Correspondence

Fast Nearest-Neighbor Search Based on Voronoi Projections and Its Application to Vector Quantization Encoding

V. Ramasubramanian and Kuldip K. Paliwal

***Abstract*—In this work, we consider two fast nearest-neighbor search methods based on the projections of Voronoi regions, namely, the box-search method and the cell-partition search method. We provide their comprehensive study in the context of vector quantization encoding. We show that the use of principal component transformation reduces the complexity of Voronoi-projection based search significantly for data with high degree of correlation across their components.**

***Index Terms*— Fast nearest-neighbor search, vector quantization, Voronoi projections.**

I. Introduction

In this work, we consider the problem of 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 [3]. Vector quantization encoding is the minimum-distortion quantization of a vector $\mathbf{x} = (x_1, \cdots, x_K)$ (referred to as the *test vector*), using a given set of N K-dimensional *codevectors* (called the *codebook* $\mathbf{C} = \{\mathbf{c}_i\}_{i=1, \cdots, N}$), under some distance measure $d(\mathbf{x}, \mathbf{y})$. This involves finding the nearest-neighbor of $\mathbf{x}$ in $\mathbf{C}$, given by $\mathbf{q}(\mathbf{x}) = \mathbf{c}_j : d(\mathbf{x}, \mathbf{c}_j) \leq d(\mathbf{x}, \mathbf{c}_i), \quad i = 1, \cdots, N$. This requires N vector distance computations $d(\mathbf{x}, \mathbf{c}_i)$ using the exhaustive full-search for a codebook of size N. Thus, the computational complexity of the nearest neighbor search is very high, especially when N and K are large. As a result, the problem of developing algorithms for fast nearest-neighbor search has recently attracted significant attention. In addition to its application in vector quantization, fast nearest-neighbor search is important in several other areas such as pattern classification, nonparametric estimation, information retrieval from multikey databases, etc.

Our objective in this paper is to investigate fast nearest-neighbor search algorithms based on Voronoi projection information. Cheng *et al.* [1] have proposed a cell-partition based fast search method where the K-dimensional space is partitioned into $(2N - 1)^K$ hyperrectangular cells using the Voronoi-projection information. They have suggested the use of mapping tables for reducing the exponential storage complexity of the cell-partition data structure to linear storage complexity, and an intersection procedure to locate a hyperrectangular cell for the given test vector. We propose the use of an intersection count procedure which has less complexity than the procedure employed in [1]. In addition, we propose a box-search (BS) method as an alternative to the cell-partition search method. This is based on direct and simple interpretation of the projection information. We carry out a comprehensive study of these two Voronoi-projection based fast search methods (namely, BS and cell-partition search). We also investigate the effect of principal component rotation on the performance of these two methods. For more detailed results, see [6].

Manuscript received February 20, 1997; revised January 7, 1998. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. W. Bastiaan Kleijn.

V. Ramasubramanian is with the Computer Systems and Communications Group, Tata Institute of Fundamental Research, Bombay 400005, India.

K. K. Paliwal is with the School of Microelectronic Engineering, Griffith University, Brisbane, Queensland. 4111, Australia (e-mail: k.paliwal@me.gu.edu.au).

Publisher Item Identifier S 1063-6676(99)01622-3.

II. Voronoi Projection Based Fast Search

The Voronoi region V_i associated with a codevector $\mathbf{c}_i$ contains all points in the space $\mathcal{R}^K$ nearer to $\mathbf{c}_i$ than any other codevector. If a test vector $\mathbf{x}$ is contained in a Voronoi region $\mathbf{V}_i$, the associated codevector $\mathbf{c}_i$ will be its nearest neighbor. The projection of the Voronoi region 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

$$P_{i,L}^j = \min_{\mathbf{x} \in V_i} x_j, \qquad P_{i,U}^j = \max_{\mathbf{x} \in V_i} x_j. \tag{1}$$

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

$$B_i = \left\{\mathbf{x} : (P_{i,L}^j \leq x_j \leq P_{i,U}^j), \qquad j = 1, \cdots, K\right\}. \tag{2}$$

Each codevector has such an 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.

