DIGITAL SIGNAL PROCESSING **7,** 260–277 (1997)
ARTICLE NO. SP970300

Voronoi Projection-Based Fast Nearest-Neighbor Search Algorithms: Box-Search and Mapping Table-Based Search Techniques

V. Ramasubramanian* and K. K. Paliwal†

*Computer Systems and Communications, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay—400 005, India; and †School of Microelectronic Engineering, Griffith University, Brisbane, Queensland 4111, Australia

Ramasubramanian, V., and Paliwal, K. K., Voronoi Projection-Based Fast Nearest-Neighbor Search Algorithms: Box-Search and Mapping Table-Based Search Techniques, *Digital Signal Processing* **7** (1997), 260–277.

In this paper we consider fast nearest-neighbor search techniques based on the projections of Voronoi regions. The Voronoi diagram of a given set of points provides an implicit geometric interpretation of nearest-neighbor search and serves as an important basis for several proximity search algorithms in computational geometry and in developing structure-based fast vector quantization techniques. The Voronoi projections provide an approximate characterization of the Voronoi regions with respect to their locus property of localizing the search to a small subset of codevectors. This can be viewed as a simplified and practically viable equivalent of point location using the Voronoi diagram while circumventing the complexity of the full Voronoi diagram. In this paper, we provide a comprehensive study of two fast search techniques using the Voronoi projections, namely, the box-search and mapping table-based search in the context of vector quantization encoding. We also propose and study the effect and advantage of using the principal component axes for data with high degree of correlation across their components, in reducing the complexity of the search based on Voronoi projections.

1. INTRODUCTION

1.1. Fast Nearest-Neighbor Search

The aim of nearest-neighbor search is to determine the closest point to a query point among N points in K-dimensional space. The problem of finding the nearest neighbor in multidimensional space arises in several areas such as pattern classification, nonparametric estimation, information retrieval from multikey data bases, and image and speech data compression using vector quantization. The computational complexity of nearest-neighbor search is a major problem in these areas, when the size N of the point set to be searched becomes very high. As a result, the problem of developing algorithms for fast nearest-neighbor search has attracted significant attention in these areas. In this paper we consider fast nearest-neighbor search in the context of vector quantization encoding.

Vector quantization is a powerful data compression technique used in speech coding, image coding, and speech recognition [1, 3, 18–20, 25, 28]. Vector quantization has the potential to achieve coding performance close to 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. Vector quantization encoding is the minimum-distortion quantization of a vector $\mathbf{x} = (x_1, \ldots, x_K)$ (referred to as the *test vector*), using a given set of N K-dimensional *codevectors* (called the *codebook* $\mathbf{C} = \{\mathbf{c}_i\}_{i=1,\ldots,N}$, of size N), under some distance measure $d(\mathbf{x}, \mathbf{y})$. This involves finding the nearest-neighbor of $\mathbf{x}$ in $\mathbf{C}$, given by $\mathbf{q}(\mathbf{x}) = \mathbf{c}_j : d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_i)$, $i = 1, \ldots, N$, which requires N vector distance computations $d(\mathbf{x}, \mathbf{c}_i)$ 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 or bits/coordinate) as $N = 2^{Kr}$ and the complexity of encoding $\mathbf{x}$ increases exponentially with K and r. The encoding complexity also constitutes the most computation intensive step in iterative vector

1051-2004/97 $25.00

quantizer design [27] and the problem of reducing the computational complexity of vector quantization encoding has received considerable attention for realizing the full potential of vector quantization and for rendering it practically useful for real-time applications.

The fast algorithms reported so far for fast nearest-neighbor search can be broadly categorized under two search constraint paradigms: (i) fast search based on elimination rules and (ii) Voronoi partition-based structured search.

1.1.1. Elimination-Based Fast Search

Elimination-based fast search exploits some properties of the distance measure used in the nearest-neighbor definition. Here, fast search is achieved by a "quick-elimination" or "throwaway" principle, where codevectors which cannot be nearer to the given test vector than the current nearest-neighbor are eliminated without incurring the cost of a distance computation. These algorithms typically employ efficient elimination schemes such as the partial distance elimination [4, 10, 31] elimination based on the hypercube [10, 42, 49] and absolute-error-inequality approximations of the Euclidean distance [41], and triangle-inequality based elimination [8, 9, 16, 17, 21–23, 30, 33–36, 39, 40, 44, 45].

1.1.2. Voronoi Structure-Based Fast Search

This paradigm is based on the Voronoi partition, which is a geometric construct describing the nearest-neighbor mapping of the input space for a given set of codevectors. The Voronoi partition permits the search to be viewed as a point location problem of determining which Voronoi region contains the test vector with the corresponding codevector being the desired nearest-neighbor. The Voronoi diagram serves as an important basis for several proximity search techniques in computational geometry ([15, 26, 32, 48] give an extensive overview of these problems).

The techniques based on the Voronoi diagram for various search problems is a classic example of a general problem solving paradigm called the "locus" method in computational geometry. In this method, the search space is partitioned into a number of *equivalence* classes—a set of nonoverlapping regions such that the answer is invariant for all query points that fall in the same region. This method is particularly applicable for search in the so-called "repetitive mode" where arbitrarily long sequence of queries are encountered. This is typically the situation in vector quantization or pattern classification applications, where it is required to encode or classify test (or feature) vectors arriving as long sequences in time using a given codebook or classifier.

Various solutions based on the Voronoi diagram have been reported in the computational geometry literature for two (planar) and higher dimensions. While several optimal solutions which are also practically realizable exist for the planar nearest-neighbor problem, all currently known solutions for large dimensions pose major practical difficulties in terms of high preprocessing, storage, and overhead costs in constructing and using the Voronoi diagram despite their excellent asymptotic search time results obtained theoretically. Consequently, emphasis in fast search rests on finding efficient geometric structures and space constraints that characterize the Voronoi partition to reduce the point location problem to a very small Voronoi region subset by computationally inexpensive operations.

1.1.3. Scope and Organization of the Paper

In this paper, we investigate fast nearest-neighbor search techniques based on the projections of the Voronoi regions. The projections of the Voronoi regions were proposed and used for fast search in an earlier work [10]. However, this basic paradigm has not received adequate attention with respect to the complexity of the higher dimensional Voronoi diagram and algorithms which use the full Voronoi diagram. The Voronoi projections provide an approximate characterization of the Voronoi regions with respect to their locus property in localizing the search to a small subset of codevectors; this results in a simplified and practically viable equivalent of point location which circumvents the complexity of the complete Voronoi diagram.

In this paper, our objective is to highlight the basic efficiency of using the Voronoi projections and to show that fast search based on the Voronoi projections, using appropriate data structures, can result in very practical and highly efficient fast search algorithms. In this respect, this paper provides a comprehensive study of two methods for using the Voronoi projections for fast search, namely, box-search and mapping table-based search. We consider their average and worst-case complexity performance, and in addition provide a detailed study of their overall performance which includes the average and worst-case overhead computations, storage and preprocessing cost. (A condensed version of this paper is reported elsewhere as a correspondence item [50].)

The paper is organized as follows. In Section 2, we describe the basic structure of Voronoi diagram-based fast search and the complexity of algorithms based on the full Voronoi diagram. In Section 3, we introduce the basic search structure based on the projections of the Voronoi regions emphasizing the

main computational step involved in finding the reduced subset of codevectors whose Voronoi projections contain the test vector. In Section 4, we describe a simple search scheme termed "box-search" (BS) which treats the projections directly as hypercuboid approximations of the Voronoi regions. In Section 5, we discuss in detail the projection method proposed by Cheng *et al.* [10] which was the first work to consider the use of the Voronoi projections. In Section 6, we propose and study the use of the projections on the principal component directions for reducing the complexity of search for the box-search and mapping table-based search procedures. In Section 7, we give simulation results comparing the performances of the box-search and the mapping table-based search in the context of vector quantization encoding of speech waveform.

2. FAST NEAREST-NEIGHBOR SEARCH USING THE VORONOI DIAGRAM

Given a set of N codevectors $\mathbf{C} = \{\mathbf{c}_i\}_{i=1,\ldots,N}$, of size N, along with a distance measure $d(\mathbf{x}, \mathbf{y})$, $\mathbf{x}, \mathbf{y} \in R^K$, the R^K 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 $\mathbf{c}_j$ contains all points in R^K nearer to $\mathbf{c}_j$ than any other codevector and is the nearest-neighbor locus region of $\mathbf{c}_j$. If the nearest-neighbor of $\mathbf{x}$ in $\mathbf{C}$ is $\mathbf{q}(\mathbf{x}) = \mathbf{c}_j : d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_i)$, $i = 1, \ldots, N$, the Voronoi region V_j is defined as $V_j = \{\mathbf{x} \in R^K : \mathbf{q}(\mathbf{x}) = \mathbf{c}_j\} = \{\mathbf{x} \in R^K : d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_i), i = 1, \ldots, N\}$.

The Voronoi region V_j is thus a convex region formed by the intersection of the half-spaces $\{H(\mathbf{c}_j, \mathbf{c}_i), i = 1, \ldots, N, j \neq i\}$, given by

$$V_j = \bigcap_{j \neq i} H(\mathbf{c}_j, \mathbf{c}_i),$$

where $H(\mathbf{c}_j, \mathbf{c}_i)$ is the set of points closer to $\mathbf{c}_j$ than $\mathbf{c}_i$, i.e.,

$$H(\mathbf{c}_j, \mathbf{c}_i) = \{\mathbf{x} \in R^K : d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_i)\}.$$

For the Euclidean distance, $H(\mathbf{c}_j, \mathbf{c}_i)$ is the half-space containing $\mathbf{c}_j$ formed by the perpendicular bisector plane of $\overline{\mathbf{c}_j\mathbf{c}_i}$, the line connecting $\mathbf{c}_j$ and $\mathbf{c}_i$. Figure 1a illustrates the idea of the Voronoi region associated with a point in the plane (R^2) for the Euclidean distance and Fig. 1b shows the Voronoi partition for the given set of points.

