Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Novel Algorithm for Vector Quantization Codebook Design

Ting-Chi Wang and Hung-Ru Huang
Department of Information and Computer Engineering
Chung Yuan Christian University
Chungli, Taiwan, R.O.C.

Abstract

This paper studies the codebook design problem arising in vector quantization, and presents a novel algorithm of polynomial time for this problem. The proposed algorithm first performs a step, called clustering, which partitions the given set of training vectors into a collection of disjoint clusters subject to a user-specified distortion constraint. Then, the proposed algorithm proceeds to reorganize the clusters and arrange them as a linearly ordered set. Finally, the dynamic programming technique is applied to further partition the linearly ordered set into a user-specified amount of groups, each of whose centroids corresponds to a codevector. The proposed algorithm has been implemented in C language, and the preliminary experimental results indicate that the proposed algorithm is capable of designing a better codebook in shorter run time than the well-known LBG algorithm.

1 Introduction

Vector quantization has been a very important technique for compressing both the image and the speech data [7]. One of the key problems arising in vector quantization is *the codebook design problem*. A popularly adopted methodology for solving the codebook design problem is to partition the given set of training vectors into a user-specified amount of disjoint groups, and to use the centroid of each group to form a codevector. Depending on how the partitioning is generated, different algorithms were proposed previously. For example, the well-known LBG algorithm [9] uses the iterative improvement technique to repeatedly generate a better partitioning based on the previous one until convergence is met. Although the LBG algorithm is easy to implement, it suffers from both its expensive run time and its solution being just locally optimum. To avoid generating a locally optimum codebook, the simulated annealing based algorithms were proposed [5,7], but their run times were much worse than the LBG algorithm. On the other hand, there were algorithms proposed to shorten the run time [4, 7], but their codebooks were not as good as the LBG algorithm. The details of other previous work on

codebook design can be found in [7].

In this paper, a novel algorithm of polynomial time is presented for the codebook design problem. To reduce the problem size (i.e., the amount of training vectors), the proposed algorithm first performs the clustering step, which partitions the data into a collection of disjoint clusters subject to the constraint that the average distortion per vector in each cluster cannot exceed a user-specified value. To make the clustering step efficient and effective, it is designed based on the idea of building a k-d tree [6] but with several modifications. Once the clustering step is done, the clusters are then reorganized and arranged as a linearly ordered set, and finally, the dynamic programming technique is applied to partition the linearly ordered set into a user-specified amount of groups, each of whose centroids corresponds to a codevector. The proposed algorithm has been implemented in C language, and the preliminary experimental results demonstrate that the proposed algorithm is capable of designing a better codebook in shorter run time than the LBG algorithm.

The rest of this paper is organized as follows. Section 2 gives a formal description of the codebook design problem considered in this paper. Section 3 gives a brief review of k-d trees. The proposed algorithm is described in Section 4. Section 5 reports the experimental results, and concludes this paper.

2 Problem Formulation

Let Q denote a vector quantizer which maps a set X of M k-dimensional training vectors to a codebook Y of N k-dimensional codevectors, where $X = \{x_i | i = 1,2..., M; x_i \in R^k\}$, and $Y = \{y_i | i = 1,2,..., N; y_i \in R^k\}$. Let $d(x,y)$ denote the distortion between two vectors x and y, and be defined as the square of the Euclidean distance between them. Under the assumption that the full-search encoding scheme is used, we have for each training vector x_i, $Q(x_i) = \arg \min_{y_j \in Y} d(x_i, y_j)$.

The codebook design problem considered in this paper is formulated as follows. Given a set X of M k-dimensional training vectors, the objective of this problem is to find a codebook Y of N k-dimensional codevectors in such a way that the total distortion caused by using Y to encode X, i.e., $\sum_{i=1}^{M} d(x_i, Q(x_i))$, is minimized. The meanings of M, N, and k mentioned in this section will be used in the rest of this paper.

