A Genetic Approach to the Design of General-Tree-Structured Vector Quantizers for Speech Coding

Lin Yu Tseng* and Shiueng Bien Yang
Department of Applied Mathematics
National Chung Hsing University
Taichung, Taiwan 402, R.O.C.
e-mail: lytseng@amath.nchu.edu.tw

Abstract

The full-search vector quantization suffers from spending much time searching the whole codebook sequentially. Recently, several tree-structured vector quantizers had been proposed. But almost all trees used are binary trees and hence the training samples contained in each node are forced to be divided into two clusters artificially. We present a general-tree-structured vector quantizer that is based on a genetic clustering algorithm. This genetic clustering algorithm can divide the training samples contained in each node into more natural clusters. A distortion threshold is used to guarantee the quality of coding. Also, the Huffman coding is used to achieve the optimal bit rate after the general-tree-structured coder was constructed. An experiment on speech coding was conducted. A comparison of the performance of this vector quantizer and the other two tree-structured vector quantizers is also given.

I. Introduction

There are two important points in the design of the codebook. The first is trying to find a good set of codewords such that the quantization errors may be minimized. The generalized Lloyd algorithm (GLA) [1] had been widely used in the design of the codebook. It suffers the drawback that the user must provide the number of clusters in advance while the user in general has no idea about how many clusters there should be in the data set. In this paper, a genetic clustering algorithm [2] is used in the codebook design. This algorithm will search for a proper number of cluster centers and do the clustering simultaneously. The second important point is trying to make the codebook search as fast as possible. The full search VQ searches the whole codebook sequentially. It takes O(n) time when the codebook contains n codewords. Recently, tree-structured vector quantizers (TSVQ) [3],[4],[5],[6],[7],[8] were proposed. In TSVQ, the codebook search takes about O(logn) time when the tree-structured codebook is roughly a balanced tree.

* Author to whom correspondence should be sent.

Chou et al. [5] proposed a method to design an unbalanced tree coder. At first, a balanced fixed rate TSVQ is grown to a predetermined height. Then it is optimally pruned back by using the generalized Breiman, Friedman, Olshen, and Stone (BFOS) algorithm [9]. Riskin et al. [6] and Balakrishnan et al. [7] independently proposed similar greedy methods for node splitting and tree growing. The tree is grown by splitting one node at a time. The design time is less for this method compared to the pruning method. All tree-structured coders mentioned above are binary trees. Usually there are cases that it is not proper to divide the set of training samples into exactly two clusters. The genetic clustering algorithm searches for a proper number of clusters all by itself. By applying this genetic clustering algorithm, we proposed a general-tree-structured vector quantizer. A distortion threshold is used to guarantee the quality of coding. The training samples contained in a node are divided into two or more clusters by applying the genetic clustering algorithm if the average distortion of this node is greater than the distortion threshold. Therefore, the tree will grow as a general tree with the average distortion rate of each leaf less than the distortion threshold. All leaves are then used to build a Huffman code tree [10] and hence the coding is optimal given the general tree coder.

The remaining parts of the paper are organized as follows. The genetic clustering algorithm is briefly described in section II. The general-tree-structured vector quantizer is described in section III. The experiment is given in section IV and the conclusions are given in section V.

II. A Brief Description of the Genetic Clustering Algorithm

The genetic algorithms are good at searching [11], [12]. Therefore, it is proper to apply the genetic approach in the searching of clusters in a data set that we have no a priori knowledge. The genetic clustering algorithm consists of an initialization step and the iterative generations. In each generation, there are three phases, namely, the reproduction phase, the crossover phase and the mutation phase. The initialization step and the three

phases of each generation are described in the following.

Initialization step:

A population of N strings is generated. All the strings are of length n. Each string represents a subset of $\{ O_1, O_2, \ldots, O_n \}$. If O_i is in this subset, the ith position of the string will be 1; Otherwise, the ith bit will be 0. Each string represents the seeds for a clustering because we will use each object in the subset represented by this string as a seed to generate a cluster.

