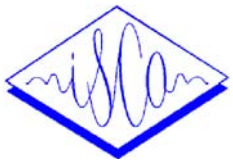

EUROSPEECH '97
5th European Conference on Speech Communication and Technology
Rhodes, Greece, September 22-25, 1997

REDUCING THE COMPLEXITY OF THE LPC VECTOR QUANTIZER USING THE K-D TREE SEARCH ALGORITHM

V. Ramasubramanian and K. K. Paliwal

ATR Interpreting Telecommunications Res. Labs.
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

10.21437/Eurospeech.1997-186

ABSTRACT

Linear predictive coding (LPC) parameters are widely used in various speech coding applications for representing the spectral envelope information of speech. Transparent quantization of the LPC parameters (average spectral distortion of 1 dB) can be achieved at 24 bits/frame using the split vector LPC quantizer (SVLPC) which quantizes 10-dimensional line spectral frequency (LSF) vectors in two parts. However, SVLPC suffers from a high computational complexity in quantizing each part (one of dimension 4 and the other of dimension 6) using independent codebooks of size 4096 (corresponding to a rate of 12 bits/part). This limits the practical real-time application of the coder. In this paper, we reduce the computational complexity of the split vector quantizer by 2 orders of magnitude using the fast K-dimensional (K-d) tree search algorithm under the bucket-Voronoi intersection (BVI) search framework. This is of significant importance in rendering the SVLPC amenable for practical real-time coding applications.

1. INTRODUCTION

Linear predictive coding (LPC) parameters are widely used in various speech coding applications for representing the spectral envelope information of speech. For low bit rate speech coding applications, it is important to quantize these parameters using as few bits as possible. Considerable work has been done in the past to develop efficient quantization procedures, both scalar and vector, for quantizing the LPC parameters with smallest number of bits. Among these, vector quantization of the LPC parameters has emerged as an effective approach to achieve 'transparent' quantization (average spectral distortion of 1 dB or less) of the LPC parameters. Currently, the lowest bit-rate for transparent quantization is achieved by the split vector quantizer using line spectral frequencies (LSF) [1]. This quantizer requires 24 bits/frame to achieve an average spectral distortion of 1 dB, less than 2% frames having spectral distortion in the range 2-4 dB and no frame having spectral distortion greater than 4 dB. Detailed studies supporting the choice of LSF representation over other LPC representation, the need to resort to splitting of the LSF vector and the distortion measure used for vector quantization have been reported earlier [1]. Following is a brief description of the basic vector quantization encoding stage in the SVLPC quantizer.

In a typical LPC based quantization scheme, LPC parameters are obtained at a rate of 50 frames/sec, using the 10-th order LPC analysis and are quantized prior to transmission. In the 24 bits/frame split vector LPC quantizer (SVLPC) [1], each LPC parameter vector is transformed to the corresponding 10-dimensional LSF vector. This LSF vector is divided into two parts — the first part consists of the first four LSFs and the second part the last six LSFs. Each part is quantized independently using 12 bits. If the LSF's of a frame representing a short segment of speech is given by the vector $\mathbf{f} = (f_1, f_2, \ldots, f_{10})$, this is quantized into a vector $\hat{\mathbf{f}} = (\hat{f}_1, \ldots, \hat{f}_4, \hat{f}_5, \ldots, \hat{f}_{10})$, where, $(\hat{f}_1, \ldots, \hat{f}_4)$ and $(\hat{f}_5, \ldots, \hat{f}_{10})$ are respectively the nearest-neighbor codevectors of the first part $(f_1, \ldots, f_4)$ and the second part $(f_5, \ldots, f_{10})$ from the corresponding codebooks of the two parts each of size 4096. The distance measure $d(\mathbf{f}, \hat{\mathbf{f}})$ between the LSF test vector $\mathbf{f}$ and a LSF codevector $\hat{\mathbf{f}}$ is the squared error distance given by $d(\mathbf{f}, \hat{\mathbf{f}}) = \sum_{i=1}^{10} [(f_i - \hat{f}_i)]^2$, where f_i and $\hat{f}_i$ are the i-th LSFs in the test and codevector respectively. The LPC quantization of one frame of speech corresponds to vector quantization encoding of each part using a codebook of size 4096. This requires computation of 4096 distances for each part and is a very high computational requirement for vector quantization (VQ) encoding which limits the practical real-time application of the coder.