3 K-d Trees

The idea of building a k-d (k-dimensional) tree is briefly reviewed in this section since the clustering step of the proposed algorithm is based on it. The k-d trees are first proposed by Bentley [3], and can be built to store k-dimensional real data. A k-d tree is a binary tree, and consists of internal nodes and external nodes. When an internal node is built, it is assigned a set of data to be further partitioned into two sets. The partitioning is done based on two components stored in the internal node; one component is a *discriminating axis* out of the k axes, and the other is a *partition key* on that discriminating axis. However, since each internal node is used only for partitioning and

searching the data, there is no need to store the data assigned to that node (except the discriminating axis and the partition key) after the partitioning is done. On the other hand, a node is said to be external if it does not have any children. The external nodes are the only nodes which store data. Therefore, the collection of external nodes in a k-d tree denotes a partitioning of the given set of data.

The process of building a k-d tree is briefly described as follows. Initially, the root is created and both its discriminating axis, say a ($1 \le a \le k$), and its partition key, say b, are determined. The two children of the root are then created, and based on a and b, the data is partitioned into two sets, which are assigned to the left and the right children of the root, respectively. Any data whose component on axis a has the value less than or equal to b is assigned to the left child, and any data whose component on axis a has the value greater than b is assigned to the right child. If the amount of data assigned to a newly created node does not exceed the user-specified value, then that node becomes an external node, otherwise that node becomes an internal node. For any newly created internal node, both its discriminating axis and its partition key need to be determined, and the set of data assigned to it needs to be partitioned into another two subsets. The whole process is finished when no node in the tree needs to be partitioned further.

Depending on how to determine the discriminating axis and the partition key, different algorithms, such as [3,6], for building a k-d tree were proposed. To achieve the expected logarithmic time for searching any data, the authors in [6] set the partition key of an internal node to the *median* of the data assigned to that node with respect to the discriminating axis. On the other hand, to get a better partitioning, the authors in [6] set the discriminating axis of an internal node to the one with the largest variance with respect to the data assigned to that node among all the axes.

4 The Proposed Algorithm

Before describing the proposed algorithm in details, the following definitions are given.

Definition 1: A set C is said to be a *cluster* if C is a subset of the set of training vectors.

Definition 2: The *average distortion* of a cluster C is defined as $(1/|C|)\Sigma_{x \in C} d(x, centroid(C))$, where $|C|$ denotes the number of training vectors in C, and *centroid* (C) denotes the centroid of C (i.e., the centroid of the training vectors in C).

Definition 3: A set $S = \{C_1, C_2, ..., C_m\}$ of m clusters is said to be a *linearly ordered set* if the ordering of the clusters in S is fixed (i.e., C_i is the ith element in S, where $1 \le i \le m$).

Definition 4: Let $S = \{C_1, C_2, ..., C_m\}$ be a linearly ordered set. A partitioning P is said to be a *q-way linear partitioning* on S if

$$p = \{\{C_1, C_2, ... C_{i_1}\}, \{C_{i_1+1}, C_{i_1+2}, ..., C_{i_2}\}, ..., \{C_{i_{q-1}+1}, C_{i_{q-1}+2}, ..., C_m\}\}$$

for some $i_1, i_2, ..., i_{q-1}$, where $1 \le i_1 < i_2 < ... < i_{q-1} < m$. Here, each element in P must be a set of consecutive elements in S.

Now, it is ready to describe the proposed algorithm. The algorithm consists of the following four steps whose details are given in the subsections.

Step1: Perform clustering.
Step 2: Reorganize the training vectors into another set of clusters.
Step 3: Arrange the set of new clusters as a linearly order set.
Step 4: Determine an N-way linear partitioning P on the linearly ordered set, and output the centroid of the training vectors in each element of P as a codevector.

4.1 Step 1

Let W denote the user-specified upper bound on the average distortion of each cluster to be generated. Step 1 partitions the set of training vectors into a set of disjoint clusters, each of whose average distortions does not exceed W. This step is done by an approach similar to the idea of building a k-d tree as presented in [6] but with the following three major modifications.

