R(h) = m_j$, the worst case complexity corresponding to the division is also m_j.

Thus, given that a balanced division with $n_L(h) = n_R(h) = m_j$ can always be achieved on any axis j at the interval I_n, and that the expected and worst case complexity for this division is given by m_j, a good optimization criterion for the final partition would be to choose the axis which has the minimum m_j over all $j = 1, \cdots, K$. We now obtain a general refinement of this criterion using a geometric interpretation of the division problem in the n_L-n_R plane using the functional behavior of $n_L(h)$ and $n_R(h)$ described above. This also provides additional insight into the optimization problem.

Considering $n_L(h)$ and $n_R(h)$ as parametric in h, $(n_L(h), n_R(h))$ for any h can be represented as a point in the n_L-n_R plane. The region of interest is the n_L-n_R plane where such points could lie is first obtained: Clearly, $0 \le n_L \le n$ and $0 \le n_R \le n$. Now, let $N_L(h)$ and $N_R(h)$ be the number of Voronoi regions entirely (i.e., not split by the division at h) in the left and right regions, respectively. Let $N_S(h)$ be the number of Voronoi regions split by the division at h. Henceforth, for convenience, we shall refer to $n_L(h)$, $n_R(h)$, $N_L(h)$, $N_R(h)$, and $N_S(h)$ simply as n_L, n_R, N_L, N_R, and N_S, ignoring the explicit reference to the functional dependence on h unless necessary and useful. Clearly, $N_L + N_R + N_S = n$ for all h, and $n_L = N_L + N_S$ and $n_R = N_R + N_S$. $N_L + N_R + N_S = n$ is the same as $n_L + n_R = n + N_S$ and with $0 \le N_S \le n$, $n_L + n_R$ is bounded as $n \le n_L + n_R \le 2n$. $n_L + n_R = n$ corresponds to all divisions for which no Voronoi regions are split, i.e., $N_S = 0$; and $n_L + n_R = 2n$ corresponds to the worst case division when all the n Voronoi regions are split, i.e., $N_S = n$. The resulting region for (n_L, n_R) points is thus given by $0 \le n_L \le n$, $0 \le n_R \le n$ and $n \le n_L + n_R \le 2n$. This is shown as the region ABC in Fig. 7.

The behavior of the point $(n_L(h), n_R(h))$ as h varies from a_j to b_j can now be obtained. As h varies within an interval I_i, $n_L(h)$ and $n_R(h)$ remain constant and correspondingly, the (n_L, n_R) point remains as a point in the n_L-n_R plane. When h varies across an interval boundary, x_i, i.e., moves from one interval (I_i) to the next (I_{i+1}), (n_L, n_R) changes to $(n_L + 1, n_R)$ if x_i is a lower boundary or to $(n_L, n_R - 1)$ if x_i is an upper boundary. As h varies from a_j to b_j,

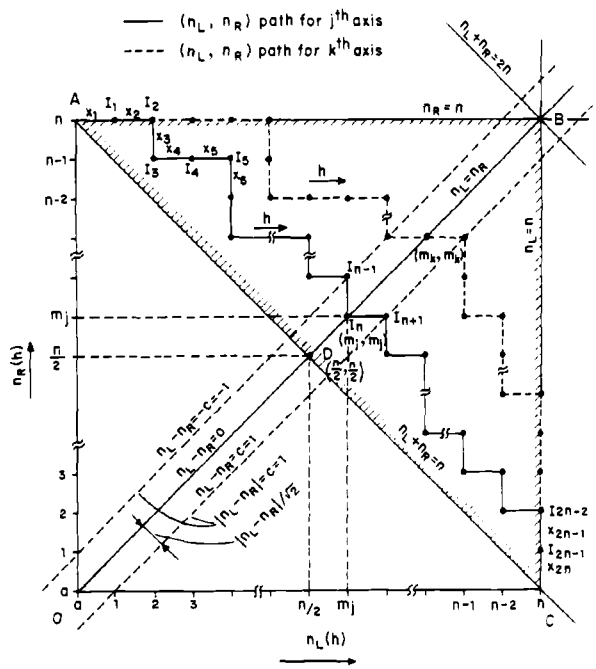


Fig. 7. Geometric interpretation of the proposed GOC in the n_L-n_R plane.

$(n_L(h), n_R(h))$ starts from the $(0, n)$ point and traverses a path which consists of changes of step 1 in directions parallel to the coordinate axis n_L or n_R as h varies across the interval boundaries, finally reaching the $(n, 0)$ point when h reaches b_j. This is illustrated in Fig. 7 where we show the path of (n_L, n_R) for the example shown in Fig. 6. This is essentially a plot of $n_L(h)$ versus $n_R(h)$ from Fig. 6(b). Corresponding to the interval I_n, where, $n_L = n_R = m_j$, the (n_L, n_R) path intersects the $n_L = n_R$ line OB at (m_j, m_j). In the interval I_n, i.e., $x_n \le h \le x_{n+1}$, the (n_L, n_R) path stays at the value of (m_j, m_j). For distinct interval boundaries, the (n_L, n_R) path for a given coordinate axis j is completely determined by the boundary label sequence on that axis. The example path illustrated in Fig. 7 for the axis j, corresponds to the label sequence ($LLULLUU \ldots LULU \ldots LULUU$). For some other axis k, the general behavior of $n_L(h)$ and $n_R(h)$ will be as described earlier, but the specific path of (n_L, n_R) will be now different since the label sequence of the $2n$ interval boundaries will be different. A typical example of the (n_L, n_R) path for another axis k is also shown in Fig. 7. This intersects the $n_L = n_R$ line at (m_k, m_k) and the optimization criterion mentioned earlier will choose the balanced division on the jth axis in preference to the kth axis division since $m_j < m_k$.