In this paper, we are concerned with the computational complexity of the SVLPC quantizer. The main aim in this paper is to report the results of applying an efficient fast nearest-neighbor search algorithm to reduce the computational complexity of the split vector quantizer. The fast algorithm is a K-dimensional (K-d) tree search algorithm under the bucket-Voronoi intersection (BVI) search framework [2]. This paper is organized as follows. In Section 2, we give a brief description of the K-d tree data structure. In Section 3, we describe the bucket-Voronoi intersection (BVI) framework. Section 4 describes the preprocessing procedure in constructing the K-d tree for using in the BVI framework. This involves optimizing the tree using efficient criteria and finding the bucket-Voronoi intersections. In this section we describe two optimization criteria used for the simulation study in this paper. In Section 5, we present results obtained in reducing the computational complexity of the SVLPC quantizer for the two parts (one of dimension 4 and the other of dimension 6) under the squared error distance.

2. THE K-DIMENSIONAL (K-D) TREE

An important approach towards fast VQ encoding is the use of data structures which facilitate fast search of the codebook which is normally unstructured. In this context, the K-d (K-dimensional) tree structure is a powerful structure in providing efficient space localization of a vector in K-dimensional space ($\mathcal{R}^K$) with very low overheads: A K-d tree structure of depth d partitions the $\mathcal{R}^K$ space into 2^d disjoint hypercuboidal regions (buckets) and allows identification of the bucket containing any given vector in $\mathcal{R}^K$ in d scalar comparisons.

The $\mathcal{R}^K$ space is split into two half spaces by means of an hyperplane orthogonal to one of the K coordinate axes. Such an hyperplane $\mathbf{H}$, given by $\mathbf{H} = \{\mathbf{x} \in \mathcal{R}^K : x_j = h\}$, defines two half spaces, R_L and R_R as $R_L = \{\mathbf{x} \in \mathcal{R}^K : x_j \leq h\}$ and $R_R = \{\mathbf{x} \in \mathcal{R}^K : x_j > h\}$. The initial region corresponds to the root of the tree at layer 1 and the two subregions R_L and R_R obtained by the division correspond to the left and right sons at layer 2. Each of these two half spaces are successively divided by hyperplanes orthogonal to the coordinate axes and d such successive divisions starting with the initial region as the root at layer 1 creates a tree of depth d with 2^d terminal regions termed 'buckets' at the $(d+1)^{th}$ layer.

Every non-terminal node is associated with a region and a partitioning hyperplane of the form $\mathbf{x} : x_j = h$, which needs storage of two scalar quantities (j, h) at each node, where j is the index of the coordinate axis orthogonal to the plane (referred to as the 'partitioning' or 'discriminator' axis), and, h is the location of the plane on this axis. Given any vector in $\mathcal{R}^K$, it can be located with respect to the dividing plane $\mathbf{H}$ at any node by the scalar comparison $x_j \leq h$, i.e., comparing the vector's j^{th} component value with the partition value h. Starting from the root node, a sequence of d such scalar comparisons of the vector's j^{th} component value with the partitioning hyperplane (j, h) at that node leads to the leaf (or bucket) containing the vector.

3. BUCKET-VORONOI INTERSECTION SEARCH

Given a set of N codevectors $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$, alongwith a specified distance measure, the $\mathcal{R}^K$ space is partitioned into N disjoint regions $\{V_1, V_2, \ldots, V_N\}$, known as *Voronoi regions*. V_i is the Voronoi region of $\mathbf{c}_i$ and contains all points in $\mathcal{R}^K$ nearer to $\mathbf{c}_i$ than any other codevector, i.e., $V_i = \{\mathbf{x} \in \mathcal{R}^K : d(\mathbf{x}, \mathbf{c}_i) \leq d(\mathbf{x}, \mathbf{c}_j), j = 1, \ldots, N\}$. Thus, if $\mathbf{q}(\mathbf{x})$ denotes the nearest-neighbor quantization of $\mathbf{x}$ then, $V_i = \{\mathbf{x} \in \mathcal{R}^K : \mathbf{q}(\mathbf{x}) = \mathbf{c}_i\}$ is the nearest-neighbor locus region of $\mathbf{c}_i$. If a vector $\mathbf{x}$ in $\mathcal{R}^K$ is contained in the Voronoi region $\mathbf{V}_i$, the associated codevector $\mathbf{c}_i$ will be the nearest-neighbor of $\mathbf{x}$.