By the definition of the Voronoi diagram, given that a test vector $\mathbf{x}$ is contained in a Voronoi region $\mathbf{V}_j$, the associated codevector $\mathbf{c}_j$ will be the nearest-neighbor of $\mathbf{x}$. Fast nearest-neighbor search using

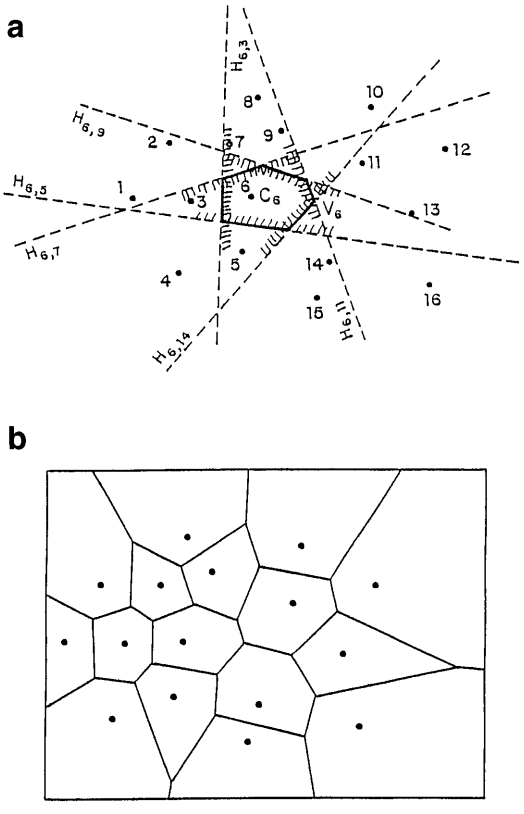

FIG. 1. Example of Voronoi partition.

the Voronoi diagram can therefore be viewed as a "locus method" which involves use of fast methods for searching the Voronoi subdivisions of the Voronoi diagram by means of a *point location* algorithm to determine the Voronoi region containing the test vector. Fast search using the Voronoi diagram thus consists of two steps:

1. *Constructing the Voronoi diagram:* A preprocessing phase of constructing the Voronoi diagram, representing it in the form of a graph data structure and associating with each facet (for instance, vertex, edge and region, in the 2-dimensional case) of the diagram, the set of closest codevectors.
2. *Point location:* Given the Voronoi subdivisions, determine the facet of the subdivision which contains the test vector and the associated codevector is the desired nearest-neighbor.

2.1. Complexity of Algorithms Based on Full Voronoi Diagram

Though the use of the Voronoi diagram provides an efficient paradigm for nearest-neighbor search to be carried out as a point location search for lower dimensions (such as for the 1-dimensional and planar cases), the construction and usage of the Voronoi

diagram for higher dimensions is beset with implementational difficulties, mainly due to the fact that for a given set of points, the number of items required to describe the Voronoi diagram grows exponentially with dimension.[1] Thus, the main complexity of algorithms based on the full Voronoi diagram arises from the preprocessing stage in constructing the Voronoi diagram of N input points which has an $\Omega(N^{K/2})$ worst case time and storage, in addition to the cost required in organizing the Voronoi diagram for point location. While the basic computation of the Voronoi diagram invariably faces this intrinsic exponential complexity, the use of the diagram for fast nearest-neighbor search has further practical difficulties since the known search (point-location) algorithms which use the Voronoi diagram in higher-dimensional subdivisions have similar exponential dependence on dimension in their average and worst-case query time performance bounds and preprocessing and storage costs which grow exponentially with dimension [11–13, 47].

3. VORONOI PROJECTION-BASED FAST SEARCH

In this section, we consider in detail, search techniques based on the *projections* of the Voronoi regions on the coordinate axes. The Voronoi projections provide an approximate characterization of the Voronoi regions and allow for a simplified, but practically viable equivalent of point location while circumventing the complexity of the complete Voronoi diagram. Though the projections constitute only a very partial information about the Voronoi regions they retain their locus property of localizing the search to a small subset of codevectors and we see from empirical results for large dimensions and typical codebook sizes that the Voronoi projection-based search has an excellent complexity reduction potential.

Let V_i be the Voronoi region associated with the codevector $\mathbf{c}_i$. Then the projection of V_i on the coordinate axis j is a projection interval P_i^j, with lower and upper boundary points $(P_{i,L}^j, P_{i,U}^j)$ given by

$$P_{i,L}^j = \min_{\mathbf{x} \in V_i} x_j$$

and

$$P_{i,U}^j = \max_{\mathbf{x} \in V_i} x_j.$$

This represents the two hyperplanes $\mathbf{x} : x_j = P_{i,L}^j$ and $\mathbf{x} : x_j = P_{i,U}^j$, normal to the jth coordinate axis. The K projections $(P_{i,L}^j, P_{i,U}^j)$, $j = 1, \ldots, K$ define a set of $2K$ hyperplanes which bounds the Voronoi region V_i by the smallest hypercuboid region

$$B_i = \{\mathbf{x} : (P_{i,L}^j \leq x_j \leq P_{i,U}^j), \quad j = 1, \ldots, K\}.$$

Each codevector has such a hypercuboid approximation of the Voronoi region associated with it and we refer to this as the *Voronoi-box* (or "box") of the codevector. The hypercuboid box has sides parallel to the coordinate axes and is a simple geometric approximation of the Voronoi region.

3.1. Basic Structure of Search Using Voronoi Projections

Clearly, $V_i \subseteq B_i$ and hence, if $\mathbf{x} \in V_i$, then $\mathbf{x} \in B_i$. Conversely, if $\mathbf{x} \notin B_i$, then $\mathbf{x} \notin V_i$. Thus any codevector whose hypercuboid box does not contain the test vector cannot be the nearest-neighbor of the test vector since its Voronoi region does not contain the test vector. However, if $\mathbf{x} \in B_i$, then it indicates that the corresponding Voronoi region may contain $\mathbf{x}$. Since the boxes are not disjoint, several boxes can contain a test vector and all such codevectors need to be considered as candidates for the nearest-neighbor search. Identifying the boxes which contain the test vector results in generating a candidate set of codevectors, only one of which will correspond to the Voronoi region that actually contains the test vector. The actual nearest-neighbor can be determined by a full-search in this candidate set.

The fast search using the Voronoi projections can be viewed as consisting of two steps:

1. Determine $C'(\mathbf{x}) = \{\mathbf{c}_i : \mathbf{x} \in B_i\}$: the candidate set of codevectors whose boxes B_i contain the test vector $\mathbf{x}$.

[1] A Voronoi diagram in K-dimensions is a cell complex made up of convex polytopes which consist of faces of dimensions varying from 0 to K, such as vertex (0-dimension), edge (1-dimension), region (2-dimension), facet ($(K-1)$-dimension), and cell (K-dimension). In general, a face in the cell complex is called a k-face, $(0 \leq k \leq K)$, if it is of dimension k. Thus, vertex, edge, region, facet and cell are 0-face, 1-face, 2-face, $(K-1)$-face, and K-face, respectively. The Voronoi diagram, being a cell complex formed by hyperplane arrangements, is ideally represented and stored in the form of an incidence graph [15]. Here, each face of the diagram is associated with a node of the incidence graph and two nodes are connected if the corresponding faces are incident. In order to provide the actual spatial correspondence, each node in the graph corresponding to a cell stores the coordinates of the point corresponding to the cell. Klee [24] showed that the maximum number of faces grows exponentially in K. Tight upper bounds obtained for the number of k-faces of the Voronoi diagram of N-points is in $O(N^{\min\{K+1-k, K/2\}})$, for $0 \leq k \leq K$ [15, 38]. Thus, the Voronoi diagram of N points in 3-dimensions may have $O(N^2)$ edges [32] and for higher dimensions ($K > 3$), the maximum number of edges and vertices of the Voronoi diagram of N points is $O(N^{K/2})$. Algorithms which construct the Voronoi diagram of N points in K-dimensional space require $O(N^{(K+1)/2})$ time in the worst-case and $O(N^{K/2})$ storage [2, 5–7, 37, 46]. A detailed formal treatment of higher dimensional Voronoi diagram and its application to various proximity search problems can be found in [15, 32].

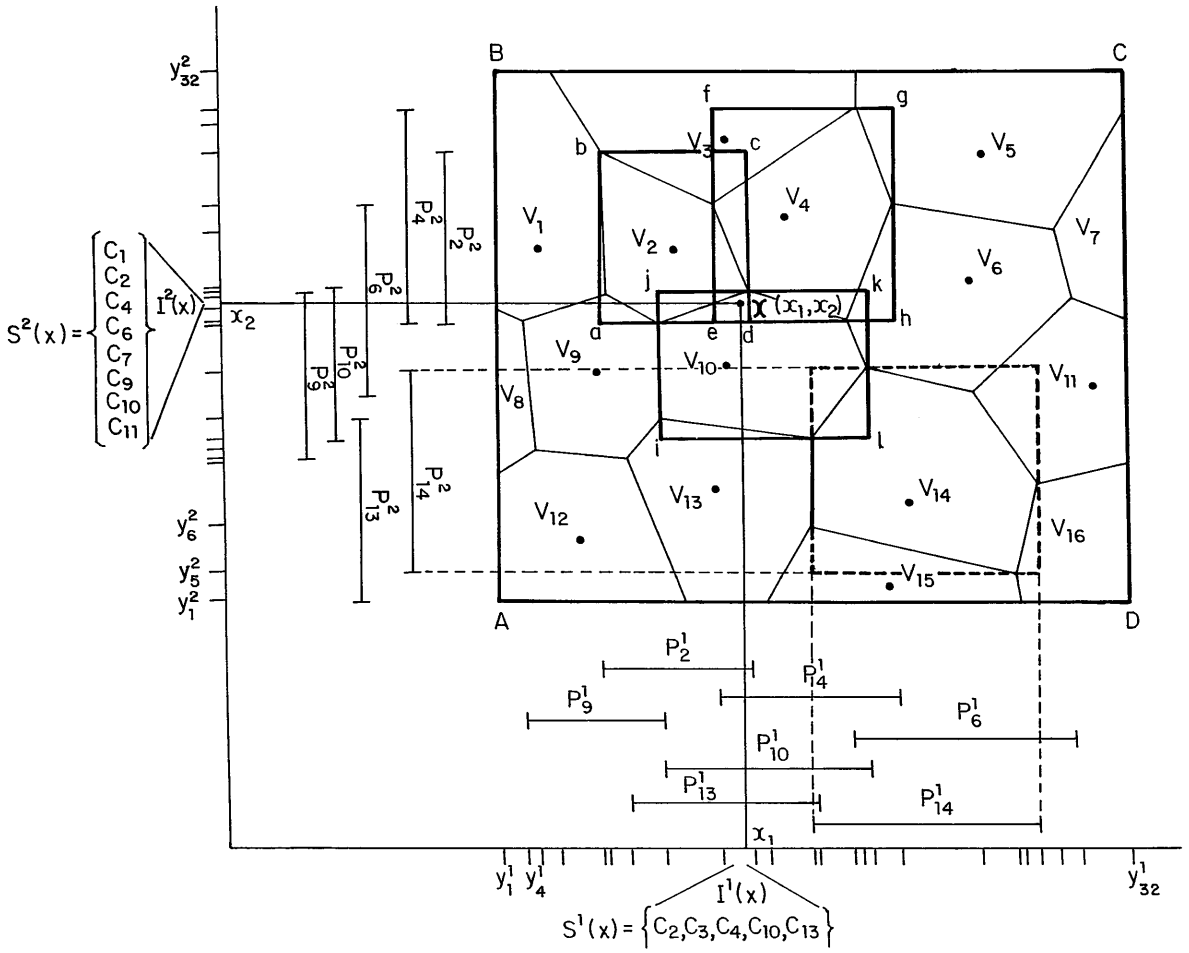

FIG. 2. Example of projection method.