In general, the optimization criterion mentioned above can be described geometrically as follows: The $n_L = n_R$ line within the region ABC represents all possible balanced divisions starting from the ideal $(n/2, n/2)$ division to the worst case division of (n, n). The K balanced divisions (m_j, m_j) of the region from the K coordinate axis $j = 1, \cdots, K$ lie on this $n_L = n_R$ line ordered in some manner between $(n/2, n/2)$ and (n, n) along the line.

Since any point (n_L, n_R) lies on the line $n_L + n_R = n + N_S$ which runs parallel to the $n_L + n_R = n$ line at a distance of $N_S/\sqrt{2}$, the ordering of the balanced division on the $n_L = n_R$ line indicates the number of Voronoi regions split (N_S) by that division. The final partition chosen as the balanced division (m_j, m_j) with smallest m_j, is the lowest in the ordering. This is closest to the ideal $(n/2, n/2)$ point along the $n_L = n_R$ line and corresponds to a balanced division with the least number of Voronoi regions split.

A more general form of this optimization criterion is obtained by considering the constant contour lines of the function $|n_L - n_R|$, which are of the form $|n_L - n_R| = c$. These are a pair of lines $n_L - n_R = c$ and $n_L - n_R = -c$ which run parallel to the $n_L = n_R$ line at a distance of $|n_L - n_R|/\sqrt{2}$. For small values of c, the $|n_L - n_R| = c$ lines lie close to the $n_L = n_R$ line. (For $c = 0$, this is just the $n_L = n_R$ line.) In choosing the optimal partition on a given axis, we could then use the condition of min $|n_L - n_R|$ instead of $n_L - n_R = 0$. This is a more general criterion which encompasses the $n_L - n_R = 0$ condition while retaining the basic idea that, from the (n_L, n_R) path for a given axis, the division which is closest to the $n_L = n_R$ line should be chosen. This generalization allows for good unbalanced divisions to be chosen whenever balanced divisions are not possible, a situation which arises in practice when interval boundaries are not distinct.

Under this criterion for choosing the partitions on each of the coordinate axes, the criterion for choosing the final partition from the K candidate partitions has to be modified to handle possible unbalanced divisions which lie in a thin region around the $n_L = n_R$ line. For this, the Euclidean distance between a candidate division (n_L, n_R) and the $(n/2, n/2)$ point in the n_L-n_R plane can be used to locate the division closest to the ideal $(n/2, n/2)$ division. Based on these considerations, we finally get the following dual constraint which we refer to as the generalized optimization criterion (GOC):

1) For each coordinate axis $j = 1, \cdots, K$, the optimal partition h_j^*, for that axis is chosen as the one with minimum $|n_L - n_R|$.

2) From the K candidate partitions (j, h_j^*), $j = 1, \cdots, K$ so obtained, the final partition (k, h_k^*) for the region is chosen as the one whose corresponding (n_L, n_R) division is closest to the $(n/2, n/2)$ point, in the minimum Euclidean distance sense, where n is the number of Voronoi regions intersecting with the region to be partitioned.

In order to carry out the above optimization, n_L and n_R are determined for any value of h using the projections of the Voronoi intersections within the region to be partitioned as described in Section V-A for the exact optimization criterion.

At the end of the optimization using EOC or GOC, each bucket has a codevector list (henceforth referred to as bucket-Voronoi intersection list or BVI list) associated with it corresponding to the Voronoi regions intersecting with the bucket. The nearest neighbor search for any given test vector proceeds in two phases: first, identifying the bucket containing the test vector and, then, searching within the list of codevectors associated with the bucket. The resulting solution will be optimal if the intersection list is obtained correctly. The tree depth can be increased indefinitely, with the search complexity decreasing monotonically, limited only by the storage requirements. The worst case complexity is merely the largest bucket size, which decreases with increasing tree depth, bounded in the limit by the maximum number of Voronoi regions which have a vertex in common. For a tree of depth d, the total storage is $2(2^d - 1) + (\bar{b} + 1)2^d$ or simply $(3 + \bar{b})2^d$, where $\bar{b}$ is the average bucket size. An increase in depth by one results in just one extra scalar comparison in bucket identification and a doubling of storage. The high storage requirements do not pose a major problem, since only $O(d)$ actual memory access is needed during the search.

The complexity of the nearest neighbor search can be reduced by using a partial distance search [16] within the bucket. Further reduction results by using an ordered list [17] in each bucket where the codevectors are arranged in the order of decreasing probability of being the nearest neighbor for a test vector belonging to the bucket. This information can be obtained if the test data distribution is available as in the case of optimization using EOC.