In the bucket-Voronoi intersection search framework, each bucket is associated with a set of codevectors whose Voronoi region intersect with the bucket region. This subset is referred to as the bucket-Voronoi intersection list (or BVI-list) of that bucket. The buckets and the Voronoi regions provide two independent partitioning of the same space using disjoint regions. Therefore, if the test vector is contained within the bounds of a particular bucket region, then the test vector can be present only in one of the Voronoi regions having a non-empty intersection with the bucket. Thus, the BVI-list of a bucket contains the nearest-neighbor codevector of any vector within the bucket. For any given test vector, its nearest-neighbor can therefore be determined by a fast and localized BVI search in two steps:

1. Determine the bucket containing the test vector. This requires d scalar comparisons for a tree of depth d.

2. Perform a search (by actual distance computation) among the small set of codevectors whose indices are stored in the BVI - list associated with that bucket.

4. PRE-PROCESSING

The BVI search requires a pre-processing phase consisting of i) Constructing the K-d tree for a given set of codevectors and, ii) Finding the BVI list for each bucket of the resulting tree. Subsequent to this, the K-d tree structure can be used for fast encoding of arbitrary new test vectors. The following gives a brief description of the steps involved in this pre-processing phase.

4.1. Construction of the K-d tree

The construction of the K-d tree involves the choice of the partitioning hyperplane at each node of the tree. This is referred to as optimization of the tree for the given set of codevectors. A non-terminal node region in the K-d tree is associated with a bounded region $R \in \mathcal{R}^K$, defined as $R = \{\mathbf{x} \in \mathcal{R}^K : a_j \leq x_j \leq b_j, \quad j = 1, \ldots, K\}$. As noted earlier, a hyperplane $\{\mathbf{x} : x_j = h;\ a_j \leq h \leq b_j\}$ (represented as (j, h) henceforth), divides R into two subregions R_L and R_R. Optimization of the K-d tree involves making a choice of the partitioning hyperplane $(j^*, h^*_{j^*}), j^* \in \{1, \ldots, K\}$ and $a_{j^*} \leq h^*_{j^*} \leq b_{j^*}$ under some criterion with respect to region R. This is termed *local* optimization of the tree and has been addressed earlier in detail in [2]. Here, we consider two main optimization criteria in designing the tree for use with the BVI search. These are namely, i) the *Friedman-Bentley-Finkel* (FBF) criterion [3], [4], and, ii) the *generalized optimization criterion* (GOC) [2].

4.1.1. Friedman-Bentley-Finkel (FBF) criterion

If $\mathbf{C}_R = \{\mathbf{c}_i : \mathbf{c}_i \in R\}$ is the set of codevectors lying within the region R represented by the node to be partitioned, the FBF criterion chooses the partitioning hyperplane $(j^*, h^*_{j^*})$ as follows:

1. j^* is chosen as the the axis along which the corresponding codevector coordinates have the maximum variance and,

2. $h^*_{j^*}$ is chosen as the median of the codevector coordinate distribution on axis j^*,

where, the codevector coordinate variance and median are computed using codevectors in $\mathbf{C}_R$.

4.1.2. Generalized optimization criterion (GOC)

The FBF criterion was originally obtained for using the K-d tree under a backtracking search [3] and hence optimizes the tree only with respect to the codevectors. However, for the bucket-Voronoi intersection search, a more direct optimization results with the use of information about the Voronoi regions intersecting with the region corresponding to the node being optimized [2]. One of the optimization criteria proposed under this framework is the *generalized optimization criterion* (GOC) which is briefly described in the following:

Let $\mathbf{C}_R$, $\mathbf{C}_{R_L}$ and $\mathbf{C}_{R_R}$ be the set of codevectors whose corresponding Voronoi regions intersect with the regions R, R_L and R_R respectively, i.e., $\mathbf{C}_R = \{\mathbf{c}_i : V_i \cap R \neq \emptyset\}$, $\mathbf{C}_{R_L} = \{\mathbf{c}_i : V_i \cap R_L \neq \emptyset\}$, $\mathbf{C}_{R_R} = \{\mathbf{c}_i : V_i \cap R_R \neq \emptyset\}$, where V_i is the Voronoi region associated with codevector $\mathbf{c}_i$. Let n, n_L and n_R be the size of the sets $\mathbf{C}_R$, $\mathbf{C}_{R_L}$ and $\mathbf{C}_{R_R}$ respectively.