1. Unlike the approach in [6] which chooses the median as the partition key, Step 1 chooses the *mean*. There are two main reasons for such a choice. Firstly, the coefficient of the linear term of the time complexity of finding the mean is much smaller than that of finding the median although both problems can be asymptotically solved in linear time [1]. Secondly, the algorithm for finding the mean is much easier to implement than the linear-time algorithm for finding the median.
2. Unlike the approach in [6] which stops further partitioning an internal node when the number of training vectors assigned to that node is no more than a user-specified value, Step 1 stops partitioning a node when the average distortion of the training vectors assigned to that node is no more than W.
3. Unlike the approach in [6] which keeps all the nodes in the k-d tree, Step 1 deletes an internal node right after that node is partitioned, and hence uses less memory space.

As mentioned in Section 3, the set of external nodes in a k-d tree forms a partitioning of the set of training vectors, and hence each external node generated by Step 1 will form a cluster. The time complexity of Step 1 is analyzed as follows. Let L denote the number of clusters generated by Step 1. Since there are a total of $2L$-1 internal and external nodes generated and each node can be checked in $O(kM)$ time to see whether it needs to be further partitioned, the time complexity of Step 1 is $O(kLM)$, which is $O(kM^2)$ as $L=O(M)$.

4.2 Step 2

Since partitioning the data into two sets is only based on the values on one axis, the amount of clusters generated could be a very large number (due to the user-specified distortion constraint) and could substantially increase the run times of the subsequent steps. To remedy this problem, Step 2 reorganizes the set of training vectors into another set of clusters. This is done as follows. Firstly, the centroid of each cluster generated by Step 1 is used as a codevector, and then the full-search scheme is applied to encode each training vector. Finally, for each codevector, the training vectors encoded by it form a new cluster.

Again, let L denote the amount of clusters generated at Step 1. Since encoding each training vector can be done in $O(kL)$ time and there are M training vectors to be encoded, the time complexity of Step 2 is $O(kLM)$, which is $O(kM^2)$ as $L=O(M)$. Throughout the rest of this paper, L will be used to denote the number of clusters generated after Step 2.

4.3 Step 3

After Step 2 is done, the problem remained to solve is to find a partitioning of the clusters generated at Step 2 into a set of groups, each of whose centroids forms a codevector. Since the remaining problem is still very hard to solve, a methodology used to solve the multi-way circuit partitioning problem arising in VLSI design [2] is adopted here to help simplify the remaining problem. This methodology consists of two steps corresponding to Step 3 and Step 4 of the proposed algorithm, respectively. The objective of Step 3 is to arrange the clusters generated at Step 2 as a "good" linearly ordered set, and the objective of Step 4 is to find an N-way linear partitioning on the linearly ordered set with "good" quality.

In the current implementation of the proposed algorithm, Step 3 uses a greedy method to generate the linearly ordered set by adding one cluster at a time. This greedy method works as follows. Initially, the set is empty, and the first cluster to be added to the set is the one whose centroid has the shortest distance from the origin vector among all the clusters. Now, consider adding the ith cluster to the set, where $i>1$. Suppose the current set is $\{C_1, C_2, ..., C_{i-1}\}$, and let C be the vector represented by $\Sigma_{j=1}^{i-1}(centroid(C_j)/2^{(i-1-j)})$. Then, the ith cluster to be added to the set is the one whose centorid has the shortest distance from C among all the remaining clusters. This selection is mainly based on the following two heuristics. Firstly, any two consecutive clusters in the set are expected to have the distance between them as short as possible. Secondly, for each cluster in the set, the clusters with short distances from it are expected to be added to the set as soon as possible. It is clear that since adding one cluster to the set can be done in $O(kL)$ time and there are L clusters, the time complexity of Step 3 is $O(kL^2)$, which is $O(kM^2)$ as $L=O(M)$.

4.4 Step 4

Before describing the details of Step 4, the following notations are given.

- $S=\{C_1, C_2, ..., C_L\}$: the linearly ordered set generated at Step 3.
- $C_{i,j}=\{C_i, C_{i+1}, ..., C_j\}$: the set of $j-i+1$ consecutive clusters in S starting from C_i.
- $w(C_{i,j}) = \Sigma_{C \in C_{i,j}} \Sigma_{x \in C} d(x, centroid(\cup_{h=i}^{j} C_h))$: the total distortion caused by using the centroid of the training vectors in the union of the clusters in $C_{i,j}$ to encode all the training vectors in the union of the clusters in $C_{i,j}$.