Reproduction phase:

The fitness of a string R is the sum of two scores, SCORE1 and SCORE2. Let $\{ S_1, S_2, \ldots, S_m \}$ be the set of clusters generated by string R. Let $C_1, C_2, \ldots, C_m$ be the centers of $S_1, S_2, \ldots, S_m$ respectively. Let S'_j be defined as follows.

$$S'_j = \{ O_i \mid O_i \in S_j \text{ and } d(O_i, C_j) \le d(O_i, C_k) \text{ for all } k \text{ such that } 1 \le k \le m \text{ and } k \ne j \},$$

where $d(O_i, C_j)$ is the Euclidean distance.

(Note that S'_j is a subset of S_j and contains those objects of S_j that are indeed closer to the center of S_j than to other centers. Also note that S'_j may be a proper subset of S_j. In other words, there may be some objects of S_j that are closer to other centers than to the center of S_j.)

Now, SCORE1 and SCORE2 are defined as follows.

If $\sum_{j=1}^{m} |S'_j| < n$ then $\text{SCORE1} = \sum_{j=1}^{m} |S'_j|$ and SCORE2=0.

If $\sum_{j=1}^{m} |S'_j| = n$ then SCORE1= n and SCORE2 is defined in the following.

Suppose $O_i \in S_j$, we define $\text{Dintra}(O_i) = d(O_i, C_j)$ and

$$\text{Dinter}(O_i) = \min_{\substack{1 \le k \le m \\ k \ne j}} d(O_i, C_k) .$$

$$\text{SCORE2} = \sum_{i=1}^{n} (\text{Dinter}(O_i) * w - \text{Dintra}(O_i)),$$

where w is a weight. (If the value of w is small, we emphasize the importance of $\text{Dintra}(O_i)$. This tends to produce more clusters and each cluster tends to be compact. If the value of w is chosen to be large, we emphasize the importance of $\text{Dinter}(O_i)$. This tends to produce less clusters and each cluster tends to be loose.) After the calculation of fitness for each string in the population, the reproduction operator is implemented by using a roulette wheel with slots sized according to fitness.

Crossover phase:

In the crossover phase, for each chosen pair of strings, two random numbers in {1,2,...,n} are generated to decide which pieces of the strings are to be interchanged. For each chosen pair of strings, the crossover operator is done with probability p_c.

Mutation phase:

In the mutation phase, bits of the strings in the population will be chosen with probability p_m. Each chosen bit will be changed from 0 to 1 or from 1 to 0.

III. The General-Tree-Structured Vector Quantizers

In this section, the design method of the general-tree coder is described. The design of the coder is based on the genetic clustering algorithm introduced in section II. The corresponding decoder is a Huffman decoding tree. An assumption has been made that the distribution of the training samples used to design the codebook resembles the distribution of the data to be coded.

Before describing how the tree coder is constructed, we first introduce the data structures used. There are two trees. A general tree T that acts as the coder and a binary tree H that acts as the decoder. The data structure for a node of T is depicted in Figure 1(a) and the data structure for a node of H is depicted in Figure 1(b). Note that the "codeword" field of a leaf node of T contains the codeword while the "codeword" field of an internal node of T contains the center of the cluster.

The following algorithm describes the design method of the general-tree coder.

Algorithm Tree_Coder_Construction

Input: A set of training samples and a distortion threshold ε.

Output: A general-tree coder T with the average distortion of each leaf node less than or equal to ε and the Huffman decoding tree H.

Step 1. The queues Q and Q′ are set to be empty. Build the root node. Set up the pointer pointing the set of whole training samples, fill in the no. of training samples, and compute the average distortion.

If The average distortion is greater than ε.

Then Add this node at the rear of the queue Q.

Else Add this node at the rear of the queue Q′.

Step 2. **If** The queue Q is empty.

Then The coder tree T has been constructed, go to Step 4.

Pick one node from the front of the queue Q. Let this node be N. Apply the genetic clustering algorithm to the set of training samples contained in this node. Let there be p clusters $S_1, S_2, \ldots, S_p$.