Given $\mathbf{x} \in R$, the nearest-neighbor of $\mathbf{x}$ belongs to $\mathbf{C}_R$ and can be found with a cost of n distance computations. However, the partitioning of R reduces the search

complexity from n to n_L or n_R after one scalar comparison. The Voronoi intersection numbers n_L and n_R for a given division (j,h) of R determine the complexity reduction that can be achieved for $\mathbf{x} \in R$. The *generalized optimization criterion* (GOC) chooses the optimal partitioning hyperplane $(j^*, h^*_{j^*})$ such that n_L and n_R are as small as possible. GOC performs this in two steps:

1. Find the optimal partition location h^*_j on each coordinate axis $j = 1, \ldots, K$ as the partition which minimizes $|n_L(j,h) - n_R(j,h)|$:

$$h^*_j = \arg \min_{a_j \le h \le b_j} |n_L(j,h) - n_R(j,h)| \qquad (1)$$

2. Find the optimal partition axis j^* as the coordinate axis for which its optimal (n_L, n_R) division at h^*_j is closest to the balanced division value $(n/2, n/2)$, i.e., which minimizes the Euclidean distance $d(\mathbf{p}, \mathbf{q})$ between $\mathbf{p} = (n_L(h^*_j), n_R(h^*_j))$ and $\mathbf{q} = (n/2, n/2)$, where n is the number of Voronoi regions intersecting with the region to be partitioned:

$$j^* = \arg \min_{1 \le j \le K} d(\mathbf{p}, \mathbf{q}) \qquad (2)$$

The main information required by the GOC are n, and $(n_L(j,h), n_R(j,h))$ (the number of Voronoi regions intersecting respectively with the left and right regions for a candidate division h on coordinate axis j) for any (j,h). These are obtained using the projections of the Voronoi intersection regions in R corresponding to the codevectors in $\mathbf{C}_R$. If $(V_i)_R$ is the intersection of Voronoi region V_i inside the region R, i.e., $(V_i)_R = V_i \cap R$, then $P^j_i = (P^j_{i,L}, P^j_{i,U})$ is the projection of $(V_i)_R$ on coordinate axis j, $P^j_{i,L}$ and $P^j_{i,U}$ being the lower and upper boundaries of the projection interval P^j_i. $\mathbf{C}_R$ is obtained using a large set of training vectors $T = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ (or uniformly distributed set of vectors inside R) as,

$$\mathbf{C}_R = \{\mathbf{c}_i : V_i \cap R \neq \emptyset\} = \bigcup_{\mathbf{x}_l \in R} \mathbf{q}(\mathbf{x}_l) \qquad (3)$$

where, $\mathbf{q}(\mathbf{x}_l)$ is the nearest-neighbor codevector of $\mathbf{x}_l$ in the codebook. Subsequent to the determination of $\mathbf{C}_R$, the projection estimates P^j_i for $j = 1, \ldots, K$ of each $\mathbf{c}_i \in \mathbf{C}_R$ is obtained from $T = \{\mathbf{x}_l\}, l = 1, \ldots, m$ as,

$$\begin{aligned} P^j_{i,L} &= \min_{\mathbf{x}_l \in R} x_j : \mathbf{c}_i = \mathbf{q}(\mathbf{x}_l) \\ P^j_{i,U} &= \max_{\mathbf{x}_l \in R} x_j : \mathbf{c}_i = \mathbf{q}(\mathbf{x}_l) \end{aligned} \qquad (4)$$

Given $P^j_i = (P^j_{i,L}, P^j_{i,U}), j = 1, \ldots, K$, $n_L(j,h)$ and $n_R(j,h)$ are obtained for any (j,h) as,

$$\begin{aligned} n_L(j,h) &= |S^j_L(h)|, \ \{S^j_L(h) = i : P^j_{i,L} < h\} \\ n_R(j,h) &= |S^j_R(h)|, \ \{S^j_R(h) = i : P^j_{i,U} > h\} \end{aligned} \qquad (5)$$

In the first step of GOC (1), the function $|n_L(j,h) - n_R(j,h)|$ changes only at the projection boundaries $(P^j_{i,L}, P^j_{i,U}), i = 1, \ldots, n$. Hence it has to be evaluated only at these $2n$ locations as the candidates for the optimal h^*. If the projection boundaries $(P^j_{i,L}, P^j_{i,U}), i = 1, \ldots, n$ are distinct, then $|n_L(j,h) - n_R(j,h)|$ has a minimum at the n^{th} interval among the $2n-1$ intervals formed by the ordered sequence $(P^j_{i,L}, P^j_{i,U})$, $i = 1, \ldots, n$. In this case, h^* can be located at the n^{th} interval directly, incurring only the $O(2n \log(2n))$ cost for ordering the $2n$ projection boundaries.