2. Perform a full-search in the candidate set $C'(\mathbf{x})$ to obtain the actual nearest-neighbor of $\mathbf{x}$.

This approach can be viewed as a simplification of the problem of point location directly in the Voronoi region to a problem of point location within the smallest hypercuboid box enclosing the Voronoi region. However, point location within the boxes does not yield a unique solution for the Voronoi region containing the test vector, thus necessitating a search among the candidate set of codevectors generated by point location among the boxes. If the size of this candidate set of codevectors $C'(\mathbf{x})$ is small in comparison to the full codebook size N, the search will achieve significant complexity reduction.[2]

The basic structure of search using the Voronoi projections is illustrated by an example in 2-dimension in Fig. 2. This figure shows the Voronoi diagram of 16 points, V_i, $i = 1, \ldots, 16$ within a range of interest ABCD. The projections of a Voronoi region forming the box enclosing it are illustrated for V_{14}. Consider a test vector $\mathbf{x} = (x_1, x_2)$ ($\in V_{10}$). The test vector is located inside the boxes of V_2, V_4, and V_{10}, marked as *abcd*, *efgh*, and *ijkl*, respectively. These are the only boxes containing the test vector and hence form the final candidate set $C'(\mathbf{x})$, which has to be searched to find the actual nearest-neighbor $\mathbf{c}_{10}$ with a reduced complexity of 3 distance computations as against 16 for the full-search.

The main complexity of the fast search using the Voronoi projections, in terms of the number of distances computed, is the size of the candidate set C' to

[2] We note here that a related algorithm which is also based on the idea of using a simple geometric approximation of the Voronoi region was proposed in [43]. Here, the smallest hypersphere enclosing the Voronoi region and centered at the codevector is determined first for each of the codevectors. Each codevector is thus implicitly associated with the smallest hypercube containing this sphere. Given a test vector, only those codevectors whose hypercubes contain the test vector are considered for a full-search. However, such a hypercube is invariably larger than the bounding hypercuboid boxes considered here, and the spherical Voronoi approximation algorithm [43] therefore has intrinsically poorer complexity reduction efficiency than Voronoi projection-based search.

be searched in step 2. This is determined by the average number of boxes that contain a test vector for a given test data distribution. However, the total complexity of the algorithm is highly dependent on the cost of step 1 in determining the candidate set of codevectors for a given test vector, i.e., the complexity of finding which boxes contain the test vector. The main complexity of step 2 is determined only by the distribution of the codevectors and the test vectors and is invariant to the method employed to perform step 1. Step 1 essentially contributes to the overhead computation of the fast search and it is important to obtain efficient procedures for carrying out this step in order to reduce the overall complexity of the algorithm.

In the following sections, we consider in detail two search techniques for using the projection information to carry out step 1. These are namely, (i) box-search and (ii) mapping table search.

4. BOX-SEARCH

The hypercuboid box B_i is a simple geometric object for point location and the final set of codevectors can be determined by directly checking if the test vector lies within the box of each of the codevectors. This requires checking whether the jth component of the test vector is contained within the projection boundaries ($P^j_{i,L}$, $P^j_{i,U}$) of the Voronoi region V_i on the jth coordinate axis requiring two comparisons per coordinate. A codevector can be rejected when any of the test vector's coordinates falls outside the corresponding Voronoi projection, i.e., $x_j < P^j_{i,L}$ or $x_j > P^j_{i,U}$ for any j. If a test vector passes the test on all the coordinates (when all the projections of a Voronoi region contain the corresponding coordinates of the test vector), then the test vector lies inside the hypercuboid box approximation of the Voronoi region and the corresponding codevector has to be considered for full-search. This search is as follows:

Box-search
  $d_{min} = \infty$
  do $i = 1, \ldots, N$
    if $x\_in\_box(i)$ **then** Box-test
      d = d(x, c_i)
      if $d < d_{min}$ **then** $nn = i$; $d_{min} = d$
    endif
  enddo
  procedure $x\_in\_box(i)$
  do $j = 1, \ldots, K$
    if $x_j < P^j_{i,L}$ **then** $x\_in\_box(i) = .false.$; **return**
    if $x_j > P^j_{i,U}$ **then** $x\_in\_box(i)$ = .false.; **return**
  enddo
  $x\_in\_box(i) = .true.$
  return

4.1. Storage and Computational Overheads of Box-Search

The boxes are stored as a codevector indexed table; i.e., the $2K$ projection values are stored along with the corresponding codevector, requiring $2KN$ real words of additional memory to store the projection values of the N codevectors.

The search described above performs a simple elimination test (box-test) for each codevector in a full-search structure before computing its distance to the test vector $\mathbf{x}$. The distance between the test vector $\mathbf{x}$ and codevector $\mathbf{c}_i$ is computed only if the codevector is not rejected by the box-test $x\_in\_box(i)$ which checks whether $\mathbf{x} \in B_i$. A codevector can be rejected after c comparisons, where $1 \le c \le 2K$, and any codevector which passes the test for distance computation incurs a "box-test" cost of $2K$ comparisons. For a given test vector, if c_i is the number of comparisons for codevector $\mathbf{c}_i$, the total cost of box-search for the test vector is $c(\mathbf{x}) = \Sigma_{i=1}^{N} c_i$. The cost per vector component (or coordinate) is $c(\mathbf{x})/K$ (with bounds $N/K \le c(\mathbf{x}) \le 2N$) and the average of this over a large set of test vectors characterizes the actual overhead comparison cost for the box-search. The quantity $c(\mathbf{x})/N$, on the other hand, gives the average number of comparison tests done *per codevector*. This can have a value of 1 to $2K$ and when obtained as an average over a large set of test vectors indicates how quickly a codevector is eliminated on an average within the $2K$ comparisons required to reject a codevector in the worst case. These quantities characterize the actual overhead computations and the relative efficiency of box-search.

5. THE MAPPING TABLE-BASED SEARCH

In the projection method [10], the projections of the Voronoi regions are reorganized to form a "mapping table" in a preprocessing stage which is then used to generate the candidate set of codevectors for a given test vector. In this method, the space is partitioned into a rectangular cell partition using the projections. Given a test vector, a rectangular cell that contains the test vector is first identified using only scalar comparison operations on the structured projections. The candidate codevectors associated with the cell containing the test vector are then formed by intersection operations on precomputed tables corresponding to each coordinate axes. A full-search within this small set of candidate codevectors yields the nearest-neighbor of the test vector.

The N overlapping projection intervals $P_i^j = (P_{i,L}^j, P_{i,U}^j)$ $i = 1, \ldots, N$ generate $2N$ projection boundaries $(y_1^j, y_2^j, \ldots, y_{2N}^j)$ with a natural ordering $(y_1^j \leq y_2^j \leq \cdots \leq y_{2N}^j)$. This partitions the jth coordinate axis into $2N - 1$ contiguous intervals $\{I_1^j, I_2^j, \ldots, I_{2N-1}^j\}$, where $I_m^j = [(y_m^j, y_{m+1}^j)]$. Each of these intervals I_m^j is associated with a set, S_m^j, of indices of the Voronoi region whose projection intervals P_i^j partially or totally overlap with I_m^j, i.e., $S_m^j = \{i : P_{i,L}^j \leq y_m^j$ and $P_{i,U}^j \geq y_{m+1}^j\}$.

Given a test vector $\mathbf{x} = (x_1, \ldots, x_K)$, let $I_m^j = [(y_m^j, y_{m+1}^j)]$ be the interval containing x_j, i.e., $y_m^j \leq x_j \leq y_{m+1}^j$. Denoting this interval as $I^j(\mathbf{x})$, let $S^j(\mathbf{x})$ be the index set associated with this interval. Any test vector whose jth component x_j lies in the interval $I^j(\mathbf{x})$ can lie within any one of the Voronoi regions whose projection overlaps with $I^j(\mathbf{x})$. The set of indices $S^j(\mathbf{x})$ associated with $I^j(\mathbf{x})$ thus corresponds to the candidate codevectors which could be the nearest-neighbor of the given test vector $\mathbf{x}$. For a given test vector $\mathbf{x}$, each coordinate axis j generates such a set of candidate codevectors $S^j(\mathbf{x})$, whose Voronoi region may contain the test vector with respect to the jth coordinate axis. The Voronoi region actually containing the test vector should have all its K projection intervals contain the corresponding test vector component and the nearest-neighbor codevector index will be contained in the intersection of the K sets $S^j(\mathbf{x})$, $j = 1, \ldots, K$. This intersection set is the set of candidate indices which need to be examined by a full-search to obtain the actual nearest-neighbor.

The $(2N - 1)$ contiguous intervals $\{I_1^j, I_2^j, \ldots, I_{2N-1}^j\}$, on each of the K coordinate axes can be seen as partitioning the space into $(2N - 1)^K$ rectangular cells. Identification of the K intervals $I^j(\mathbf{x})$, $j = 1, \ldots, K$ containing the test vector locates the test vector in a hyperrectangular cell $H(\mathbf{x}) = \prod_{j=1}^K I^j(\mathbf{x})$. Since all scalar points in the interval $I^j(\mathbf{x})$ are associated with a common set of candidate indices $S^j(\mathbf{x})$, all vector points inside the cell $H(\mathbf{x})$ are also associated with a single set of final candidate codevectors given by the intersection of $S^j(\mathbf{x})$, $j = 1, \ldots, K$. (Note that the cell containing the test vector $H(\mathbf{x})$ is actually contained within the volume of intersection of the Voronoi boxes containing the test vector $\mathbf{x}$ and that the set of final candidate codevectors obtained as the intersection of $S^j(\mathbf{x})$, $j = 1, \ldots, K$ is identical to that obtained by box-search.)

This represents an organization of the projections into a large number ($(2N - 1)^K$) of disjoint rectangular cells, each of which have an invariant set of candidate codevectors which can be seen to be on the lines of the "locus" method described in Section 1.1.2. The set associated with each of the $(2N - 1)^K$ hyperrectangular cells can in principle be precomputed and stored for each cell directly. By this, a very small set of candidate codevectors will be available for full-search subsequent to K binary searches (each of cost log $(2N)$) to find the K intervals $I^j(\mathbf{x})$, $j = 1, \ldots, K$ containing the test vector projections. This, however, will be at the cost of an exhorbitant storage of $(2N - 1)^K\gamma$, where γ is the average size of the candidate set associated with the rectangular cells.

Alternately, this is obtained at lesser storage by computing the intersections of individual interval subsets $S^j(\mathbf{x})$ for every test vector $\mathbf{x}$. This, however, introduces the additional computational overhead cost in forming the intersection of K subsets.

We illustrate this search for the example shown in Fig. 2. This shows the Voronoi diagram of 16 points, V_i, $i = 1, \ldots, 16$, and their corresponding projections P_i^j, $i = 1, \ldots, N$, $j = 1, 2$ within a range of interest ABCD. The projections are shown explicitly only for some of the Voronoi regions to maintain clarity while the complete set of projection boundaries forming the contiguous intervals are shown on each axis. Given the test vector $\mathbf{x} = (x_1, x_2)$ $(\in V_{10})$, the intervals $I^1(\mathbf{x})$ and $I^2(\mathbf{x})$ containing x_1 and x_2 are identified first. These intervals are associated with the index sets $S^1(\mathbf{x}) = \{\mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_{10}, \mathbf{c}_{13}\}$ and $S^2(\mathbf{x}) = \{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_4, \mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_9, \mathbf{c}_{10}, \mathbf{c}_{11}\}$, respectively. The intersection of these two sets gives the final candidate set $\{\mathbf{c}_2, \mathbf{c}_4, \mathbf{c}_{10}\}$, which has to be searched for the actual nearest-neighbor $\mathbf{c}_{10}$.