VI. The Maximum Product Criterion (MPC)

Here we discuss the maximum product criterion used in [7] and [8] for the optimization of the K-d tree using only the Voronoi intersection information under the bucket-Voronoi intersection search. The BHT (binary hyperplane testing) algorithm in [7] is essentially a K-d tree structure where the partitioning hyperplanes are general $(K - 1)$ dimensional hyperplanes with arbitrary orientation and are not constrained to be orthogonal to the coordinate axes as in the case of the standard K-d tree. In this structure, $K + 1$ coefficients are needed to represent the general $K - 1$ dimensional hyperplane at each node and hence requires considerably higher storage than the standard K-d tree. Locating the test vector with respect to the partitioning hyperplane has to be done by a dot product with the hyperplane coefficients and this requires K multiplies and one comparison at each node, as opposed to just one scalar comparison with the use of the standard K-d tree. The optimization problem also is rendered more difficult by the use of general hyperplanes with arbitrary orientation. The algorithm is reported to have complexity of the order of $(r - 1)K + 2^K$ and the 2^K bound limits the usefulness of the algorithm only for bit rates $r > 1$. In [8] the standard K-d tree structure is used to realize a more efficient structure than that used in the BHT algorithm. However, the general tree structure used in both these works are the same and the optimization procedure for the design of the tree chooses the partitioning hyperplane as the one which maximizes the product $N_L N_R$ over all the K coordinate axes, N_L and N_R being the number of Voronoi

regions entirely on the left and right regions, respectively. Here, we give a geometric interpretation of this maximum product optimization criterion (MPC) in the n_L-n_R plane and compare it with the GOC criterion proposed here.

For any division h, $0 \le N_L \le n$ and $0 \le N_R \le n$. Moreover, since $N_L + N_R + N_S = n$, we have $N_L + N_R = n - N_S$ and thereby, $0 \le N_L + N_R \le n$. Thus, for any division h, (N_L, N_R) will lie in the region given by $0 \le N_L \le n$, $0 \le N_R \le n$ and $0 \le N_L + N_R \le n$ which is shown as region AOC in Fig. 8. Since the actual search complexity is determined by n_L and n_R and not by N_L and N_R, we wish to interpret the MPC in the n_L-n_R plane. For this purpose, the n_L-n_R region described earlier is shown in Fig. 8 as the region ABC along with the N_L-N_R region for ease of interpretation. For any given point $E(N_L, N_R)$ in the N_L-N_R region (AOC), the corresponding $F(n_L, n_R)$ point in the n_L-n_R region (ABC) is obtained by using the relation $n_L = n - N_R$ and $n_R = n - N_L$. (This follows from the fact that $n_L = N_L + N_S$, $n_R = N_R + N_S$ and $N_L + N_R + N_S = n$.) The points E and F lie on the $N_L + N_R = n - N_S$ and $n_L + n_R = n + N_S$ lines, respectively, which run parallel to the $N_L + N_R = n$ (same as the $n_L + n_R = n$) line at the distance of $N_S/\sqrt{2}$ as shown in Fig. 8. Therefore, given $E(N_L, N_R)$, the corresponding point $F(n_L, n_R)$ lies at a distance of $\sqrt{2}N_S$ along the line containing E and parallel to the $n_L = n_R$ line. Thus, for any given $E(N_L, N_R)$ in the (N_L, N_R) region the corresponding point $F(n_L, n_R)$ in the (n_L, n_R) region is simply a reflection of E about the $n_L + n_R = n$ line.

We interpret the MPC using the constant value contours of the function $N_L N_R$. The $N_L N_R = C$ contours for $C = C_1$, C_2, and C_3 with $C_3 > C_2 > C_1$ are shown in Fig. 8. The corresponding contours in the n_L-n_R plane are obtained by reflecting these $N_L N_R = C$ contours about the $n_L + n_R = n$ line. These are also shown in Fig. 8. The MPC chooses points lying on the $N_L N_R$ curves with the maximum C. For instance, MPC would choose the division corresponding to $N_L N_R = C_3$ as the optimal partition in preference to other divisions which lie on $N_L N_R = C_1$ and $N_L N_R = C_2$. It is easily seen that the point which has the absolute maximum of $N_L N_R = n^2/4$ is the $(n/2, n/2)$ point denoted by D. This ideal division is unlikely due to the inevitable splitting of the Voronoi regions, and hence the next best choice would be to choose balanced divisions with the least split, i.e., minimum N_S value. Since the distance of any point to the point D along the $n_L = n_R$ line is proportional to N_S, such "balanced least split" divisions would correspond to a point closest to D on the $n_L = n_R$ line and will have a value of $N_L N_R = C$ with C less than the maximum $n^2/4$ value. However, the function $N_L N_R = C$ has a large spread about the $n_L = n_R$ line, even for small reductions in C and due to this even for a good division along the $n_L = n_R$ line with a small N_S, several other highly unbalanced divisions (far away from the $n_L = n_R$ line) with the same $N_L N_R$ value would compete for selection as the optimal division. Consequently, the MPC might choose divisions which are far from the $n_L = n_R$ line in preference over points which are closer to the $n_L = n_R$ line and this will result in highly unbalanced divisions with poor complexity reductions since one of the subregions will have a large number of Voronoi regions.