From (1) and (2), $V_i \subseteq B_i$. Hence, if $\mathbf{x} \in V_i$, then $\mathbf{x} \in B_i$. Thus, if $\mathbf{x} \in B_i$, the corresponding Voronoi region V_i may contain $\mathbf{x}$, and $\mathbf{c}_i$ can be the nearest-neighbor of $\mathbf{x}$. 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. This results in a set of candidate codevectors, $C'(\mathbf{x}) = \{\mathbf{c}_i : \ \mathbf{x} \in B_i\}$, which contains the actual nearest-neighbor that can be determined by a full-search of this candidate set. 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. The Voronoi-projection based fast-search algorithm thus consists of the following 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 $\mathbf{x}$.
2) Perform a full-search in the candidate set $C'(\mathbf{x})$ to obtain the actual nearest-neighbor of $\mathbf{x}$.

The complexity of the Voronoi projection based search algorithm is mainly determined by the complexity of Step 2. Step 1 essentially contributes to the overhead computation. The complexity of Step 2 depends on the distribution of the codevectors and the test vectors, and is invariant to the method employed to perform Step 1. In this paper, we refer to the complexity of Step 2 as the main complexity of the Voronoi projection based fast-search algorithm, and the complexity of Step 1 as its overhead complexity. It is important to obtain efficient procedures for carrying out both of these steps in order to reduce the overall complexity of the algorithm.

In the following sections, we consider in detail two methods, which utilize the projection information, to carry out Step 1. These are 1)

BS method and 2) cell-partition search method. Note that these two methods differ only in terms of how Step 1 is carried out, Step 2 is identical for both of them.

A. Box-Search Method

This method is based on the observation that, if $\mathbf{x} \notin B_i$, then $\mathbf{x} \notin V_i$. This means that $\mathbf{c}_i$ cannot be the nearest-neighbor of $\mathbf{x}$. The BS performs a simple elimination test (*box-test*) for each codevector in a full-search structure before computing its distance to the test vector $\mathbf{x}$. A codevector is rejected when any of the test vector's coordinate 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 the test vector $\mathbf{x}$ passes the test on all the coordinates (when all the projections of the Voronoi region V_i contain the corresponding coordinates of the test vector), then the test vector lies inside the hypercuboid box B_i and the corresponding codevector $\mathbf{c}_i$ has to be considered for full-search.

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." 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.

Thus, the hypercuboid box B_i is used as a simple geometric object for point location. The set of candidate codevectors $C'(\mathbf{x})$ is determined by directly checking if the test vector lies within the box of each of the codevector. Thus

$$C'(\mathbf{x}) = S_{box}(\mathbf{x}) = \{\mathbf{c}_i\colon\ \mathbf{x} \in B_i\}. \tag{3}$$

The cost of BS for a test vector is $c(\mathbf{x}) = \sum_{i=1}^{N} c_i$, where, c_i is the number of comparisons incurred for a codevector $\mathbf{c}_i$ (with $1 \le c_i \le 2K$). The cost per vector-component (or coordinate) is $c(\mathbf{x})/K$ (with N/K as its lower bound and $2N$ as its upper bound). The quantity $c(\mathbf{x})/N$, on the other hand, gives the number of comparisons done *per codevector*. This can have a value of 1 to $2K$. These quantities, when obtained as an average over a large set of test vectors, characterize the overhead complexity of the search algorithm.

B. Cell-Partition Search Method

In this method, first studied by Cheng *et al.* [1], the Voronoi projection information is organized to form "cell-partitions" in a preprocessing stage. These cells are then used to generate the set of candidate codevectors for a given test vector.

The N overlapping projection intervals, $P^j_i = (P^j_{i,\,L},\ P^j_{i,\,U}),\ i = 1, \cdots, N$, generate $2N$ projection boundaries $(y^j_1, y^j_2, \cdots, y^j_{2N})$ with a natural ordering $(y^j_1 \le y^j_2 \le \cdots \le y^j_{2N})$. This partitions the jth coordinate axis into $(2N-1)$ contiguous intervals $\{I^j(1), I^j(2), \cdots, I^j(2N-1)\}$, where $I^j(m) = \{(y^j_m, y^j_{m+1})\}$. Each of these intervals $I^j(m)$ is associated with a set $S^j(m)$ of indices of the Voronoi regions whose projection intervals P^j_i partially or totally overlap with $I^j(m)$; i.e., $S^j(m) = \{i\colon P^j_{i,\,L} \le y^j_m$ and $P^j_{i,\,U} \ge y^j_{m+1}\}$.