5.1. Storage and Computational Overheads of Mapping Table Search

5.1.1. Organization, Preprocessing, and Storage Overheads

Cheng *et al.* [10] have used the following data structures to perform fast search using the *mapping table* at reduced storage complexity. A mapping table for coordinate axis j consists of the ordered boundary points $\{y_m^j\}_{m=1,\ldots,2N}$ describing the $2N - 1$ intervals $\{I_m^j\}_{m=1,\ldots,2N-1}$ and the associated index sets $\{S_m^j\}_{m=1,\ldots,2N-1}$. These are obtained in a preprocessing step for each coordinate axis and K such tables, one for each coordinate axis, comprise the complete mapping table. The mapping table entries can be organized in two ways: (i) index map (IM) and (ii) bit map (BM).

Index map. In the IM, the indices are stored as numerical values. If α_m^j is the size of the index set S_m^j, the average number of indices per interval for the jth axis is $\bar{\alpha}^j = [1/(2N - 1] \sum_{m=1}^{2N-1} \alpha_m^j$ and the overall average number of indices per interval for the entire mapping table is $\alpha = (1/K) \sum_{j=1}^K \bar{\alpha}^j$. The storage of the

mapping table in the index mode requires $2NK$ real words for the ordered projections, $(2N-1)\alpha K$ and $(2N-1)K$ integer words for the index sets and the index set size respectively. The total storage is then $(3+\alpha)NK$ real words. This is referred to as the *full table* (FT) organization in [10].

Cheng *et al.* also considered a *reduced table* (RT) organization which reduces the storage to $5KN$ words. This makes use of the fact that an interval boundary is either a lower or an upper projection of a Voronoi region, and consequently the index sets on adjacent intervals differ by only one element; i.e., if an interval boundary y_m is a lower projection $P^j_{i,L}$ of some Voronoi region V_i, then the index set S^j_m corresponding to the interval $I^j_m = [(y^j_m, y^j_{m+1})]$ is $S^j_m = S^j_{m-1} + \{i\}$ and if y_m is an upper projection $P^j_{i,U}$, then $S^j_m = S^j_{m-1} - \{i\}$. Therefore, only the differing element i has to be stored as a signed index $s_m = +i$ or $s_m = -i$ for each interval $I^j_m = [(y^j_m, y^j_{m+1})]$, $+/-$ representing y_m to be a lower/upper Voronoi projection. Then, the index set S^j_m of any interval I^j_m can be formed recursively starting from the index s_1 on the first interval, i.e., $S^j_m = S^j_{m-1} + \{s_m\}$ with $S^j_1 = \{s_1\}$. This form of storage, therefore, represents a significant reduction in storage in comparison to the full-table organization, since only one signed index is stored per interval. This, however, contributes to increased overhead operations to form the index candidate sets S^j_m, $j = 1, \ldots, K$ recursively after determining the interval I^j_m containing x_j. The empirically estimated cost for forming the K index sets for a test vector is KN logical and comparison operations.

Bit map. In the BM organization, the indices are position coded in a N bit word with the nth bit as one if the nth codevector is a candidate and zero otherwise. The BM storage requires $(3 + N/b)NK$ words, where b is the processor word length. These storage requirements can be very high for large dimensions and codebook sizes.

5.1.2. Computational Overheads of Mapping Table Search

In order to consider the computational overheads of the fast search based on the mapping table, we give here a search procedure for finding the nearest-neighbor of a test vector $\mathbf{x}$:

(a) Find the interval $I^j_m = [(y^j_m, y^y_{m+1})]$ in each coordinate j containing the component x_j of the test vector, i.e., $y^j_m \le x_j \le y^j_{m+1}$.

(b) Find $C' = \cap_{j=1}^{K} S^j_m$, the final set of candidate codevectors C' from intersection of the K index candidate sets S^j_m, $j = 1, \ldots, K$. Let $S^j_m = \{i^j_1, i^j_2, \ldots, i^j_{\beta j}\}$, where $\beta_j = |S^j_m|$. This can be obtained as follows:

```
C' = { }
C[i] = 0, i = 1, . . . , N
do j = 1, . . . , K
    do k = 1, . . . , βj
        C[i^j_k] = C[i^j_k] + 1
        if C [i^j_k] = K then C' = C' + {ι^j_k}
    enddo
enddo
```

(c) Carry out a full-search within this candidate set C' to obtain the actual nearest-neighbor of $\mathbf{x}$.

The first step (a) of interval location is carried out by a binary search in the mapping table of each of the coordinate axis, incurring a total cost of $K \log(2N)$ comparisons. In the case when the mapping table is organized as an IM/RT, an additional step is necessary to form the index candidate sets $S^j(\mathbf{x})$, $j = 1, \ldots, K$ by the recursion $S^j_m = S^j_{m-1} + \{s_m\}$, with $S^j_1 = \{s_1\}$ after determining the interval I^j_m containing x_j.

In step (b), we have given a simple, but efficient, procedure for finding the intersection of subsets. This is a variant of the shift method employed by Yunck [49] and will be referred to henceforth as the *intersection count* (IC) procedure (introduced in a related data structure [35]). This procedure is based on the observation that the candidate set required by the intersection of the K index sets consists of indices common to all the K index sets and hence will appear exactly K times in the collection of the K sets. All other indices which do not belong to the K-set intersection will occur less than K times. Therefore, it suffices to keep count of the number of times a particular index has occurred in all the K sets and the intersection set is formed by those indices which have a count of K. This intersection count procedure requires an additional memory of N scalar array of resolution $(\log \lceil K \rceil)$ bits. (In the method used by Yunck [49], the above procedure would correspond to using a K-bit register in which the count is maintained by shifting the contents of the register to the left with the full-count value after K shifts corresponding to 2^K.)

The intersection complexity of the IM storage can be quantified in the following manner. For a test vector $\mathbf{x}$, let $I^j(\mathbf{x})$ be the interval containing x_j and let $S^j(\mathbf{x})$ be the index set associated with this interval. Denoting the size of the set $S^j(\mathbf{x})$ by β_j (with bounds $1 \le \beta_j \le N$), the intersection complexity is proportional to $\beta = \sum_{j=1}^{K} \beta_j$, irrespective of the nature of the intersection procedure carried out on the K sets $S^j(\mathbf{x})$, $j = 1, \ldots, K$. The average β, $\bar{\beta}$, obtained over a large set of test vectors characterizes the basic

intersection complexity in the IM mode and has bounds, $K \leq \overline{\beta} \leq KN$. The complexity of the intersection-count procedure given above is, therefore, β integer increment and scalar comparison operations in performing the counting and in checking for count value K to form the final intersection indices. The average intersection complexity is therefore $\delta = (\overline{\beta} + N)$ scalar operations per vector, including the cost of initializing the count array of size N to zero and has bounds $(K + N) \leq \delta \leq (K + 1)N$.

Based on empirical observation, Cheng *et al.* estimated the intersection complexity to be about KN comparisons for the IM case and $KN/3$ logical operations for the BM case. The BM case was noted to have a smaller intersection complexity as it allows the intersection computation using logical AND operations on the position encoded bit representation of the candidate indices.

The total overhead computational complexity of the mapping table-based intersection procedure (as given by steps (a) and (b)) is, therefore, $(K \log 2N + \overline{\beta} + N)$ scalar operations per vector, and that of box-search is $c(\mathbf{x})$ scalar comparisons per vector, with bounds $N \leq c(\mathbf{x}) \leq 2KN$. From simulation results (Section 7.2), we see that the average value of this cost for box-search asymptotically saturates to a value of $2N/K$ per vector component, which decreases linearly with dimension. This is significantly less than the corresponding bounds of $[\log 2N + (K + N)/K$, $(\log 2N + (K + 1)N/K]$ of the mapping table-based intersection procedure, which has an average complexity bound of $O(N)$ which increases linearly with dimension. Moreover, we also see that $(\overline{\beta} + N)$ scalar operations/vector of the intersection count procedure given above represents a overhead complexity whose average bound is lower than (and worst-case bound comparable to) the intersection complexity of KN for the IM mode observed by Cheng *et al.* [10].

PRINCIPAL COMPONENT ROTATION

In this section, we propose and study the influence of principal component rotation of the codevectors in reducing the complexity of different search algorithms using the Voronoi projections. The complexity of the projection-based search depends mainly on the extent of intersection between the Voronoi boxes. This in turn reflects in the extent of overlap of the projections of the Voronoi regions on the coordinate axes. We show that the projections of the Voronoi regions obtained using the principal component directions as the new "rotated" coordinate system reduce the relative overlap between the Voronoi boxes significantly and improve the efficiency of the projection-based search in decreasing the main search complexity of number of distance computations, i.e., by reducing the size of the final set of candidate codevectors. We also show that the box-search scheme derives advantage by the principal component rotation in reducing both the main search complexity and the overhead computational complexity and that, in comparison, the mapping table-based intersection search suffers increased storage and computational overheads.

The K principal component directions of a given set of vector points are the directions of the eigenvectors of the covariance matrix of the points [14]. If λ_i, $i = 1, \ldots, K$ (with $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_K$) are the eigenvalues of the covariance matrix of the points, then the principal component directions $\mathbf{e}_i$, $i = 1, \ldots, K$ represent the orthogonal directions of maximum variance in the data, the variance of the projections of the points along the direction $\mathbf{e}_i$ being proportional to the corresponding eigenvalue λ_i. $\mathbf{e}_1$ is referred to as the first principal component direction (or coordinate axis), $\mathbf{e}_2$ the second principal component direction, and so on. $\mathbf{e}_1$ is the direction of maximum variance of the projections of the given set of points on it, and $\mathbf{e}_2$ is the direction perpendicular to $\mathbf{e}_1$ with the largest variance of the point projections on it. In general, $\mathbf{e}_i$ is the direction perpendicular to $\mathbf{e}_1, \ldots, \mathbf{e}_{i-1}$ having the largest variance of the projections of the points on it. $\mathbf{e}_i$ is also the direction which is the solution for the best perpendicular least-square error (or eigenvector) fit perpendicular to $\mathbf{e}_1, \ldots, \mathbf{e}_{i-1}$, with minimum cross-sectional volume orthogonal to it in the subspace $\mathbf{e}_{i+1}, \ldots, \mathbf{e}_K$. The principal component rotation is particularly applicable in the case of speech waveform vector quantization where the vectors are highly correlated across their components. Figure 3a shows the high degree of correlation between the components of speech waveform vectors for the 2-dimensional case.