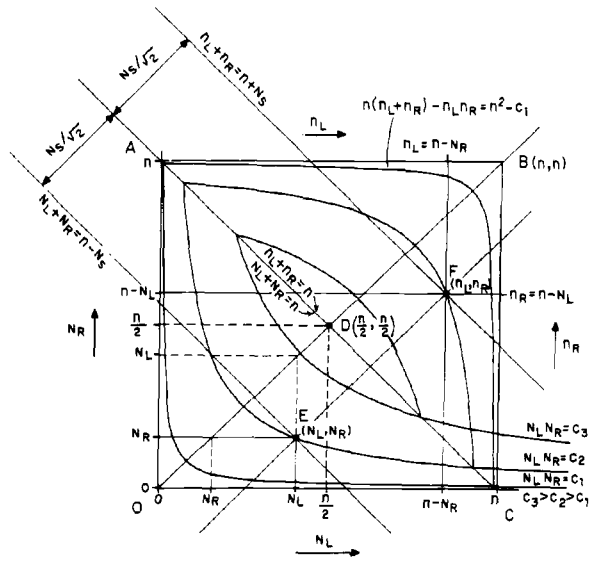


Fig. 8. Geometric interpretation of the maximum product criterion.

In contrast to MPC, the optimization criterion GOC will have better complexity reduction as it seeks to construct the most balanced tree with the lowest expected search complexity, by first explicitly choosing divisions as close to the $n_L = n_R$ line as possible; and from each coordinate axis, subsequently choosing the division which is closest to the ideal $(n/2, n/2)$ point.

VII. Experiments and Results

In this paper, we study the complexity reduction performance of the proposed optimization criteria EOC and GOC in the context of vector quantization using speech data. Here we present the performance results of the various optimization criteria EOC, GOC, and MPC developed specifically under the bucket-Voronoi intersection (BVI) search. We also show the performance of the optimization proposed by Friedman *et al.* with the backtracking search as in the original algorithm [6] as well as with bucket-Voronoi intersection search.

First, we compare the performance of the following algorithms: i) EOC—the exact optimization criterion with BVI search, ii) GOC—the generalized optimization criterion with BVI search, iii) MPC—the maximum product criterion with BVI search [7], [8], iv) FBF-BCK—the Friedman-Bentley-Finkel (FBF) algorithm with its backtracking search [6], and v) FBF-BVI—the FBF optimization used for bucket-Voronoi intersection search. The complexity is measured in terms of the number of multiplications per sample which is also the effective codebook size or the number of distance calculations per vector for a squared error distance. For BVI search, the complexity of search for a test vector is simply the full-search cost

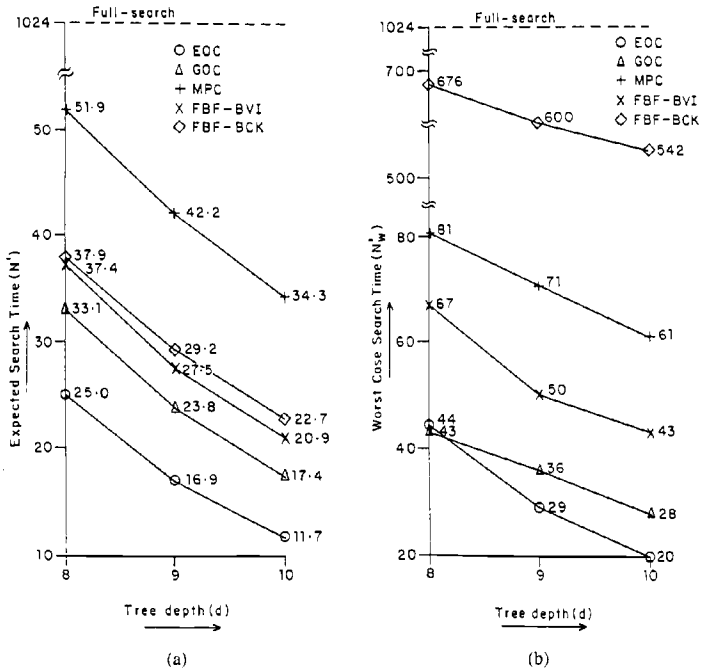


Fig. 9. Comparison of different K-d tree optimization criterion and search procedures. (a) N' versus d. (b) N'_W versus d. Dimension $K = 8$, codebook size $N = 1024$, data used: 50 s of speech (50 000 vectors outside of the design data).

for searching within the codevectors in the BVI list of the bucket containing the test vector. The average complexity is obtained as an average over a large number of test vectors. For the FBF-BCK algorithm with backtracking search, the complexity is the number of codevectors checked. The overall complexity of FBF-BCK is actually much higher, if the effective number of bounds-overlap-ball tests (which is essentially a distance computation) made during the backtracking search is also considered.