The $(2N-1)$ contiguous intervals, $\{I^j(1), I^j(2), \cdots, I^j(2N-1)\}$, on each of the K coordinate axes partitions the space $\mathcal{R}^K$ into $(2N-1)^K$ hyperrectangular cells. Given a test vector $\mathbf{x} = (x_1, \cdots, x_K)$, let $I^j(m_j) = \{(y^j_{m_j}, y^j_{m_j+1})\}$ be the interval containing x_j; i.e., $y^j_{m_j} \le x_j \le y^j_{m_j+1}$. The set of indices $S^j(m_j)$ associated with $I^j(m_j)$ corresponds to the candidate codevectors which could be the nearest-neighbor of the test vector $\mathbf{x}$. Identification of the K intervals, $I^j(m_j),\ j = 1, \cdots, K$, containing the test vector $\mathbf{x}$ locates it in a hyperrectangular cell

$$H(\mathbf{x}) = \prod_{j=1}^{K} I^j(m_j). \tag{4}$$

The final set of candidate codevectors $C'(\mathbf{x})$ is obtained as[1]

$$S_{cell}(\mathbf{x}) = \bigcap_{j=1}^{K} S^j(m_j). \tag{5}$$

Equations (4) and (5) suggest two possible procedures for finding the set $C'(\mathbf{x})$ of candidate codevectors for the test vector $\mathbf{x}$. In the first procedure, the set of candidate codevectors associated with each of the $(2N-1)^K$ hyperrectangular cells are precomputed and stored in a table. The hyperrectangular cell that contains the test vector $\mathbf{x}$ is identified by using (4). The set of candidate codevectors associated with this cell, prestored in a table, is directly used as the final set of candidate codevectors $C'(\mathbf{x})$. A full-search within this small set of candidate codevectors then yields the nearest-neighbor of the test vector. This procedure requires only $K \log(2N)$ comparisons to find the final set of candidate codevectors. However, the problem with this procedure is that it requires an exorbitant storage of $(2N-1)^K\gamma$, where γ is the average size of the candidate set associated with a hyperrectangular cell.

In the second procedure, proposed by Cheng *et al.* [1], exponential storage complexity is reduced to linear storage complexity. Here, the sets of candidate codevectors, $[S^j(m),\ m = 1, \cdots, (2N-1)$ and $j = 1, \cdots, K]$, are precomputed and stored in K mapping tables. Each mapping table corresponds to one coordinate axis and contains $(2N-1)$ rows, one for each interval. Thus, this procedure requires a storage of $(2N-1)K\theta$, where θ is the average size of the candidate set associated with one interval. The final set of candidate codevectors is obtained by using (5), which requires the computation of the intersection of K sets. Cheng *et al.* [1] have used a procedure to compute this intersection operation, which has a complexity of about KN comparisons. In this work, we propose to use an alternate procedure which we refer to as *intersection count* (IC) procedure. We outline below the cell-partition search method using the intersection count procedure.

1) Find the interval $I^j_m = \{(y^j_m, y^j_{m+1})\}$ in each coordinate j containing the component x_j of the test vector; i.e., $y^j_m \le x_j \le y^j_{m+1}$.
2) Find the final set of candidate codevectors by computing the intersection of the K sets, $S^j(m_j),\ j = 1, \cdots, K$, using the intersection count procedure. Let $S^j(m_j) = \{i^j_1, i^j_2, \cdots, i^j_{\beta_j}\}$, where $\beta_j = |S^j(m_j)|$. The intersection count (IC) procedure is as follows:

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

[1]It can be easily seen that the hyperrectangular cell $H(\mathbf{x})$ is contained within the volume of intersection of the Voronoi boxes containing the test vector $\mathbf{x}$. As a result, the sets of candidate codevectors, $S_{box}(\mathbf{x})$ obtained from (3) and $S_{cell}(\mathbf{x})$ obtained from (5), are identical.