6.1. Effect of Principal Component Rotation on Extent of Projection Overlap

In view of the basic property of the principal component directions, the projections of the Voronoi regions on the first principal component axis will have the maximum spread (more than on any other direction) and the projection intervals will be maximally separated from each other. This implies minimum overlap between the various projection intervals. (This can be easily visualized by considering the extreme case where the codevectors are aligned

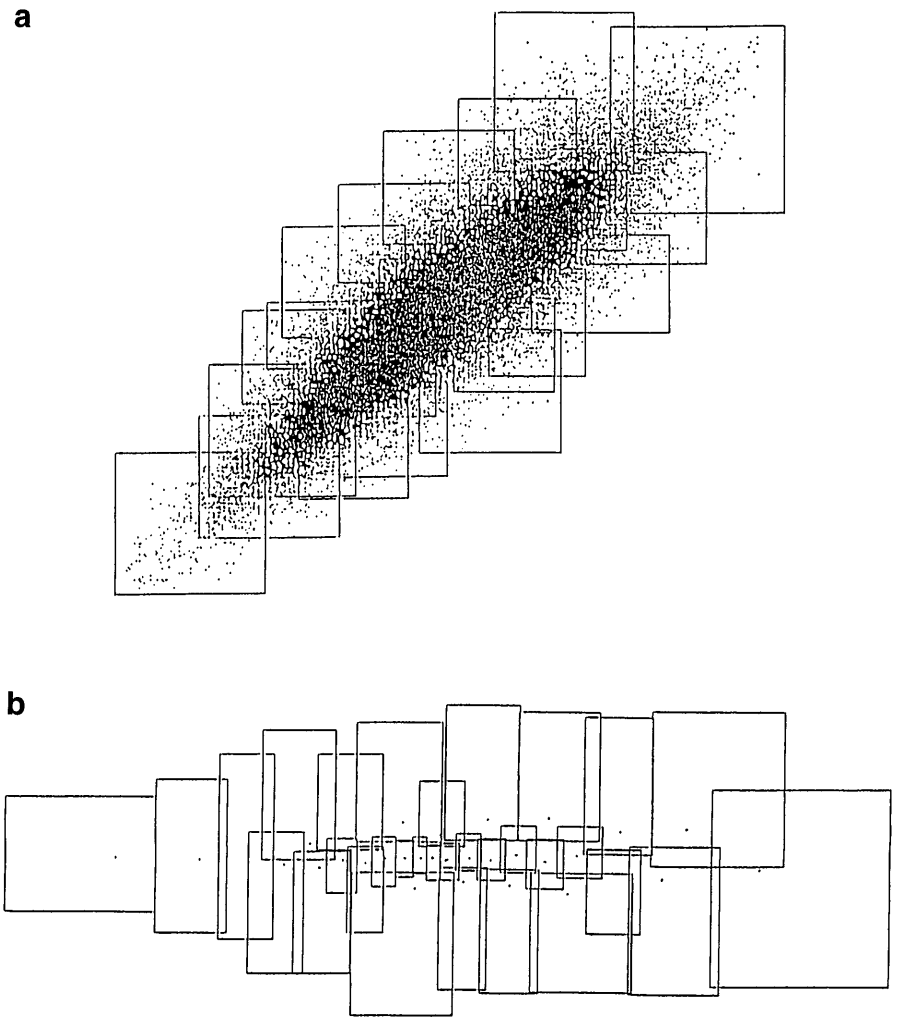

FIG. 3. Voronoi boxes of codevectors for codebook size $N = 32$ and dimension $K = 2$ for (a) without rotation along with 150,000 vectors of speech waveform vectors used for obtaining the projection estimates and (b) with rotation.

on a straight line along some direction. This will then be the direction of the first principal component with some variance and other directions having zero variance. This actually degenerates to a 1-dimensional case with the Voronoi projections corresponding to contiguous and disjoint intervals.) This will in turn reflect in the smallest $\overline{\alpha}^j$, the average size of the index sets of the contiguous intervals formed by the overlapping projections. The extent of overlap on the second principal component axis will be more than on the first, and in general, the overlap will increase from principal coordinate axis 1 to K. To illustrate this, Fig. 3 shows the Voronoi boxes of the codevectors of a codebook of size 32 in 2-dimensions for the rotated and the unrotated case alongwith 150,000 speech waveform vectors used for obtaining the projection estimates. This also shows how the data determine the box estimates and the extent of intersection between the various Voronoi boxes; the spreading out of the projections along the first coordinate axis in the rotated space and the reduction in the extent of overlap due to rotation can be noted here.

In Fig. 4, we show the effect of rotation on the extent of overlap across the coordinates by plotting $\overline{\alpha}^j$ and $\hat{\alpha}^j$ as a function of coordinate axis $j = 1, \ldots, K$ with and without rotation for dimension $K = 8$ and codebook size $N = 1024$. $\overline{\alpha}^j$ is the average index set size and $\hat{\alpha}^j$ is the maximum index set size $\max_{1 \le m \le 2N-1} \alpha_m^j$. It can be seen that for the unrotated case, $\overline{\alpha}^j$ and $\hat{\alpha}^j$ are rather unchanging across the coordinates, implying the codevector orientation to be along a direction which generates similar projection overlap on all the coordinate axis. For the rotated case, $\overline{\alpha}^j$ and $\hat{\alpha}^j$ values are much smaller than

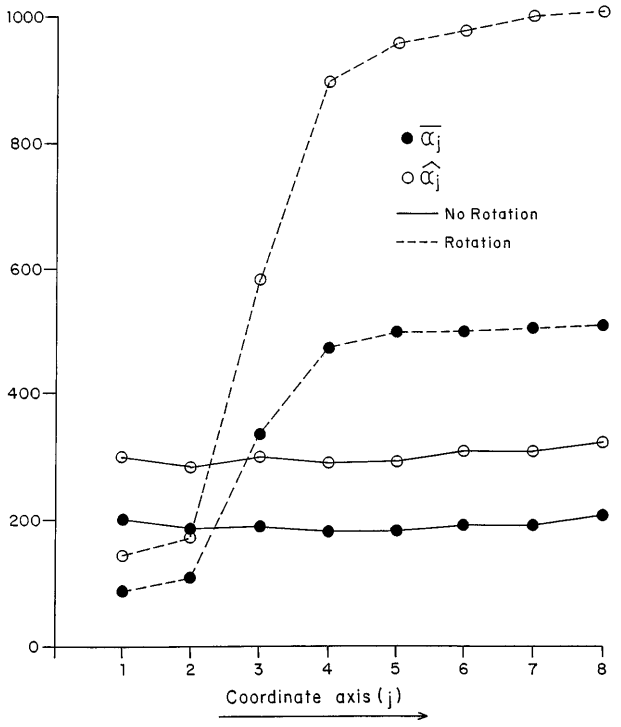

FIG. 4. Average and maximum index set sizes ($\bar{\alpha}_j$, $\hat{\alpha}_j$) for coordinate axis $j = 1, \ldots, K$ for dimension $K = 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

for the unrotated case in the first and second principal coordinate axes. However, for the higher coordinate directions in the rotated space, $\bar{\alpha}^j$ and $\hat{\alpha}^j$ increase to values larger than for the unrotated case. This is due to the fact that the variances of the codevector projections along these directions are smaller than the corresponding coordinates in the unrotated case. These directions belong to the subspace orthogonal to the first and second rotated axes and therefore have a smaller projected cross-sectional volume than for any other direction. Hence the projections on the directions in this subspace are confined within smaller cross sections with a resultant increase in their overlap.

In general, the number of principal component directions which have smaller $\bar{\alpha}^j$ and $\hat{\alpha}^j$ values than the corresponding unrotated coordinates and the differences in magnitudes of the index set sizes between the rotated and unrotated cases depend on the correlation between the vector components and the relative magnitudes of the variances along the principal coordinate directions. A highly favorable situation is one where a large number of principal component coordinates have significantly higher variances (resulting in lower $\bar{\alpha}^j$ and $\hat{\alpha}^j$ values) than the corresponding unrotated axes.

7. SIMULATION RESULTS

In this section, we present simulation results obtained in the context of vector quantization of speech waveform to characterize the complexity of the box-search and the mapping table-based search methods. First we describe the means of obtaining the Voronoi projections of a given set of codevectors. The projection of each Voronoi region can in principle be obtained either by analytical procedures such as linear programming techniques or by constructing the Voronoi diagram, from which only the vertex information of each Voronoi polytope is used further for determining the projections of each Voronoi region. But, as seen earlier, these can involve considerable computational cost. Alternately, the projection boundary points can be determined from the "extreme" vectors in the respective Voronoi region with respect to each coordinate axis; this is done by first generating a Voronoi "cluster" around each codevector using a Monte Carlo approach of encoding a large amount of training data (or uniformly distributed points) and then finding the extreme vectors of each cluster with respect to each coordinate axis. The projection estimates thus obtained by encoding the points that fall within the region are then

$$P^j_{i,L} = \min_{\mathbf{x} \in R} \; x_j : \mathbf{q}(\mathbf{x}) = \mathbf{c}_i$$

and

$$P^j_{i,U} = \max_{\mathbf{x} \in R} \; x_j : \mathbf{q}(\mathbf{x}) = \mathbf{c}_i,$$

where R represents the training set (or the domain of interest in the R^K space in the case of uniformly distributed points).

This constitutes the main preprocessing step which involves encoding of a large training data set and the use of scalar comparison operations to update and maintain the extreme projections of vectors on the coordinate axes for each Voronoi region. The estimates so obtained are approximate, but have been found to be quite acceptable when obtained using sufficiently large data. If the size of the training set R is n, then the cost in obtaining the Voronoi projection estimates is nN distance computations (to find $\mathbf{q}(\mathbf{x})$, $\forall \mathbf{x} \in R$) and $2Kn$ scalar comparisons to update the projections as $P^j_{i,L} = \min \{x_j : \mathbf{q}(\mathbf{x}) = \mathbf{c}_i\}$ and $P^j_{i,U} = \max \{x_j : \mathbf{q}(\mathbf{x}) = \mathbf{c}_i\}$, $j = 1, \ldots, K$, $\forall \mathbf{x} \in R$. This is an effective method to obtain the projection estimates since it can be performed as a part of the codebook

design at the end of the LBG algorithm [27]. Here, the nearest-neighbor of the vectors in the training set is available at the end of the algorithm. It is necessary only to maintain the updates of the projections, the preprocessing cost then being only $2Kn$ scalar comparisons. In the present simulation, the projections were obtained using 150,000 vectors of speech waveform for dimensions $K = 2, 4, 6, 8$, and 10 and codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. All results shown here were obtained using 50,000 vectors from inside the data used for estimating the Voronoi projections.

7.1. Main Complexity of Box-Search and Mapping Table Methods

Here, we give the main complexity of the fast search using the projections in terms of the number of distances computed, which is the size of the candidate set of codevectors obtained by box-search or the mapping table-based intersection method. Figure 5 shows the average and maximum number of codevectors searched by the projection methods for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. Figure 6 shows the average and maximum number of codevectors searched for dimensions $K = 2, 4, 6$, and 8 and codebook size $N = 1024$. This is the average and maximum complexity of the fast search by projection method in terms of vector distances computed per vector. The excellent complexity reduction of the projection method with respect to full-search can be noted in all these cases. (The curves marked as "Rotation" correspond to the projections obtained using the principal component rotation which is discussed in a later section; here, we concern ourselves only with the results using the projections obtained on the regular "unrotated" space as in Cheng *et al.* [10] and we refer only to the curves marked "No rotation" in Figs. 5 and 6 and in subsequent figures.)