In Figs. 9 (a) and (b), the expected search time (N') and worst case search time (N'_W) performances of these five algorithms (EOC, GOC, MPC, FBF-BCK, and FBF-BVI) are shown for vector dimension $K = 8$ and codebook size $N = 1024$ for tree depths 8, 9, and 10. Here the K-d tree was optimized using 150 000 vectors (150 s of speech from multiple speakers sampled at 8 kHz) and the results shown are obtained for 50 000 vectors of data (different from the data used for optimization). The codebook was generated from 200 s of data using the LBG algorithm [15] (splitting procedure) under the squared error criterion.

It can be observed that EOC offers the best complexity reduction as it uses the test data distribution and has the best expected search minimization possible. It sets the performance limit achievable and can be used here as a reference for the performances of the other optimizations which do not use the test data distribution. GOC performs very close to EOC. The MPC-based tree performs poorly in comparison to GOC. The very poor worst case performance of the FBF optimization with backtracking search can be clearly observed. In comparison, the proposed optimizations with the BVI search have an excellent expected and worst case performance, both of them showing a monotonic decrease with increase in tree depth. The FBF-BVI, where the partitioning obtained by FBF optimization on the codevectors is used in a bucket-Voronoi intersection search performs quite efficiently and has a complexity close to that of GOC, which handles the Voronoi information explicitly. It even performs better than the MPC. The good performance of FBF optimization under the bucket-Voronoi intersection search, despite not being optimized using the Voronoi information explicitly, can possibly be attributed to the fact that the partitioning generated by this optimization is such that each bucket encloses an equal number of codevectors. The corresponding Voronoi regions will be located centrally and prominently within the bucket, along with only a small number of intersections from adjacent Voronoi regions. This can result in uniform bucket sizes and, consequently, in the good efficiency of FBF optimization under BVI search.

In Table I, we show the complexity reduction efficiency of the proposed optimization criteria EOC and GOC under BVI search for tree depth $d = \log N$ in terms of the average complexity (N'), average complexity with partial-distance search [16] (N'-PD) and worst case complexity (N'_W) using codebooks of size $N = 32$, 64, 128, 256, 512, and 1024 and vector dimension $K = 8$. For each case, the K-d tree was optimized using 150 000 vectors. The table shows N', N'-PD, and N'_W for 50 000 vectors of data (different from the data used for optimization) and N' for the 150 000 vectors of data used for optimi-

TABLE I
PERFORMANCE OF PROPOSED OPTIMIZATION CRITERIA EOC AND GOC FOR TREE DEPTH d = LOG N. DIMENSION K = 8; CODEBOOK SIZE N = 32, 64, 128, 256, 512, 1024

		EOC				GOC			
		Data I			Data II	Data I			Data II
N	d	N'	N'-PD	N'_W	N'	N'	N'-PD	N'_W	N'
32	5	7.3	3.6	12	7.0	9.4	4.1	12	9.3
64	6	8.3	4.2	19	7.8	11.7	5.0	15	11.7
128	7	10.5	5.1	21	10.2	14.0	5.8	17	14.1
256	8	11.1	5.6	22	10.5	16.6	7.3	21	16.8
512	9	12.2	6.4	26	11.8	18.4	7.9	26	19.0
1024	10	11.7	6.4	20	11.5	17.4	7.9	28	18.1

Data I—Outside design data: 50 000 vectors of speech.
Data II—Inside design data: 150 000 vectors of speech.

TABLE II
PERFORMANCE OF PROPOSED OPTIMIZATION CRITERIA EOC AND GOC FOR DIMENSIONS K = 2, 4, 6, 8, 10. CODEBOOK SIZE N = 1024; TREE DEPTH d = log N = 10

	EOC				GOC			
	Data I			Data II	Data I			Data II
K	N'	N'-PD	N'_W	N'	N'	N'-PD	N'_W	N'
2	2.6	2.0	19	2.5	3.7	3.0	12	3.7
4	8.0	5.2	16	7.9	11.6	6.9	17	12.1
6	11.7	6.5	21	11.9	16.1	8.0	26	16.9
8	11.7	6.4	20	11.5	17.4	7.9	28	18.1
10	12.4	6.7	23	12.5	18.3	8.3	29	19.1

Data I—Outside design data: 50 000 vectors of speech.
Data II—Inside design data: 150 000 vectors of speech.

TABLE III
COMPARISON OF PROPOSED OPTIMIZATION CRITERIA EOC AND GOC FOR TREE DEPTH d = LOG N WITH FULL-SEARCH. DIMENSION K = 8; CODEBOOK SIZE N = 1024. DATA: 50 000 VECTORS OF SPEECH.

	N'	N'_W	$\bar{b}$	Total Storage	m	a	c
EOC	11.7	20	11.3	22834	11.7	21.9	2.6
GOC	17.4	28	12.1	23695	17.4	32.6	3.3
Full-search	1024	1024	—	8192	1024	1920	127.9

zation. Here it can be seen that both EOC and GOC can achieve a constant expected and worst case complexity for tree depths of O (log N). In addition, the average complexity for both the inside- and outside-optimization data are very nearly the same, indicating the sufficiency of optimization and the consistently in the performance of the algorithms. The worst case complexity is identical for inside- and outside-optimization data.