Note that the intersection count procedure is a variant of the shift method employed by Yunck [7] and has been introduced earlier in a related data structure [4]. The complexity of the IC-based cell-partition search for one test vector is $[K \log(2N) + \delta]$ operations, where δ is the complexity of the intersection count procedure. It can be easily seen that $\delta = (N + \sum_{j=1}^{K} \beta_j)$, where $\beta_j = |S^j(m_j)|$ (with bounds $1 \leq \beta_j \leq N$). Thus, the lower and upper complexity bounds for the IC-based cell-partition search are $[K \log(2N) + (K + N)]$ and $[K \log(2N) + (K + 1)N]$.

III. Results and Discussion

In this section, we present simulation results obtained in the context of vector quantization of speech waveform. These results characterize the complexity of the BS and the cell-partition search methods.

First, we describe the means of obtaining the Voronoi projections of a given set of codevectors. The projection boundary points are 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 obtained by encoding the points that fall within the region are as follows:

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

where R represents the training set (or the domain of interest in the R^K space in the case of uniformly distributed points). 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.[2] All results shown here were obtained using 50 000 vectors picked randomly from the data used for estimating the Voronoi projections.

A. Main Complexity

Here, we give the main complexity of the Voronoi projection based fast-search methods in terms of the number of distances computed. This is provided by the size of the candidate set of codevectors obtained by the BS or the IC-based cell-partition search method.[3] Fig. 1 shows the average and maximum number of codevectors searched by the fast-search methods for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. Fig. 2 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 methods in terms of vector distances computed per vector. The excellent complexity reduction of the Voronoi projection based fast-search methods with respect to full-search can be noted in all these cases.

[2]The projection estimates obtained by this method are approximate. If the training data used is not sufficient to obtain accurate estimates of the projections, then the search with the test data (different from training data) can incur errors in the nearest-neighbor obtained. However, these errors cause a marginal and graceful degradation in the signal-to-noise ratio of the encoded test-data (resulting from the determination of second or third nearest-neighbors). This has been reported in our earlier work on K-d tree structures optimized with respect to bucket-Voronoi intersections [5].

[3]As discussed earlier, the box-search and the cell-partition search methods provide identical sets of candidate codevectors. Therefore, their main complexity is same.

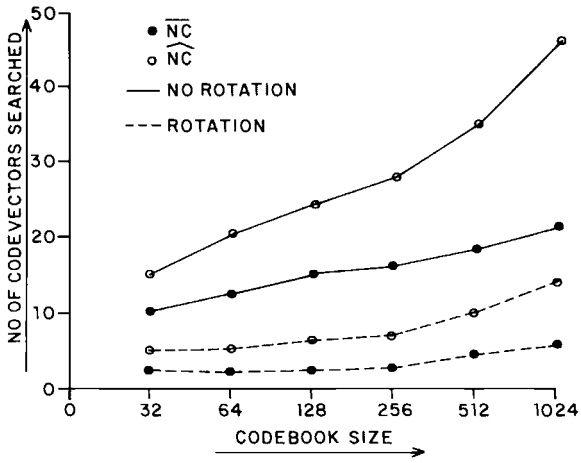


Fig. 1. Average and maximum number of codevectors searched ($\overline{NC}$, $\widehat{NC}$) by the projection-based search methods (BS/IC-based cell-partition search methods) for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

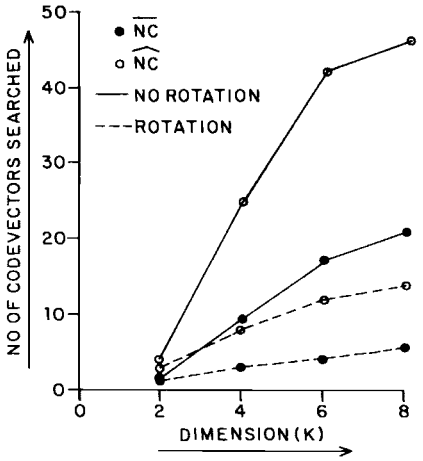
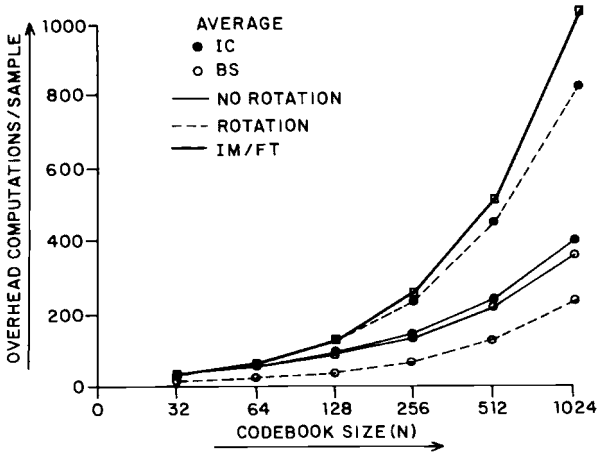