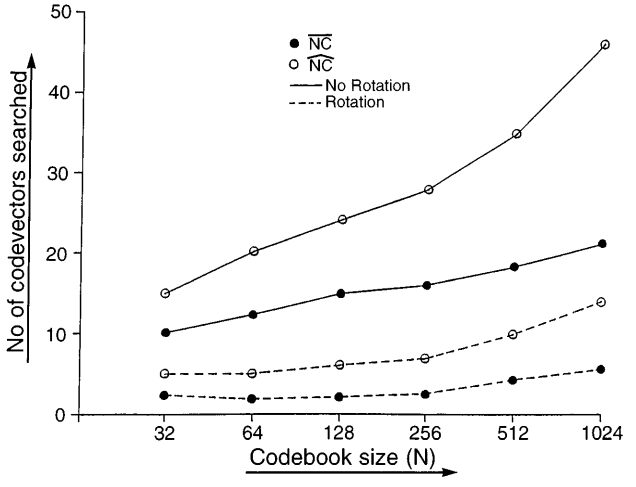

FIG. 5. Average and maximum number of codevectors searched ($\overline{NC}$, $\widehat{NC}$) by projection method (intersection count/box-search procedures) for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512$ and 1024. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

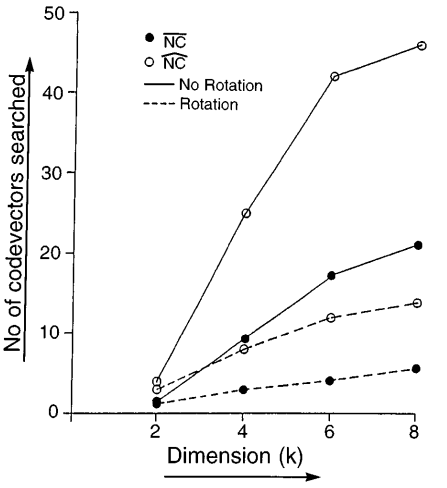

FIG. 6. Average and maximum number of codevectors searched ($\overline{NC}$, $\widehat{NC}$) by projection method (intersection count/box-search procedures) for dimensions $K = 2, 4, 6, 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

7.2. Overhead Complexities of Box-Search and Mapping Table Methods

Here, we compare the computational overhead cost of the intersection count procedure using the mapping table and that of the box-search. As seen earlier (in Section 5.1.2), the intersection count incurs a total of $(K \log 2N + \overline{\beta} + N)$ scalar operations (such as comparisons and increment) per vector and the box-search incurs $c(\mathbf{x})$ scalar comparisons per vector. These are also numerically comparable directly since the constituent scalar operations are same and of similar cost. Figures 7a and 7b respectively show the average and maximum costs per vector-component incurred by the IC and the BS procedures for dimension $K = 8$ and different codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. The box-search can be seen to have a smaller (but comparable) overhead cost than the intersection count procedure using the mapping table.

In Figs. 8a and 8b, we show the above costs for codebook size $N = 1024$ and different dimensions $K = 2, 4, 6, 8$, and 10. Here, it is important to observe that while the cost of IC increases with dimension, that of BS shows a marked decrease. This is due to

the fact that in the case of the intersection procedure, the cost is proportional to the number of index sets and their sizes. An increase in dimension increases the number of index sets and their sizes, thereby resulting in an increase in the computational overhead cost with increase in dimension. On the other hand, in the case of box-search, as the dimension increases, there is an increase in the degree of freedom for a box to *not contain* a given test vector with a consequent reduction in the average number of comparisons required per coordinate in the box-test before a codevector can be rejected. In addition, we have shown in Figs. 7 and 8 the cost of N operations per vector component (plot marked as IM/FT), reported to be required in [10] for this computation using the image-map/full-table organization of the mapping table. It can be noted that, for the various codebook sizes and dimensions shown in Figs. 7 and 8, the overhead cost of the IC procedure proposed here is considerably less than the N operations per vector component observed in [10].

In Figs. 9 and 10, we show the average number of comparisons required per codevector in the box-search. As mentioned earlier, this measures how quickly a codevector is eliminated within the maximum $2K$ comparisons required in the worst case to determine if a point lies within a box. Figure 9 shows this for dimension $K = 8$ and codebook sizes $N = 32$,

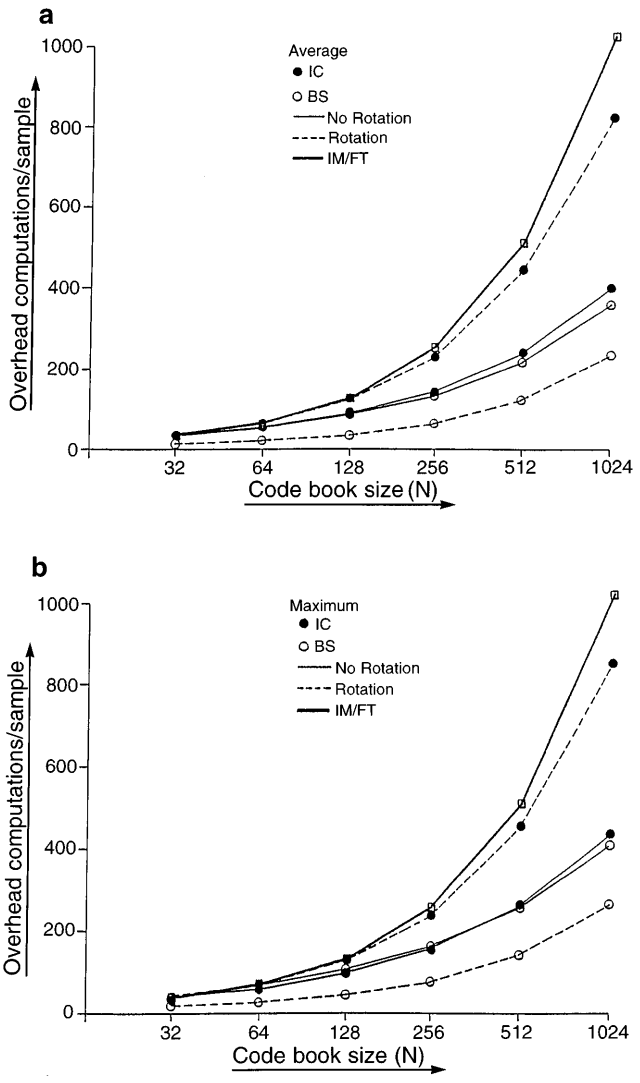

FIG. 7. Average (a) and maximum (b) overhead computations/vector-component of intersection count (IC) procedure and box-search (BS) procedure for dimension $K = 8$ and codebook sizes $N = 32$, 64, 128, 256, 512, and 1024. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

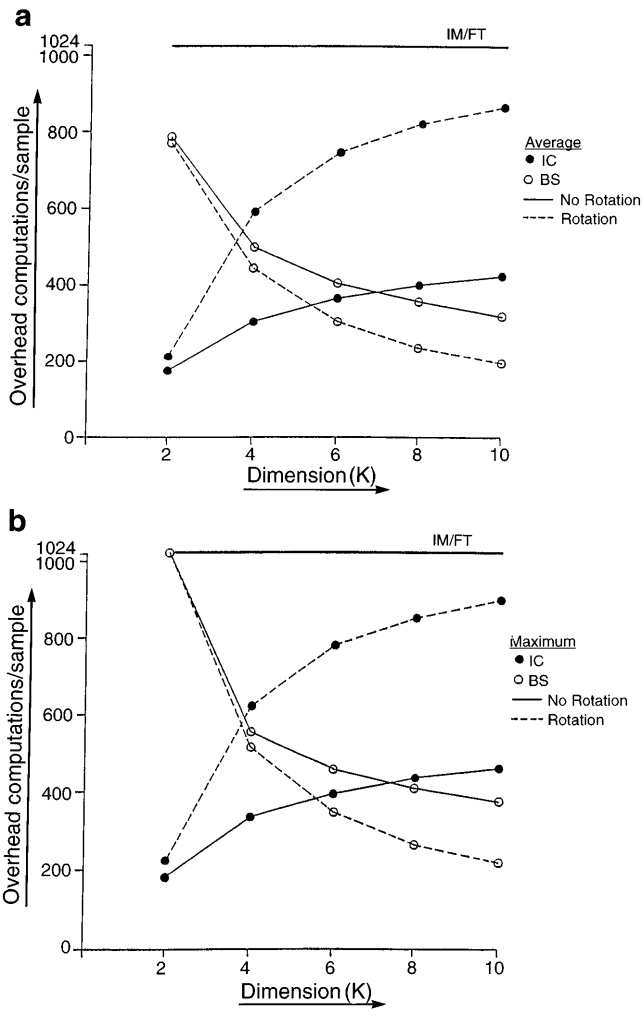

FIG. 8. Average (a) and maximum (b) overhead computations/vector component of intersection count (IC) procedure and box-search (BS) procedure for dimension $K = 2$, 4, 6, 8, 10 and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

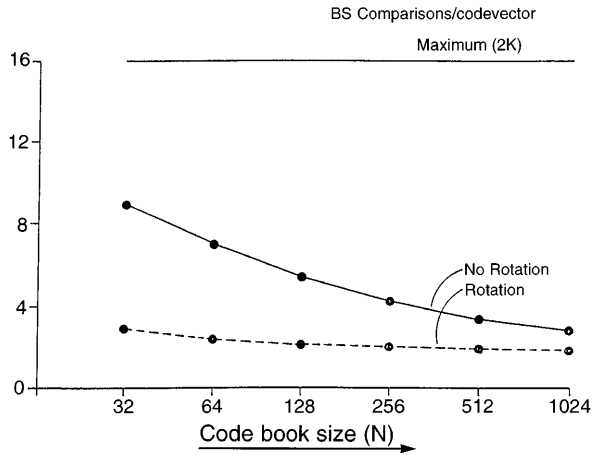

FIG. 9. Average number of overhead comparisons per codevector in box-search for dimension $K = 8$ and codebook sizes $N = 32$, 64, 128, 256, 512, and 1024. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

64, 128, 256, 512, and 1024, while Fig. 10 shows this for codebook size $N = 1024$ and dimensions $K = 2$, 4, 6, 8, and 10. In both cases, the average box-test cost can be seen to be considerably less than the worst-case $2K$ cost. It is important to note here that the average value for the various dimension and codebook sizes asymptotically saturates to a value of 2, placing the empirical cost of box-search per vector component at $2N/K$, which is significantly less than the cost of N per vector component of the mapping table-based intersection procedure for large dimensions.