In Table II, we show the performance of the proposed criteria with respect to dimensionality. Here, the performance efficiency is shown for a codebook of size N = 1024 and vector dimensions K = 2, 4, 6, 8, and 10 with tree depth of d = log N = 10 for each dimension. For each case, the K-d tree was optimized using 150 000 vectors. The table shows N', N'-PD, and N'_W for 50 000 vectors of data (different from the data used for optimization) and N' for the 150 000 vectors of data used for optimization. Here, it can be seen that both EOC and GOC achieve constant expected and worst case complexity for tree depths of O (log N) over the various dimensions. The sufficiency of optimization and performance consistency of the algorithms over the various dimensions is again indicated by the almost identical average complexity for the inside- and outside-optimization data.

In Table III, we show the performance of the proposed EOC and GOC with respect to full search for K = 8 and N = 1024 in terms of average, worst case complexity, and storage. The results are obtained for 50 000 vectors of speech data. The K-d tree algorithm using BVI search under the EOC and GOC optimizations can be seen to offer excellent complexity reduction over the full search with very low average and worst case complexity though at the cost of increased storage. The total storage required by these algorithms for a tree of depth d, is about $(3 + \bar{b})2^d$, where $\bar{b}$ is the average bucket size. The average number of multiplications, additions (subtractions), and comparisons per sample ((macs) measure [14]) are also shown. The full search has a (macs) complexity of $[N, N(2K - 1)/K, (N - 1)/K]$ and the BVI search has a complexity $[N', N'(2K - 1)/K, (N' - 1 + d)/K]$, where N' is the average number of codevectors examined by the BVI search procedure and the additional d comparisons are incurred for locating the bucket containing the test vector. The significant reduction in the overall complexity offered by the BVI search using EOC and GOC optimizations over full search can be noted from this.

In Fig. 10, we show the effect of the amount of data used for bucket-Voronoi intersection list generation for a tree optimized by EOC. Given the codevectors and an optimized tree, the BVI list can in principle be generated by determining the Voronoi regions intersecting with each bucket analytically. Here, as described earlier, the BVI list is generated by encoding a large set of training (or, design) data. If the data used for BVI list generation is small, it can be insufficient for detecting all the Voronoi intersections within each bucket. Consequently, the exclusion of codevectors whose Voronoi intersections actually intersect with the buckets will result in encoding errors when the tree is used for encoding new test data using the BVI search. This in turn will reflect as lower signal-to-noise ratio (SNR) in comparison to the full-search SNR. At the same time, since the bucket sizes will be quite small, the average complexity during search will be very low. The sufficiency of data used for the BVI list generation can be established by increasing the data used for BVI list generation until the SNR performance of the BVI search on the test data set (different from the design data set) is the same as or admissibly close to its full-search SNR, and when the average and worst complexity of search (or, alternately, simply the average bucket size

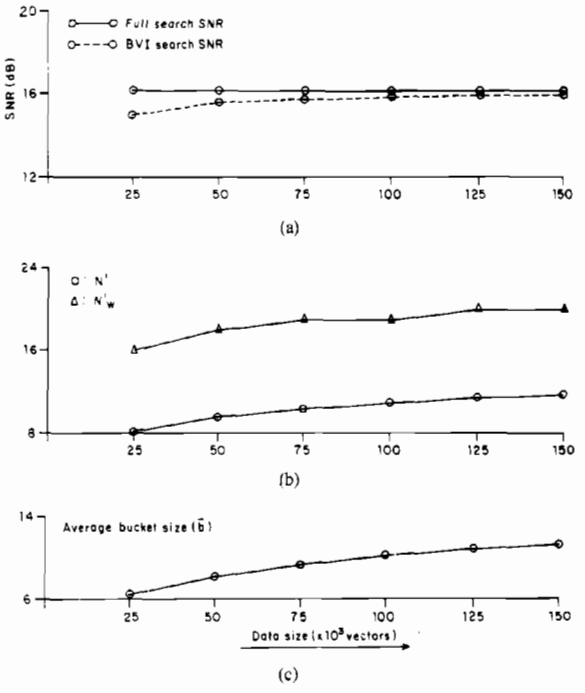


Fig. 10. Sufficiency of data for BVI list generation. (a) SNR, (b) average and worst case complexity, and (c) average bucket size for 50 000 vectors outside of the design data as a function of number of inside-design vectors used in BVI list generation for EOC. Dimension $K = 8$, codebook size $N = 1024$, tree depth $d = 10$.