Fig. 2. Average and maximum number of codevectors searched ($\overline{NC}$, $\widehat{NC}$) the projection-based search methods (BS/IC-based cell-partition search methods) for dimensions $K = 2, 4, 6, 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

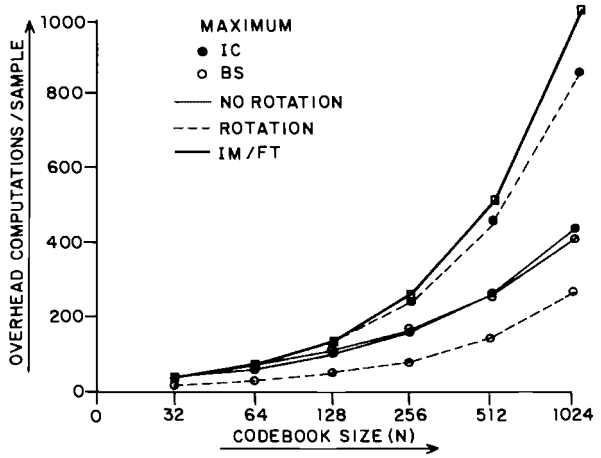
B. Overhead Complexity

Fig. 3(a) and (b), respectively, shows the average and maximum costs per vector-component incurred by the IC-based cell-partition search and the BS methods for dimension $K = 8$ and different codebook sizes $N = 32, 64, 128, 256, 512$, and 1024. The BS method can be seen to have a smaller (but comparable) overhead cost than the IC-based cell-partition search method.

In Fig. 4(a) and (b), we show the above-mentioned 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 the IC-based cell-partition search method increases with dimension, that of the BS method shows a marked decrease. This is due to the fact that in the case of the IC-based cell-partition search method, 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 the BS method, as the dimension increases, there is an increase in

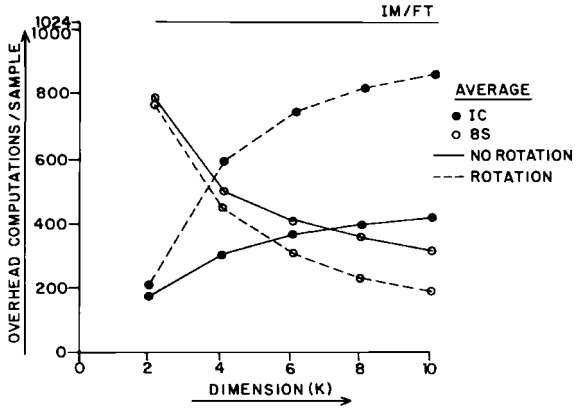


(a)

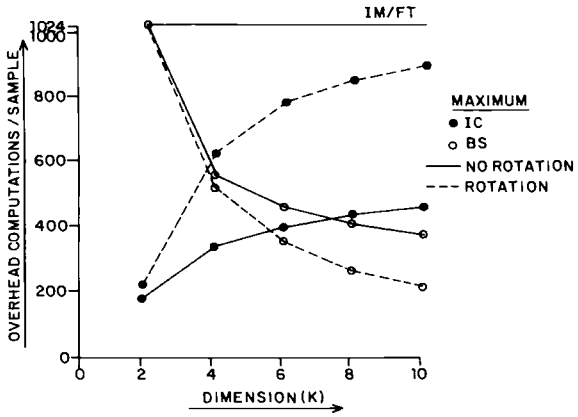


(b)

Fig. 3. (a) Average and (b) maximum overhead computations/vector-component of the IC-based cell-partition search and BS methods 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.