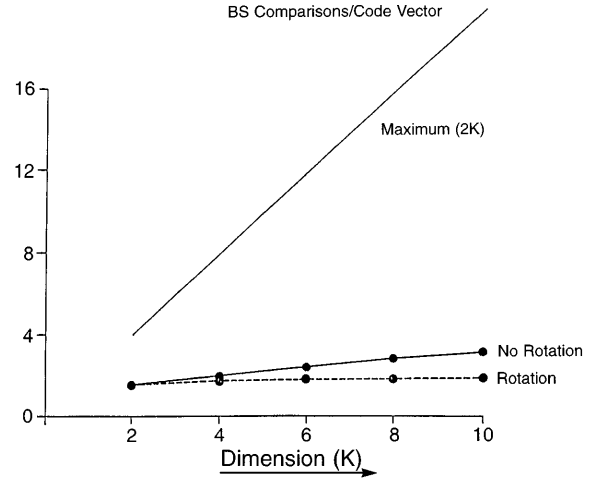

FIG. 10. Average number of overhead comparisons per codevector in box-search dimension $K = 2$, 4, 6, 8, 10 and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

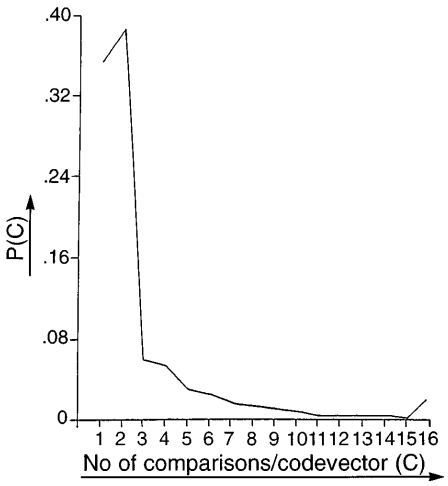

FIG. 11. Histogram of the number of overhead computations per codevector in box-search for dimension $K = 8$ and codebook size $N = 1024$.

In Fig. 11, we show the histogram of the number of comparisons per codevector for dimension $K = 8$ and codebook size $N = 1024$. As noted earlier, this has a range of 1 to $2K$. It can be noted that the histogram has a sharp high probability mode at the lower end for values 1 and 2, falling drastically beyond this, indicating that most of the codevectors, for a large representative test data, tend to be eliminated within 2 comparisons; this fraction is about 70% in the histogram shown. This histogram clearly demonstrates the excellent efficiency of the box-search elimination in having a low overhead computational cost. It can also be noted from this figure that the probability (number of comparisons/codevector = 16) corresponds to the number of codevectors for which $2K$ comparisons were computed and incur the full box-test cost and is the upper bound of the number of codevectors whose distances are computed; this worst-case probability of $2K$ comparisons is about 0.02 and corresponds to approximately 10 codevectors (also shown as the worst-case complexity of box-search in Table 1).

Figure 12 compares the total storage required by the mapping table with that of box-search for dimension $K = 8$ and codebook sizes $N = 32$, 64, 128, 256, 512, and 1024. The box-search storage of $2KN$ can be observed to be significantly less than that of the IM/FT $((3 + \alpha)NK)$ and BM/FT $((3 + N/b)NK$ with $b = 32)$.

7.3. Effect of Principal Component Rotation on the Complexity of Box-Search and Mapping Table Methods

Here, we show the number of codevectors examined (final candidate set size) for various dimensions

TABLE 1
Comparison of the Intersection Count Procedure and Box-Search with Full-Search

Algorithm	Main complexity $\overline{NC}$	Main complexity $\widehat{NC}$	Overhead computation Ave	Overhead computation Max	Total storage	m	a	c
IC (NR)	21.1	46	399.3	436.0	1596447	21.1	39.6	401.8
IC (R)	5.6	14	359.4	411.1	3120534	13.6	10.5	824.6
BS (NR)	21.1	46	822.1	855.0	24576	21.1	39.6	360.1
BS (R)	5.6	14	237.2	265.6	24576	13.6	10.5	237.9
Full-search	1024	1024	—	—	8192	1024	1920	127.9

Note. Dimension $K = 8$. Codebook Size $N = 1024$. Data: 50,000 vectors of speech waveform. NR, no rotation. R, rotation. Main complexity: [$\overline{NC}$, $\widehat{NC}$], [average, maximum] number of codevectors searched (distance computations). Overhead computation: Integer increment and comparisons for intersection count procedure (IC); Real comparisons for box-search (BS); [m, a, c], average number of [multiplications, additions, comparisons] per vector-component.

and codebook sizes in Figs. 5 and 6 with and without principal component rotation. The rotated case can be seen to have a consistently lower complexity than the unrotated case. This is so, despite the increased overlap in the higher principal coordinate directions, because the reduced overlap in the lower principal coordinate directions produces a resultant reduction in the overlap between the actual Voronoi boxes, which in turn reflects in the final candidate set size (i.e., the intersection set of the individual coordinate index sets is smaller for the rotated case since the lower (main) principal coordinate directions give a smaller candidate set).

In Fig. 13, we show the histogram of the number of codevectors searched by the projection method with and without the principal component rotation for dimensions $K = 8$ and codebook size $N = 1024$. It can be noted that there is a significant reduction in the range of the histogram with a shift in the high probability mode toward the lower range clearly indicating the extent of reduction in both the average and maximum complexity of the search.

The increase in the average index set size with rotation results in an increase in both storage of the mapping table based intersection procedure and computational overheads of the intersection procedure. In Figs. 7 and 8, the increase in the computational overhead of the intersection procedure due to rotation over the unrotated case can be seen. However, the box-search overhead cost has reduced due to rotation. This can also be noted in Figs. 9 and 10, which show the box-test comparisons per codevector.

In this context, it is possible to consider the effect of using only the main principal coordinate directions which have more highly reduced index set sizes than the corresponding unrotated case. By this, the

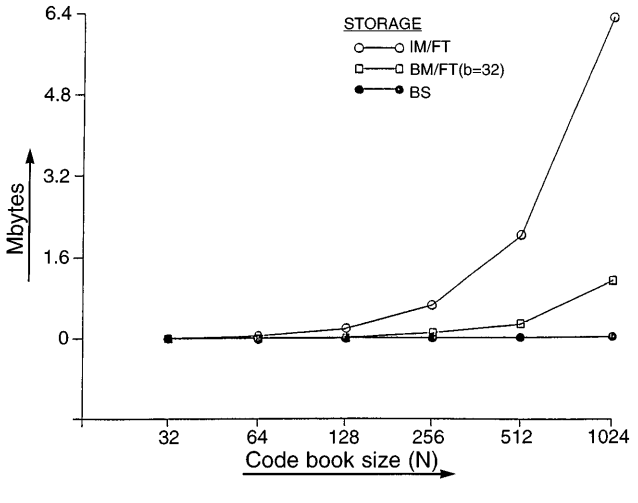

FIG. 12. Total storage required by mapping table (IM/FT and BM/FT) and box-search (BS) for dimension $K = 8$ and codebook sizes $N = 32$, 64, 128, 256, 512, and 1024.

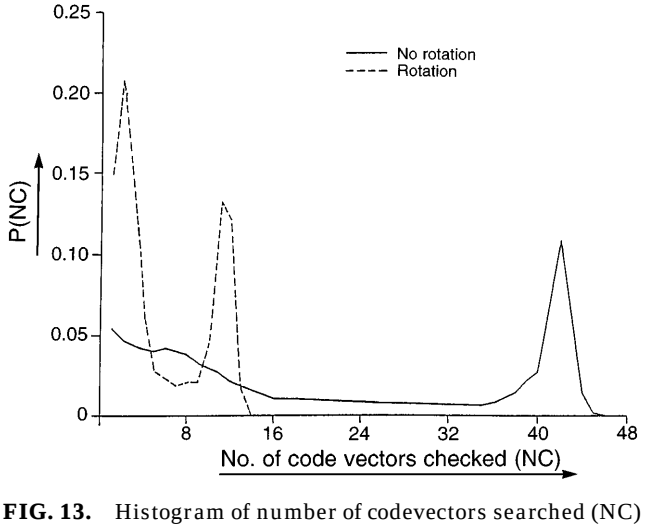

FIG. 13. Histogram of number of codevectors searched (NC) for dimension $K = 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

mapping table incurs reduced storage and computational overheads. However, since fewer index sets are used in finding the final candidate set, this will result in an increase in the final candidate set size and the search complexity in comparison to the full K-axes intersection in the rotated case. The same applies to the box-search also, where the use of less than K axes results in decreasing the box-test overheads. However, the reduction in the box constraints in using only some of the projections results in use of boxes "unbounded" in the higher coordinate directions, consequently increasing the final candidate set size and the final search complexity. The trade-off between the reduction in the overhead computation and the increase in the final search complexity in both the mapping table-based intersection procedure and the box-search can be studied only empirically, since the number of principal component axes used and their relative index set sizes with respect to the unrotated case depend on the correlation in the data and the size and dimensionality of the problem.

An interesting effect of the rotation in reducing the extent of overlap between the Voronoi boxes and consequently in the final candidate set size can be observed from the average error incurred (or equivalently the signal-to-noise ratio (SNR)) when the test data are quantized by selecting the final codevector *randomly* from the final candidate set. Figure 14 compares the full-search SNR and the "random selection" SNR (denoted by SNR_r) for the rotated and unrotated case, respectively, for dimensions $K = 8$ and codebook sizes $N = 32$, 64, 128, 256, 512, and 1024. Rotation can be seen to improve the SNR_r very significantly closer to the full-search SNR for all the codebook sizes.[3]

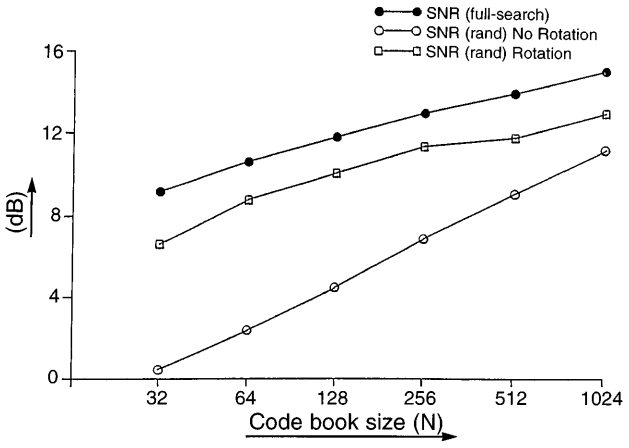

FIG. 14. SNR for full-search and for encoding by random selection (SNR) from final candidate set in projection method for dimension $K = 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation. *Rotation:* with principal component rotation.

7.4. Overall Complexity of Different Projection-Based Search Algorithms

In Table 1, we give a comparison of the mapping table-based IC procedure and the BS procedure (with and without the use of rotation) with the full-search complexity for dimensions $K = 8$ and codebook size $N = 1024$ in terms of computational complexity, storage and overhead computational cost. Here, the following can be noted:

1. IC and BS procedures have the same complexity in terms of the average and maximum number of codevectors examined, $\overline{NC}$ and $\widehat{NC}$.
2. The mapping table-based intersection procedure has an exhorbitant storage complexity in comparison to the box-search method.
3. The overhead costs of the IC procedure is roughly comparable to that of BS for the unrotated case, but, for the rotated case, while the overhead cost of IC procedure almost doubles, that of box-search shows a significant decrease. The storage requirement of the IC procedure also nearly doubles with rotation.
4. The overall computational complexity of the various algorithms, including the overhead costs, can also be compared in terms of the standard (macs) measure of average number of multiplications, additions, and comparisons per vector component [10]; the box-search with rotation can be seen to have the lowest complexity and offers excellent reduction over the full-search complexity with only the cost of scalar comparisons being double that of full-search.