Step 3. **For** i = 1 **to** p
Begin
Build a child node N_i of node N. Set up the pointer pointing to S_i. fill in the size of S_i, compute the center of S_i, compute the average distortion.
If The average distortion of N_i is greater than ε.
Then Add N_i at the rear of the queue Q.
Else Add this node at the rear of the queue Q'.
End
Go to Step 2.

Step 4. Take all the nodes in the queue Q'. For each node, build a new node with the data structure depicted in Figure 1(b). Use these nodes to construct a Huffman decoding tree H.

Step 5. Traverse each leaf node of H to find the code for this node and put this code into the "code" field of the corresponding node in the coder tree T. Stop.

An example is given to illustrate the algorithm. In Figure 2(a), the coder tree T is depicted. Figure 2(b) shows the Huffman decoding tree. Each leaf node of the Huffman decoding tree is traversed and the code of each leaf node of the coder tree is thus derived.

IV. The Experiment

In this experiment, 5000 spectral feature vectors of speech were used as the training samples to construct the tree coder and the corresponding Huffman tree decoder. These 5000 spectral feature vectors were taken from 300 single syllable speeches. The sampling rate of the speech is 11025 Hz. Each frame contains 400 sample points and is transformed into a spectral feature vector that contains 30 components. After obtaining the 5000 training samples, these 300 single syllable speeches were spoken again by the same person and another 5000 spectral feature vectors were taken as the testing samples. Two other methods, the Riskin's method [6] and the Chou's method [5], were implemented to make a comparison with our method. These two methods are denoted as TSVQ[Greedy] and TSVQ[Prune] in Table 1 and Figure 3. The SNR is computed by the formula $\mathrm{SNR} = 10\log\frac{\sum_n x_n^2}{\frac{1}{N}\sum_n (x_n - x_n')^2}$. It is noted that for different distortion levels, both the bit rate and the SNR of our method outperforms the other two methods. This is revealed by Table 1 and Figure 3.

V. Conclusions

The tree-structured vector quantizers (TSVQ) have the advantage of efficiency in searching codebook in comparison with the traditional full-search vector quantizers. But in TSVQ, it is not proper to always divide a set of training samples into two clusters in order to keep the tree binary. This in general will results in the increase of either the bit rate or the average distortion or both. A general-tree-structured VQ is proposed in this paper. A genetic clustering algorithm is used to grow the coder tree. By using this clustering algorithm, a set of training samples will be divided into several natural clusters in accordance to the characteristics of this set. A distortion threshold is used to guarantee the quality of the codebook. After the general-tree coder was constructed, the Huffman coding is used to achieve the optimal bit rate.

References

[1] Y. Linde, A. Buzo and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, no. 1, pp.84-95, 1980.

[2] L. Y. Tseng and S. B. Yang, "Genetic algorithms for clustering, feature selection and classification," *Proc. of IEEE International Conference on Neural Networks*, pp. 1612-1616, 1997.

[3] A. Buzo, A. H. Gray, Jr., R. M. Gray and J. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acout., Speech, Signal Process.*, vol. 28, no.5, pp. 562-574, 1985.

[4] J. Makhoul, S. Roucos and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, no. 11, pp. 1551-1588, 1985.

[5] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Information Theory*, vol. 35, no. 2, pp. 299-315, 1989.

[6] E. A. Riskin and R. M. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Trans. Signal Process.*, vol. 39, no. 11, pp. 2500-2507, 1991.

[7] M. Balakrishnan, W. A. Pearlman and L. Lu, "Variable-rate tree-structured vector quantizers," *IEEE Trans. Information Theory*, vol. 41, no. 4, pp. 917-930, 1995.

[8] T. D. Chiueh, T. T. Tang, and L. G. Chen, "Vector quantization using tree-structured self-organizing feature maps," *IEEE Journal Selected Areas Commun.*, vol. 12, no. 9, 1994.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C.

J. Stone, *Classification and Regression Trees. The Wadsworth Statistics/Probability Series.* Belmont, CA: Wadsworth, 1984.