(a)



(b)

Fig. 4. (a) Average and (b) maximum overhead computations/vector-component of the IC-based cell-partition search and BS methods for dimension $K = 2, 4, 6, 8, 10$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

the degree of freedom for a box to *not contain* a given test vector. This reduces average number of comparisons required in the box-test before a codevector can be rejected. We have also shown in Figs. 3 and 4 the cost of the Cheng *et al.*'s method [1] (plot marked as IM/FT for "image-map full-table" storage). It can be noted from Figs. 3 and 4 that the overhead cost of the IC-based cell-partition search method proposed here is considerably less than the method of Cheng *et al.* [1].

In Figs. 5 and 6, we show the average number of comparisons required per codevector in the box search. These figures illustrate how quickly a codevector is eliminated with respect to the maximum $2K$ comparisons required in the worst-case to determine if a point lies within a box. Fig. 5 shows this for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512,$ and 1024, and Fig. 6 for codebook size $N = 1024$ and dimensions $K = 2, 4, 6, 8,$ and 10. In both these 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 dimensions and codebook sizes asymptotically saturates to a value of 2, placing the empirical cost of BS per vector-component to be $2N/K$. This is significantly less than the cost of the method of Cheng *et al.* [1], especially for large dimensions.

Fig. 7 compares the total storage required by the IC-based cell-partition search method with that of the BS method for dimension

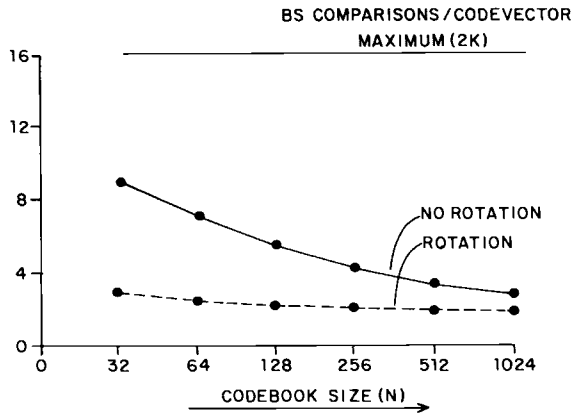


Fig. 5. Average number of overhead comparisons per codevector resulting from the BS method for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512,$, and 1024. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

$K = 8$ 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 (bit-map full-table storage) ($(3 + N/b)NK$ with $b = 32$) used in [1].

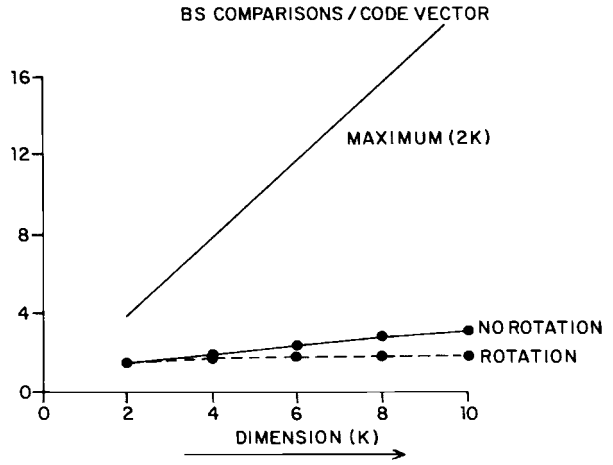


Fig. 6. Average number of overhead comparisons per codevector resulting from the BS method for dimension $K = 2, 4, 6, 8, 10$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

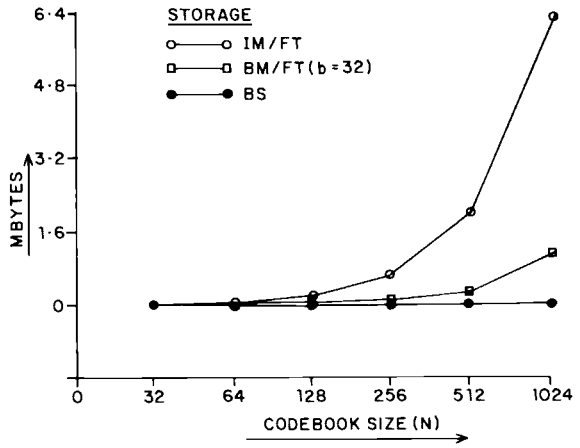


Fig. 7. Total storage required by the cell-partition search method (using IM/FT and BM/FT) and the BS method for dimension $K = 8$ and codebook sizes $N = 32, 64, 128, 256, 512$, and 1024.