[3] We note here that [10] mentions such a random selection to be very efficient with a difference of only 0.18 dB between the full-search and random selection SNR in a simulation for dimension $K = 8$ and codebook size $N = 1024$ using 14,720 vectors of data. However, from the simulation results shown in Fig. 14, we observe that this difference can be much larger—as much as 9 dB for codebook size $N = 32$ and about 4 dB for codebook size $N = 1024$. The small difference of 0.18 dB observed by Cheng *et al.* can possibly be attributed to insufficiency of the amount of data used (about one-tenth that used here). This will result in underestimation of the Voronoi projections and will produce only a limited extent of overlap. Consequently, the final candidate sets will be small and will mainly consist of codevectors which are close to each other to permit even their underestimated Voronoi boxes to overlap and contain a test vector. As a result, the contention in a random selection from the candidate set will result in only small quantization errors accounting for the small reduction in the SNR. Due to the large SNR difference we have observed, contrary to [10], we believe that it may not be possible to make use of *random encoding* in the Voronoi projection based search for practical purposes.

8. CONCLUSION

In this paper, we have considered fast search techniques based on the implicit geometric interpretation of the nearest-neighbor search in terms of a geometric construct called the Voronoi diagram of the given set of points. This paper is mainly concerned with fast search techniques based on the projections of the Voronoi regions; it highlights the basic efficiency of using the Voronoi projections and shows that fast search based on the Voronoi projection, using appropriate data structures, can result in very practical and highly efficient fast search algorithms.

In this respect, this paper provides a detailed study of two fast search techniques using the Voronoi projections, namely, the box-search and the mapping table-based search. We have investigated the box-search as an alternate and simpler interpretation of the projection information, with significantly less storage than the mapping table-based search (which has a very high storage complexity). In the mapping table-based method, we have considered an intersection count procedure for obtaining a small set of candidate codevectors to be searched using the Voronoi projections. This procedure, while being simple to implement, has been shown to have less complexity than the procedure employed by Cheng *et al.* [10] to use the Voronoi projections.

The main observations in this study can be summarized as follows:

1. The box-search requires a significantly low storage of $2KN$ words in comparison to the much higher storage of $(3 + \alpha)K\,N$ of the mapping table.

2. The overhead complexity of box-search decreases with increasing dimension. In contrast, the intersection complexity and storage increase with dimension in the mapping table-based search. The average value of the overhead computational cost for box-search asymptotically saturates to a value of $2N/K$ per vector component, which decreases linearly for increase in dimension. This is significantly less than the corresponding bounds of $[\log 2N + (K + N)/K$, $(\log 2N + (K + 1)N/K]$ of the mapping table-based intersection procedure, which has an average complexity bound of $O(N)$ which increases linearly with dimension.

3. Box-search derives benefit from principal component rotation in reducing both the search complexity and overhead computation. However, in the case of the mapping table, principal component rotation reduces the main search complexity only at the expense of significant increase in storage and overhead computations.

REFERENCES

1. Abut, H. *Vector Quantization*. IEEE, Piscataway, NJ, 1990.
2. Avis, D., and Bhattacharya, B. K. Algorithms for computing *d*-dimensional Voronoi diagrams and their duals. In *Advances in Computing Research*, Vol. I, pp. 159–180. JAI Press, London, 1983.
3. Barlaud, M., *et al. IEEE Trans. Image Process.* **5**(2) (1996). [Special issue on vector quantization]
4. Bei, C. D., and Gray, R. M. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Trans. Commun.* **COM-33** (1985), 1132–1133.
5. Bowyer, A. Computing dirichlet tesselations. *Comput. J.* **24**(2) (1981), 162–172.
6. Brown, K. Q. *Geometric Transformations for Fast Geometric Algorithms.* Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Dec. 1979.
7. Browstow, W., Dussault, J.-P., and Fox, B. L. Construction of Voronoi polyhedra. *J. Comput. Phys.* **29** (1978), 81–92.
8. Burkhard, W. A., and Keller, R. M. Some approaches to best-match file searching. *Commun. ACM* **16**(4) (1973), 230–236.
9. Chen, S. H., and Pan, J. S. Fast search algorithm for VQ-based recognition of isolated words, *IEE Proc.* **136**(6) (1989), 391–396.
10. Cheng, D. Y., Gersho, A., Ramamurthi, B., and Shoham, Y. Fast search algorithms for vector quantization and pattern matching. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Mar. 1984, Vol. 1, pp. 9.11.1–9.11.4.
11. Clarkson, K. L. A probabilistic algorithm for the post office problem. In *Proc. 17th Ann. ACM Symp. Theory Comput.*, 1985, pp. 175–184.
12. Clarkson, K. L. A randomized algorithm for closest-point queries. *SIAM J. Comput.* **17**(4) (1988), 830–847.
13. Dobkin, D. P., and Lipton, R. J. Multidimensional searching problems. *SIAM J. Comput.* **5**(2) (1976), 181–186.
14. Duda, R. O., and Hart, P. E. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.
15. Edelsbruner, H. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin/New York, 1987.
16. Fischer, F. P., and Patrick, E. A. A preprocessing algorithm for nearest neighbour decision rules. In *Proc. Nat. Electronics Conf.*, 1970, Vol. 26, pp. 481–485.
17. Fukanaga, K., and Narendra, P. M. A branch and bound algorithm for computing *k*-nearest neighbours. *IEEE Trans. Comput.* **24** (1975), 750–753.
18. Gersho, A., and Cuperman, V. Vector quantization: A pattern matching technique for speech coding. *IEEE Commun. Mag.* **21** (1983), 15–21.
19. Gersho, A., and Gray, R. M. *Vector Quantization and Signal Compression*. Kluwer, Boston, 1992.
20. Gray, R. M. Vector quantization. *IEEE ASSP Mag.* **1** (1984), 4–29.
21. Huang, S. H., and Chen, S. H. Fast encoding algorithm for VQ-based image coding. *Electron. Lett.* **26**(19) (1990), 1618–1619.

22. Kalantari, I., and McDonald, G. A data structure and an algorithm for the nearest point problem. *IEEE Trans. Software Eng.* **SE-9**(5) (1983), 631–634.
23. Kamgar-Parsi, B., and Kanal, L. N. An improved branch and bound algorithm for computing k-nearest neighbours. *Pattern Recognition Lett.* **3** (1985), 7–12.
24. Klee, V. On the complexity of *d*-dimensional Voronoi diagrams. *Arch. Math.* **34** (1980), 75–80.
25. Kleijn, W. B., and Paliwal, K. K. *Speech Coding and Synthesis.* Elsevier, Amsterdam, 1995.
26. Lee, D. T., and Preparata, F. P. Computational geometry—A survey. *IEEE Trans. Comput.* **C-33**(12) (1984), 1072–1101.
27. Linde, Y., Buzo, A., and Gray, R. M. An algorithm for vector quantization design. *IEEE Trans. Commun.* **COM-28** (1980), 84–95.
28. Makhoul, J., Roucos, S., and Gish, H. Vector quantization in speech coding. *Proc. IEEE* **73** (1985), 1555–1588.
29. Niemann, H., and Goppert, R. An efficient branch-and-bound nearest-neighbour classifier. *Pattern Recognition Lett.* **7** (1988), 67–72.
30. Orchard, M. T. A fast nearest-neighbor search algorithm. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, Toronto, Canada,* May 1991, pp. 2297–2300.
31. Paliwal, K. K., and Ramasubramanian, V. Effect of ordering the codebook on the efficiency of the partial distance search algorithm for vector quantization. *IEEE Trans. Commun.* **COM-37** (1989), 538–540.
32. Preparata, F. P., and Shamos, M. I. *Computational Geometry.* Springer-Verlag, Berlin/New York, 1985.
33. Ramasubramanian, V., and Paliwal, K. K. An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation. In *Signal Processing V: Theories and Applications, Proc. of the EUSIPCO '90, Barcelona, Spain* (L. Torres-Urgell and M. A. Lagunas-Hernandez, Eds.), pp. 1323–1326, North-Holland, Amsterdam, 1990.
34. Ramasubramanian, V., and Paliwal, K. K. An efficient approximation-elimination algorithm for fast nearest-neighbour search based on a spherical distance coordinate formulation. *Pattern Recognition Lett.* **13**(7) (1992), 471–480.
35. Ramasubramanian, V., and Paliwal, K. K. An efficient approximation-elimination algorithm for fast nearest-neighbor search. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, San Francisco,* pp. I89–I92, Mar. 1992.
36. Van Rijsbergen, C. J. The best-match problem in document retrieval. *Commun. ACM* **17**(11) (1974), 648–649.
37. Seidel, R. A convex hull algorithm for points in even dimensions. Technical Report Comp. Sci. Tech. Rep. 81-14, University of British Columbia, Vancouver, 1981.
38. Seidel, R. The complexity of Voronoi diagrams in higher dimensions. In *Proc. 20th Ann. Allerton Conf. Commun. Control and Comput.*, pp. 94–95, 1982.
39. Shapiro, B. A. The choice of reference points in best-match file searching. *Commun. ACM* **20**(5) (1977), 339–343.
40. Shasha, D., and Wang, Tsong-Li. New techniques for best-match retrieval. *ACM Trans. Inform. Syst.* **8**(2) (1990), 140–158.
41. Soleymani, M. R., and Morgera, S. D. An efficient nearest-neighbour search method. *IEEE Trans. Comput.* **COM-35**(6) (1987), 677–679.
42. Soleymani, M. R., and Morgera, S. D. A high speed search algorithm for vector quantization. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* 1987, pp. 1946–1948.
43. Soleymani, M. R., and Morgera, S. D. A fast mmse encoding technique for vector quantization. *IEEE Trans. Commun.* **COM-37**(6) (1989), 656–659.
44. Vidal, E. An algorithm for finding nearest neighbours in (approximately) constant average time complexity. *Inform. Process. Lett.* **4**(3) (1986), 145–157.
45. Vidal, E., Rulot, H. M., Casacuberta, F., and Benedi, J. M. On the use of a metric-space search algorithm (AESA) for fastdtw-based recognition of isolated words. *IEEE Trans. Acoust. Speech Signal Process.* **36**(5) (1988), 651–660.
46. Watson, D. F. Computing the *n*-dimensional delaunay tesselation with applications to Voronoi polytopes. *Comput. J.* **24**(2) (1981), 167–172.
47. Yao, A. C. On constructing minimum spanning trees in *K*-dimensional spaces and related problems. *SIAM J. Comput.*, **11**(4) (1982), 721–736.
48. Yao, F. F. Computational geometry. In *Handbook of Theoretical Computer Science* (J. van Leeuwen, Ed.), pp. 344–389. Elsevier, Amsterdam, 1990.
49. Yunck, T. P. A technique to identify nearest-neighbours. *IEEE Trans. Syst. Man Cybern.* **SMC-6**(10) (1976), 678–683.
50. Ramasubramanian, V., and Paliwal, K. K. Fast nearest-neighbor search based on Voronoi projections and its application to vector quantization encoding. *IEEE Trans. Speech and Audio Processing,* 1997 (under review).