[10] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms,* The MIT Press, Cambridge, Massachusetts, 1990.

[11] M. Srinivas and M. Patnaik, "Genetic algorithm-A survey," *IEEE Computer*, vol. 27, no. 6, pp. 17-26, 1994.

[12] D. E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

Table 1. The comparison of our method and the other two methods (testing samples).

Experiments	# of codewords	Distortion threshold (ε)	Our method		TSVQ [Greedy]		TSVQ [Prune]	
			Average bits per codeword	SNR	Average bits per codeword	SNR	Average bits per codeword	SNR
(1)	22	250000	4.633	13.985	4.711	13.511	4.721	13.489
(2)	85	200000	6.321	15.521	6.682	15.382	6.701	15.298
(3)	124	150000	6.727	16.128	6.879	15.977	6.891	15.879
(4)	157	100000	7.155	18.471	7.385	18.310	7.401	18.285
(5)	257	50000	7.911	20.178	8.184	20.011	8.211	20.023

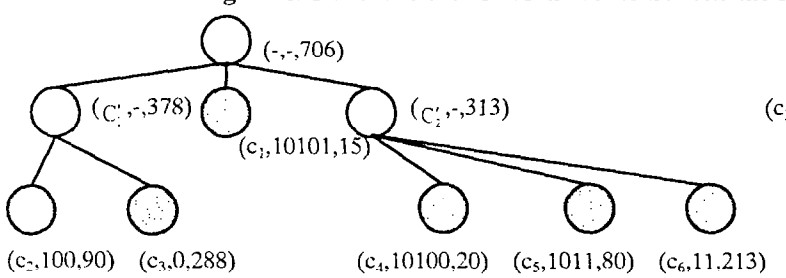
codeword	code	pointer to the set of training samples	no. of training samples	average distortion	first child pointer	right sibling pointer

(a) Fields of a node of the coder tree.

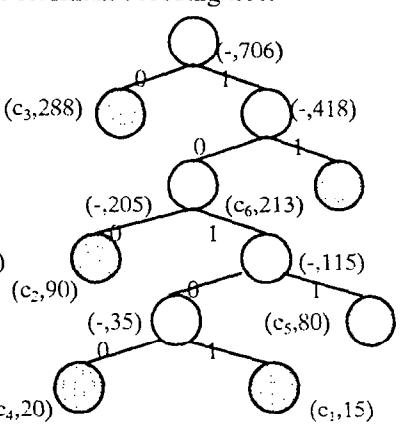
codeword	no. of training samples	pointer to the corresponding node in the coder tree	left child pointer	right child pointer

(b) Fields of a node of the Huffman decoding tree.

Figure 1. The data structure of the coder tree and the Huffman decoding tree.



(a) The coder tree T.



(b) The Huffman decoding tree H.

Figure 2. An example of the coder tree T and the Huffman decoding tree H.

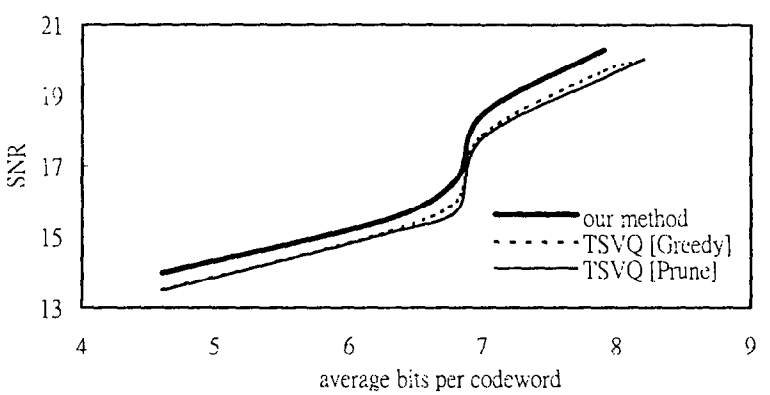


Figure 3. The ratio of the SNR and the bit rate of the data shown in Table 1.