- $P_{1,j,q} = \{C_{1,i_1}, C_{i_1+1,i_2}, \ldots, C_{i_{q-1}+1,j}\}$: a q-way linear partitioning on $C_{1,j}$.
- $\cos t(P_{1,j,q}) = \Sigma_{C \in P_{1,j,q}} w(C)$: the total distortion caused by $P_{1,j,q}$.
- $\hat{P}_{1,j,q}$: a q-way linear partitioning on $C_{1,j}$ with the minimum total distortion among all possible q-way linear partitionings on $C_{1,j}$.

The objective of Step 4 is to find an N-way linear partitioning on S ($=C_{1,L}$) with the minimum total distortion, i.e., to find a $\hat{P}_{1,L,N}$. According to the notations defined above, the following equation can be easily derived:

$$\cos t(\hat{P}_{1,j,q}) = \min_{1 \le j' < j} \{\cos t(\hat{P}_{1,j',q-1}) + w(C_{j'+1,j})\} \qquad (1)$$

Based on Equation (1), $\cos t(\hat{P}_{1,L,N})$ can be obtained using the dynamic programming technique [1]. Once $\cos t(\hat{P}_{1,L,N})$ is known, $\hat{P}_{1,L,N}$ can be easily obtained based on the information which is stored during the process of computing all $\cos t(\hat{P}_{1,j,q})$'s [1].

To compute Equation (1) more efficiently, the term $w(C_{j'+1,j})$ in Equation (1) needs to be computed only once and stored in a table; thereafter, its value will be obtained in $O(1)$ time by table lookup. Moreover, each $w(C_{j',j})$ can be also computed efficiently, where $1 \le j' < j \le L$. That is, for each j', $w(C_{j',j'+1}), w(C_{j',j'+2}), \ldots, w(C_{j',L})$ are computed in that order. Since each $C_{j',j'+r}$ can be obtained by the union of $C_{j',j'+r-1}$ and $C_{j'+r,j'+r}$, $w(C_{j',j'+r})$ can be efficiently computed using the following equation as reported in [4], where $1 \le r \le L - j'$.

$$w(C_{j',j'+r}) = w(C_{j',j'+r-1}) + w(C_{j'+r,j'+r}) + \frac{n_{j',j'+r-1} \times n_{j'+r,j'+r}}{(n_{j',j'+r-1} + n_{j'+r,j'+r})} \times d(\bar{x}_{j',j'+r-1}, \bar{x}_{j'+r,j'+r}) \qquad (2)$$

In Equation (2), $n_{j',j'+r-1}$ and $n_{j'+r,j'+r}$ denote the numbers of the training vectors in $C_{j',j'+r-1}$ and $C_{j'+r,j'+r}$, respectively; $\bar{x}_{j',j'+r-1}$ and $\bar{x}_{j'+r,j'+r}$ denote the centroids of the training vectors in $C_{j',j'+r-1}$ and $C_{j'+r,j'+r}$, respectively. Based on Equation (2), computing $w(C_{j',j'+r})$ takes $O(k)$ time as $w(C_{j',j'+r-1})$, $w(C_{j'+r,j'+r})$, $n_{j',j'+r-1}$, and $n_{j'+r,j'+r}$ each can be obtained in $O(1)$ time by table lookup. Therefore, computing all $w(C_{j',j'+r})$'s takes $O(kL^2)$ time as there are $O(L^2)$ such $w(C_{j',j'+r})$'s.

According to Equation (1), it is clear that each $\cos t(\hat{P}_{1,j,q})$ can be computed in $O(L)$ time, since there are $O(L)$ possible values of j', and $\cos t(\hat{P}_{1,j',q-1})$ and $w(C_{j'+1,j})$

each can be obtained in $O(1)$ time by table lookup. Since $O(NL)$ such $\cos t(\hat{P}_{1,j,q})$'s need to be computed in order to get $\cos t(\hat{P}_{1,L,N})$, obtaining $\cos t(\hat{P}_{1,L,N})$ takes $O(NL^2)$ time. Finally, based on $\cos t(\hat{P}_{1,L,N})$, $\hat{P}_{1,L,N}$ can be generated in $O(L)$ time based on the stored information.