C. Effect of Principal Component Rotation

In this section, we study the influence of principal component rotation [2] in reducing the complexity of different search methods using the Voronoi projections. In Figs. 1 and 2, we show the number of codevectors examined (final candidate set size) for various dimensions and codebook sizes with and without principal component rotation. The rotated case can be seen to have a consistently lower complexity than the unrotated case. This happens because 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. This decreases the size of the final set of candidate codevectors, thereby reducing the main complexity of the projection based search methods.

Figs. 3 and 4 show the effect of principal component rotation on the computational overhead of the two methods. It is clear from these figures that the principal component rotation increases storage and computational overheads for the IC-based cell-partition search method. This happens due to the fact that the rotation causes an increase in the average index set size. In comparison, these figures show that the overhead cost resulting from the BS method is reduced due to rotation. This can also be noted in Figs. 5 and 6, which show the box-test comparisons per codevector.

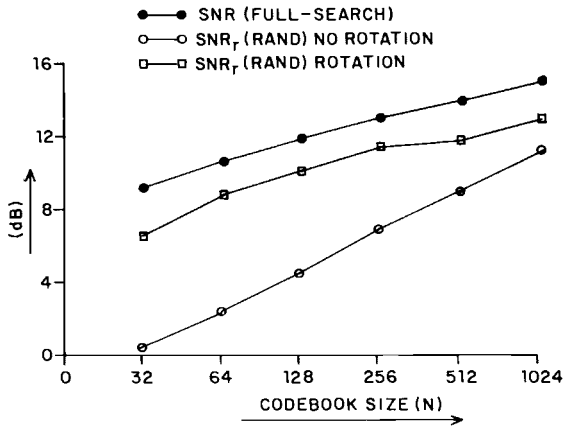


Fig. 8. SNR for full-search and for encoding by random selection (SNR$_{rand}$) from final candidate set in projection method for dimension $K = 8$ and codebook size $N = 1024$. *No Rotation:* without principal component rotation; *Rotation:* with principal component rotation.

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 is quantized by selecting the final codevector *randomly* from the final candidate set. Fig. 8 compares the full-search SNR and the "random selection" SNR (denoted by SNR$_r$) for the rotated and unrotated cases 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. Also, rotation makes SNR$_r$ closer to the full-search SNR for all the codebook sizes.

D. Overall Complexity of the Projection-Based Fast-Search Methods

Here, we compare the IC-based cell-partition search method and the BS method (with and without the use of rotation) in terms of their computational complexity and storage requirements. We list in Table I their main computational complexity, storage and overhead computational cost for dimension $K = 8$ and codebook size $N = 1024$ In order to put these methods in proper perspective, we also provide here the complexity of the exhaustive full-search method. The following can be noted from this table:

1) The IC-based cell-partition and the box search methods have the same complexity in terms of the average and maximum number of codevectors examined, $\overline{NC}$ and $\widehat{NC}$.
2) The IC-based cell-partition search method has an exorbitant storage complexity in comparison to the BS method.
3) The overhead costs of the IC-based cell-partition search method is roughly comparable to that of the BS method for the unrotated case. Principal component rotation increases the overhead cost of the IC-based cell-partition method by almost a factor of 2. The BS method shows a significant decrease in overhead cost due to rotation. The storage requirement of the IC-based cell-partition search method also nearly doubles up due to the principal component 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 [1]. The BS method with rotation can be seen to have the lowest complexity and offers excellent reduction over the full-search complexity.

TABLE I
COMPARISON OF THE IC-BASED CELL-PARTITION SEARCH AND BOX-SEARCH (BS) WITH FULL-SEARCH. DIMENSION $K = 8$. CODEBOOK SIZE $N = 1024$. DATA: 50 000 VECTORS OF SPEECH WAVEFORM

Algorithm	Main Complexity		Overhead Computation		Total storage	m	a	c
	$\overline{NC}$	$\widehat{NC}$	ave	max				
IC (NR)	21.1	46	399.3	436.0	1596447	21.1	39.6	401.8
IC (R)	5.6	14	822.1	855.0	3120534	13.6	10.5	824.6
BS (NR)	21.1	46	359.4	411.1	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

(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.