4.2. Bucket-Voronoi intersection (BVI) list, storage overhead and complexity of BVI search

4.2.1. Bucket-Voronoi intersection (BVI) list

In order to use the K-d tree (subsequent to its optimization), for bucket-Voronoi intersection based search, it is necessary to find the set of codevectors associated with each bucket corresponding to the Voronoi regions intersecting with the bucket. This set (henceforth, referred to as bucket-Voronoi Intersection list or BVI list) can be determined after the optimization (by FBF or GOC) is completed. This is done in the same manner as finding $\mathbf{C}_R$ using (3) from the training data T. This requires a total cost of $O(m \log N)$ scalar operations to find the bucket (at depth $d = \log N$) containing each test vector in T updating the corresponding bucket-Voronoi intersection list.

4.2.2. Storage

The storage for BVI search consists of representing the tree using two scalar quantities (j,h) per non-terminal node to describe the partitions along with the BVI list indices of the terminal bucket nodes. For a tree of depth d, the total storage is $(2^d - 1)$ integer words for the partition index j, $(2^d - 1)$ real words for the partition location h, and $(\overline{b}+1)2^d$ integer words for the the BVI list indices at the terminal bucket nodes. For tree depth $d = \log N$ this amounts to a storage equivalent of $(3+\overline{b})2^d$ integer words, where $\overline{b}$ is the average bucket size. $\overline{b}$ has been empirically observed to approach a constant $O(1)$, i.e., independent of N for tree depth $d = \log N$, and the total storage of the K-d tree for BVI search is linear $(O(N))$.

4.2.3. Complexity of BVI search

Given the tree and the BVI list, the nearest-neighbor search for any new test vector proceeds in two phases — first identifying the bucket containing the test vector and, then, searching within the list of codevectors associated with the bucket. The resulting solution will be optimal if the bucket-Voronoi intersection list of all the buckets have been obtained correctly. The BVI search complexity is directly determined by the size of the BVI lists in the tree. The size of the bucket regions decreases linearly for increasing tree depth. This results in a decrease in the number of Voronoi regions which intersect with the bucket (size of the BVI list) which inturn reduces the BVI search complexity linearly with tree depth. The tree depth can be increased indefinitely, with the search complexity decreasing monotonically, limited only by the storage requirements.

For searching a set of N codevectors, a properly optimized tree of depth $d = \log N$ can achieve $O(1)$ (constant, i.e., independent of the codebook size N) complexity reduction with linear $(O(N))$ storage and $O(\log N)$ memory access overhead cost [2]. In the experiments carried out here, we have limited our tree depth d to $\log N$ and the tree is uniform with N buckets at the terminal layer.

5. RESULTS

Here, we present simulation results using the K-d tree based BVI search for reducing the complexity of SVLPC quantization of 10 - dimensional LSF vectors with the squared error distance. We have two codebooks $\mathbf{C}_4$ and $\mathbf{C}_6$, respectively for the first part of dimension 4 and the second part of dimension 6, each of size $N = 4096$. The

data and codebooks used here are the same as used in [1]. The codebooks were obtained using a *training* set of 60000 LSF vectors. The *test* set consists of 8000 LSF vectors [1].

In order to apply the BVI search to the split-vector quantizer, the K-d tree is constructed first using the given codevectors for the two parts separately. Since the codebook size N in this case is 4096, the tree depth used here is $d = \log N = 12$ such that there are 4096 ($= 2^d$) buckets in the tree. Here, we report results for trees optimized using the FBF and GOC criteria. For the GOC criteria, 60000 vectors are not adequate in estimating the Voronoi projections for codebook size 4096 in dimensions $K = 4$ and $K = 6$. Therefore, we have increased the training data set to 1,200,000 vectors by a factor of 20 by generating 20 new vectors by random perturbation of each vector in the 60000 vector set within a small radius around the vector. We use this 1,200,000 vector set for the GOC optimization and for generating the BVI lists subsequent to optimization by FBF and GOC.

Here we report results for the following optimization and search procedures:

1. BCK: Backtracking search [2] with tree optimized using FBF criterion.
2. BVI-FBF: Bucket-Voronoi intersection search with tree optimized using FBF criterion.
3. BVI-GOC: Bucket-Voronoi intersection search with tree optimized using GOC criterion.