According to the time complexities analyzed in the above paragraphs, the total complexity of Step 4 is $O(kL^2)+O(NL^2)+O(L)$, which is $O((k+N)M^2)$ since $L=O(M)$.

4.5 Overall Time Complexity

By summing up the time complexity of each step, the total time complexity of the proposed algorithm is as follows:

$O(kM^2)+O(kM^2)+O(kM^2)+O((k+N)M^2)= O((k+N)M^2)$.

5 Experimental Results and Conclusions

The proposed algorithm has been implemented in C language on an SGI Challenge L workstation . Two 512×512 monochrome images (i.e., the peppers and the F-16 airplane images) were used as the training data. Each vector was a 4×4 block of pixels (i.e., 16 dimensions), and each codebook was designed to have 256 codevectors. In the current implementation of the proposed algorithm, the upper bound W on the average distortion of each cluster was obtained using the following simple approach. Firstly, 256 vectors were randomly chosen as codevectors from the training data. Then, the training data were encoded using the full-search scheme. Finally, the average distortion per vector based on the encoding result was computed and then multiplied by a user-specified parameter R to get W. The time spent on obtaining W was also added to the run time of the proposed algorithm.

The proposed algorithm was compared with the well-known LBG algorithm. The results are given under "Training data" of Table 1 where four different values of R were used in the proposed algorithm. Clearly, the proposed algorithm was able to design a better codebook in shorter run time than the LBG algorithm for each value of R. In particular, the PSNR could be improved by more than 0.8dB, and the run time could be reduced by at least about 50%. To further demonstrate the effectiveness of the proposed algorithm, the codebooks designed by the proposed algorithm and by the LBG algorithm were used to encode the Lena image, which is a 512×512 monochrome image and is outside the training data. The results are given under "Lena" of Table 1. Again, the proposed algorithm was able to generate a higher PSNR than the LBG algorithm for each value of R. In particular, the PSNR could be increased by more than 0.7dB. Due to space limitation, some other experimental results which have been obtained cannot be given in this paper, but their details can be found in [8].

Algorithm		Training data			Lena
		# of Clusters	PSNR	Run Time(sec)	PSNR
LBG Algorithm			30.030	1027	30.007
Proposed Algorithm	R=0.077	1190	30.597	367	30.671
	R=0.073	1284	30.695	408	30.805
	R=0.069	1358	30.697	443	30.731
	R=0.065	1505	30.835	519	30.728

Table 1: Experimental results.

Recently, the idea of the proposed algorithm has been also extended to solve the codebook design problem for entropy-constrained vector quantization, and the experimental results are very promising [10]. Currently, several possible directions to further improve the idea of the proposed algorithm is under study.

Acknowledgments

This work was partially supported by the National Science Council of R.O.C. under grant NSC-86-2215-E-033-001.

References

[1] Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] J. Alpert and S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, The Better, " *Proc. Design Automation Conf.*, 1995, pp. 195-200.

[3] L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Comm. ACM,* September 1975, pp. 509-517.

[4] H. Equitz, "A New Vector Quantization Clustering Algorithm," *IEEE Trans. Acoust. Speech Signal Porcess.*, October 1989, pp. 1568-1575.

[5] K. Flanagan, D. R. Morrell, R. L. Frost, C. J. Read and B. E. Nelson, "Vector Quantization Codebook Generation Using Simulated Annealing," *Proc. ICASSP*, 1989, pp. 1759-1762.

[6] H. Friedman, J. L. Bentley and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Software*, September 1977, pp. 209-226.

[7] A. Gresho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.

[8] H.-R. Huang, *Efficient Algorithms for Vector Quantization Codebook Design Based on Clustering*, Master Thesis, Department of Information and Computer Engineering, Chung Yuan Christian University, 1996.

[9] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantizer Design, " *IEEE Trans. Comm.*, January 1980, pp. 84-95.

[10] T.-C. Wang and H.-R. Huang, "An Effective Codebook Design Algorithm for Entropy-Constrained Vector Quantization," to appear in *National Symposium on Telecommunication*, 1996.