IV. CONCLUSIONS

In this work, we have provided a detailed study of two fast-search methods using the Voronoi projection information, namely, the box-search method and the cell-partition search method. In the cell-partition search 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.* [1]. In addition, we have proposed the box-search method as an alternative to the cell-partition search method. This method is based on direct and simple interpretation of the projection information. The box-search method requires significantly less storage than the intersection count-based cell-partition search method. We have also investigated the effect of principal component rotation and found that it reduces the complexity of Voronoi projection based search significantly.

REFERENCES

[1] D. Y. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Mar. 1984, vol. 1, pp. 9.11.1–9.11.4.

[2] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

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

[4] V. Ramasubramanian and K. K. Paliwal, "An efficient approximation-elimination algorithm for fast nearest-neighbor search," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Mar. 1992, vol. 1, pp. 89–92.

[5] ——, "Fast K-d tree algorithms for nearest-neighbor search with application to vector quantization encoding," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 40, pp. 518–531, Mar. 1992.

[6] ——, "Voronoi projection-based fast nearest-neighbor search algorithms: Box-search and mapping table-based search techniques," *Digital Signal Process.*, vol. 7, pp. 260–277, Oct. 1997.

[7] T. P. Yunck, "A technique to identify nearest-neighbors," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 678–683, Oct. 1976.

On Using Formants to Improve SCHMM Speaker Adaptation

Tae-Young Yang, Won-Ho Shin, Weon-Goo Kim, Dae Hee Youn, and Il-Whan Cha

***Abstract*—A speaker adaptation algorithm using formant frequencies is proposed. The formants extracted from the cepstral means in the reference codebook are iteratively shifted toward the formants of a test speaker. The number of cepstral means selected at each iteration decreases as the iteration increases. Decision of the number of selected cepstral means and a formant based distance measure are formulated. The proposed algorithm was implemented in two schemes and evaluated by a speaker-independent, a male speaker-dependent, and a female speaker-dependent recognition experiments. A combined scheme with the Bayesian adaptation obtained 9.7% enhancement for the average recognition accuracy in speaker-independent experiments and 52.6% in speaker-dependent recognition experiments.**

***Index Terms*—Formant shifting, semicontinuous hidden Markov model, speaker adaptation.**

I. INTRODUCTION

We consider a speaker adaptation process for a semicontinuous hidden Markov model-based (SCHMM-based) speech recognizer. In speaker-adaptive recognizers, the recognition unit models are modified to represent the acoustic characteristics of a test speaker using a small amount of adaptation speech data.

Several approaches have been proposed for speaker adaptation. The spectrum of a test speaker is mapped onto the target spectral space by mapping matrices in the spectral mapping technique [1]. In the speaker Markov model approach, the spectral differences between a test speaker and the acoustic features in the recognizer are modeled by the speaker Markov model [2]. The Bayesian adaptation method is based on maximum *a posteriori* estimation of hidden Markov model parameters [3], [4].

However, the performance of the speaker adaptation degrades when there is a large acoustic mismatch between reference and test speakers. To solve such a problem, we use the formants extracted from cepstral coefficients. Even though it is difficult to use the formants directly as a feature of speech recognition, they certainly contain helpful information for speaker adaptation. Due to various vocal tract shapes and lengths, the formant frequencies vary considerably among speakers, but they are distributed in similar shape (e.g., vowel triangle). Thus, we can reduce the acoustic mismatch by shifting the formants.

In our method, the cepstral means in the reference codebook are adapted by shifting their formants iteratively. At each iteration, one speech frame is randomly chosen from the adaptation data, and the cepstral means that have similar formants to those of the adaptation speech frame are selected. The formants of the selected cepstral means are shifted toward the formants of the adaptation speech frame. The number of selected cepstral means decreases as

Manuscript received April 8, 1996; revised April 10, 1998. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Mazin Rahim.

T.-Y. Yang, W.-H. Shin, D. H. Youn, and I.-W. Cha are with the Department of Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: tyyang@ghost.yonsei.ac.kr).

W.-G. Kim is with the Department of Electrical Engineering, Kunsan National University, Kunsan City, Chonbook 573-360, Korea.

Publisher Item Identifier S 1063-6676(99)01631-4.