Table I shows the results for BCK, BVI-FBF and BVI-GOC in the split-vector quantizer of codebook size $N = 4096$ for the two parts of dimensions $K = 4$ and $K = 6$. Here, the full-search complexity is 4096 distances per test vector. The performance of the fast search is measured in terms of the average ($\overline{nc}$) and worst-case complexity ($\widehat{nc}$) of the search. Here we have shown results for quantizing two sets of data for tree depths $d = 8$ and $d = 12$: i) 1,200,000 vectors used in optimizing the tree using GOC criterion; this is referred to as the training data (Trg) and, ii) test data of 8000 vectors (Tst).

From this table, it can be seen that both backtracking search BCK and the BVI search (with either FBF or GOC optimization) have comparable average complexity $\overline{nc}$. However, the backtracking search algorithm BCK suffers a very high worst-case complexity in comparison to BVI search. Considering the BVI search, it can be noted that the GOC optimization offers lower worst-case complexity in comparison to the FBF criterion. The BVI search is able to reduce the search complexity of both parts ($K = 4$ and $K = 6$) by 2 orders of magnitude for depth $d = 12$ over the full-search algorithm. The consistency of performance of BVI search for both the training and test data can also be noted.

[1] These were obtained from the 'FM radio' data base described in detail in [1]. This consists of 23 minutes of speech recorded from 35 different FM radio stations. The first 1200 seconds of speech (from about 170 speakers) forms the training set and the last 160 seconds of speech (from 25 speakers) forms the test set. The 10-dimensional LSF vectors were obtained by 10-th order LPC analysis performed for every 20 ms using a 20-ms analysis window.

[2] *Backtracking search:* The FBF criterion [3], [4] was proposed for optimizing the tree to minimize the expected search time under a backtracking search procedure. The backtracking search consists in first finding a tentative (current) nearest-neighbor of the given test vector $\mathbf{x}$ from among the set of codevectors within the bucket containing the test vector $\mathbf{x}$ and then in determining the actual nearest-neighbor from among other buckets which overlap with the current nearest-neighbor ball. The overall search is carried out by a recursive procedure which implicitly performs a backtracking to move from one overlapping bucket to another, the overlap being detected by a bounds-overlap-ball test. The algorithm based on this optimization and backtracking search has a $O(\log N)$ average complexity performance. However, the main shortcoming of the backtracking search is its high computational overhead and the resulting high worst-case complexity [4].

6. CONCLUSIONS

The quantization of the LPC parameters at very low bitrates to achieve transparent quality quantization is an important problem. Recent solutions to this are based on vector quantization (VQ) of the LPC vectors using large codebook sizes. However, the resultant high computational complexity of VQ encoding is a main problem in these quantizers. In this paper, we have used a fast vector quantization encoding procedure termed the bucket-Voronoi intersection (BVI) search to reduce the computational complexity of the split-vector quantizer. We have shown that the BVI search algorithm can offer over 2 orders of magnitude reduction in the computational complexity of the split vector quantizer, thereby rendering it amenable for practical real-time coding.

Table 1: Performance comparison of the K-d tree search algorithms BCK, BVI-FBF and BVI-GOC

d	Data set	Search Algorithm	Part-I $K = 4$		Part-II $K = 6$	
			$\overline{nc}$	$\widehat{nc}$	$\overline{nc}$	$\widehat{nc}$
8	Trg	BCK	64.0	325	181.6	1460
		BVI-FBF	63.6	103	133.1	231
		BVI-GOC	60.8	71	125.1	154
	Tst	BCK	68.7	286	185.0	949
		BVI-FBF	63.9	103	131.7	231
		BVI-GOC	61.1	71	124.9	154
12	Trg	BCK	28.5	162	91.5	1211
		BVI-FBF	14.3	33	31.7	83
		BVI-GOC	12.8	22	26.9	51
	Tst	BCK	31.4	130	93.7	637
		BVI-FBF	14.4	33	31.8	83
		BVI-GOC	12.9	22	26.7	51

REFERENCES

[1] K. K. Paliwal and B. S. Atal, "Efficient vector quantization of LPC parameters at 24 bits/frame", *IEEE Trans. Speech and Audio Processing*, vol. 1, no.1, pp. 3-14, Jan. 1993.

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

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

[4] V. Ramasubramanian and K. K. Paliwal, "Fast vector quantization encoding based on K-d tree backtracking search algorithm", *Digital Signal Processing*, 1997 (in print).