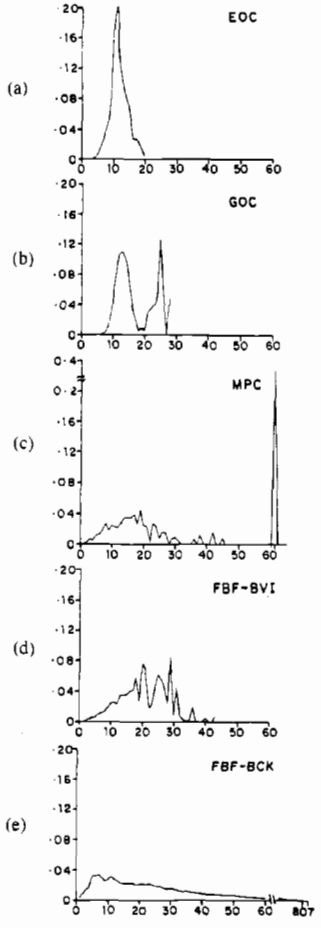


Fig. 11. Histograms of number of codevectors searched for (a) EOC, (b) GOC, (c) MPC, (d) FBF-BVI, and (e) FBF-BCK algorithms. Dimension $K = 8$, codebook size $N = 1024$, tree depth $d = 10$. Data used: 50 000 vectors outside of the design data.

of the tree) shows no appreciable increase. We show this for the experiments reported here with $K = 8$ and $N = 1024$ using upto 150 000 vectors of data for BVI list generation. The data used for generating the BVI list was increased from 25 000 vectors to 150 000 vectors in steps of 25 000 vectors. We show the SNR, average complexity (N'), worst case complexity (N'_W) for 50 000 vectors outside the design data. Also shown is the average bucket size ($\bar{b}$) in the tree for each data size. For small data sizes (25 000 vectors) the SNR difference can be seen to be quite high (about 1.17 dB) which reduces to a very small difference of about 0.2 dB when 150 000 vectors are used. The average, worst case complexity and average bucket size can all be seen to register a gradual increase until they saturate showing the sufficiency of 150 000 vectors for BVI list generation.

In Fig. 11, we show the histograms of number of codevectors searched for the BVI search under the EOC, GOC, MPC, and FBF optimizations and the backtracking search (FBF-BCK) under the FBF optimization. The histograms were obtained for $K = 8$ and $N = 1024$ with a tree depth of $d = \log N = 10$ using 50 000 vectors outside of the design data. The histogram gives the probability that a given number of codevectors N' will be searched for a given test vector. In general, this histogram reveals the characteristics of the search efficiency in the following manner: The maximum of the range of N' over which the histogram extends is the worst case complexity and a spread of the histogram over a large range of N' indicates high worst case complexity behavior of the search. More importantly, the location of high probability modes in the histogram corresponds to the dominant values of N' the search is likely to have, and mainly determines the average complexity behavior of the search. Here, it can be seen that EOC has the narrowest histogram with one high-probability mode located at a small N' value. The GOC has a slightly larger spread and has modes of lower probability located at a relatively higher N'. However, it can be noted that the GOC histogram shows a significant mode very close to that of EOC attributing to its close overall performance to EOC. In comparison, MPC has a very large spread with a very high probability mode at the maximum (worst case) number of codevectors. FBF-BVI also has large spreads but does not suffer from any high probability mode at higher N' values. The backtracking search FBF-BCK has a good average complexity behavior with high probabilities at the lower end of N' with the histogram tapering down uniformly to zero till its maximum

N'. However, the spread of the histogram is very high due to the high worst case complexity of the backtracking search. For FBF-BCK here, the histogram is shown for the overall complexity including the effective bounds-overlap-ball tests made (which is essentially a distance computation as mentioned earlier); this makes the actual overall worst case complexity of the search significantly higher (807) than that shown in Fig. 9 (542) for only the number of codevectors examined.

So far, we have discussed the performance of different optimization criteria for the design of the K-d tree and shown that the EOC and GOC criteria offer an efficient and improved complexity reduction for nearest neighbor search. We have not yet talked about the complexity of the optimization procedures for these criteria. Though it is not a very important issue as it is done only during the design phase, some comments about it are in order here. The main computational step in the GOC optimization procedure at each node is the determination of the Voronoi intersections within the region to be partitioned and their revised projection estimates. This can be done with a small amount of computational cost provided the nearest neighbors of all the vectors used in the optimization process are precomputed and stored. The EOC optimization procedure requires, in addition to the projection estimates, the conditional histograms from the training data for getting the $p(h)$ values needed to find the minimum expected search complexity $E(h)$ on each coordinate axis. Since the projection estimates and histograms can be obtained in the same pass through the data used for optimization, the EOC optimization procedure is only slightly more complex than the GOC optimization procedure. The FBF optimization procedure is the least complex as it does not require any projection estimates and probability computations. It directly deals with the codevectors and requires only the computation of variance and median of the coordinates of codevectors inside the region to be partitioned. However, it will be of interest to find means for making the EOC and GOC optimization procedures faster and less complex while retaining their efficient performance under the bucket-Voronoi intersection search.

VIII. Conclusions

In this paper, we have addressed the issue of fast nearest neighbor search is multidimensional space using the K-d tree data structure with particular emphasis on the optimization of the tree under a bucket-Voronoi intersection framework for fast vector quantization. We have presented two efficient optimization criterion and procedures for designing the K-d tree—one for the case when the test data distribution is available, as in vector quantization applications in the form of training data, and the other when the test data distribution is not available and only the Voronoi intersection information is to be used. The proposed algorithm is studied in the context of vector quantization encoding of speech and is empirically observed to achieve constant expected search complexity for O (log N) tree depths. The bucket-Voronoi intersection search framework and the proposed optimizations are shown to be more efficient than the backtracking search proposed earlier by Friedman *et al.* The proposed optimization criteria are also shown to be more efficient than the maximum product criterion (MPC) used recently. Under the framework used for obtaining the proposed optimization criterion, we have given a geometric interpretation of the MPC with reasons for its inefficiency.

References

[1] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, vol. 1, pp. 4–29, Apr. 1984.

[2] A. Gersho and V. Cuperman, "Vector quantization: A pattern matching technique for speech coding," *IEEE Commun. Mag.*, vol. 21, pp. 15–21, Dec. 1983.

[3] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, pp. 1555–1588, Nov. 1985.

[4] K. K. Paliwal and V. Ramasubramanian, "Vector quantization in speech coding: A review," *Indian. J. Technol.*, vol. 24, pp. 613–621, Oct. 1986.

[5] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. Ass. Comput. Mach.*, vol. 18, no. 9, pp. 509–517, Sept. 1975.

[6] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209–226, Sept. 1977.

[7] D. Y. Cheng and A. Gersho, "A fast codebook search algorithm for nearest neighbor pattern matching," in *Proc. IEEE ICASSP '86*, 1986, pp. 265–268.

[8] A. Lowry, S.Q.A.M.A. Hossain, and W. Millar, "Binary search trees for vector quantization," in *Proc. IEEE ICASSP '87* 1987, pp. 2205–2208.

[9] V. Ramasubramanian and K. K. Paliwal, "An optimized K-d tree algorithm for fast vector quantization of speech," in *Signal Processing IV: Theories and Applications*, J. L. Lacoume *et al.*, Eds. North-Holland, 1988, pp. 875–878; also in *Proc. EUSIPCO '88* (Grenoble, France), Sept. 1988.

[10] V. Ramasubramanian and K. K. Paliwal, "A generalized optimization of the K-d tree for fast nearest neighbor search," in *Proc. 4th IEEE Region 10 Int. Conf. TENCON'89*, Nov. 1989, pp. 565–568.

[11] W. H. Equitz, "Fast algorithms for vector quantization picture coding," master's thesis, Mass. Inst. Technol., June 1984.

[12] W. H. Equitz, "Fast algorithms for vector quantization picture coding," in *Proc. IEEE ICASSP'87*, 1987, pp. 725–728.

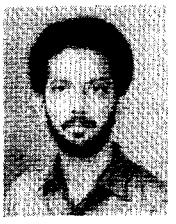
[13] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 10, pp. 1568–1575, Oct. 1989.

[14] D. Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *Proc. IEEE ICASSP '84*, vol. 1, Mar. 1984, pp. 9.11.1–9.11.4.

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

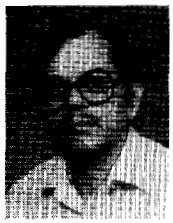
[16] C. D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. Commun.*, vol. COM-33, pp. 1132–1133, Oct. 1985.

[17] K. K. Paliwal and V. Ramasubramanian, "Effect of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantization," *IEEE Trans. Commun.*, vol. COM-37, pp. 538–540, May 1989.



V. Ramasubramanian was born in Coimbatore, India, on March 24, 1962. He received the B.Sc. degree in applied science from the PSG College of Technology, Coimbatore, India, in 1981 and the B.E. degree in electronics and communications engineering from the Indian Institute of Science, Bangalore, India, in 1984. Currently, he is working toward the Ph.D. degree at the Tata Institute of Fundamental Research, Bombay. His research activities have been mainly in the area of speech coding using vector quantization and fast

algorithms for nearest neighbor search. His main research interests are in the areas of speech recognition, speech coding, and neural networks for pattern recognition.



Kuldip K. Paliwal (M'89) was born in Aligarh, India, in 1952. He received the B.S. degree from Agra University, India, in 1969, the M.S. degree from Aligarh University, India, in 1971, and the Ph.D. degree from Bombay University, India, in 1978.

Since August 1972, he has been with Tata Institute of Fundamental Research, Bombay, India, where he has worked on various aspects of speech processing: speech recognition, speech coding, and speech enhancement. From September 1982 to October 1984, he was an NTNF fellow at the Department of Electrical and Computer Engineering, Norwegian Institute of Technology, Trondheim, Norway. He was a Visiting Scientist at the Department of Communications and Neuroscience, University of Keele, U.K., from June to September 1982 and January to March 1984, and at the Electronics Research Laboratory (ELAB), Norwegian Institute of Technology, Trondheim, Norway, from April to July 1987, April to July 1988, and March to May 1989. Since May 1989, he has been working as a consultant at the Acoustics Research Department, AT&T Bell Laboratories, Murray Hill, NJ. His work has been concentrated on fast search algorithms for vector quantization, 24 b/frame vector quantization of linear predictive coding parameters, better feature analysis and distance measures for speech recognition, and robust spectral analysis techniques. His current research interests are directed towards automatic speech recognition using hidden Markov models and neural networks.