Variable-Branch Tree-Structured Vector Quantization

Shiueng-Bien Yang

***Abstract*—Tree-structured vector quantizers (TSVQ) and their variants have recently been proposed. All trees used are fixed M-ary tree structured, such that the training samples in each node must be artificially divided into a fixed number of clusters. This paper proposes a variable-branch tree-structured vector quantizer (VBTSVQ) based on a genetic algorithm, which searches for the number of child nodes of each splitting node for optimal coding in VBTSVQ. Moreover, one disadvantage of TSVQ is that the searched codeword usually differs from the full searched codeword. Briefly, the searched codeword in TSVQ sometimes is not the closest codeword to the input vector. This paper proposes the multiclassification encoding method to select many classified components to represent each cluster, and the codeword encoded in the VBTSVQ is usually the same as that of the full search. VBTSVQ outperforms other TSVQs in the experiments presented here.**

***Index Terms*—Genetic clustering algorithm, multiclassification encoding method, tree-structured vector quantizer (TSVQ).**

I. Introduction

Tree-structured vector quantizer (TSVQ) [1]–[7] is the form of vector quantization (VQ) where the codebook is grown on a binary (or M-ary) tree, thus requiring the search $\mathbf{O}(\log n)$ rather than $\mathbf{O}(n)$, for a codebook of size n. In [1], the TSVQ was designed one layer at a time using the generalized Lloyd algorithm. Each new layer of the tree was obtained by splitting each leaf node of the previous layer into two nodes, and, thus, the tree grows into a balanced tree that implements a fixed-rate code. In [2], an alternative TSVQ design algorithm was introduced. In this algorithm, the tree is grown one node at a time rather than one layer at a time, and the node that contributes most to reducing the overall distortion is selected for splitting. The tree is grown into an unbalanced tree with a predetermined number of leaves, which is a power of two. Chou *et al.* [3] proposed a method for designing an unbalanced tree coder. A balanced fixed-rate TSVQ first is grown to a predetermined height and is then optimally pruned back using the generalized Breiman, Friedman, Olshen, and Stone (BFOS) algorithm [4]. This algorithm works by always pruning the subtree with the smallest value of λ, where λ is defined as the ratio of the increase in distortion to the decrease in rate. Riskin *et al.* [5] proposed a greedy method for node splitting and tree growing. Moreover, Balakrishnan *et al.* [6] proposed a recursive splitting process to grow the tree, which outperforms [5]. Recently, the variable-length constrained-storage tree-structured vector quantization was proposed in [7]. The method proposed in [7] utilizes codebook sharing by multiple vector sources to greedily grow an unbalanced tree-structured residual vector quantizer with constrained storage. The performance of [7] is similar to [5] and [6].

Manuscript received February 5, 2003; revised January 1, 2004. This work was supported by the National Science Council of the Republic of China under Constraint NSC 91-2213-E-426-002. The associate editor coordinating the review of this manuscript and approving it for publication Dr. Zhigang Fan.

The author is with the Department of Computer Science and Information Engineering, Leader University, Tainan City, Taiwan, R.O.C. (e-mail: ysb@mail.leader.edu.tw).

Digital Object Identifier 10.1109/TIP.2004.832918

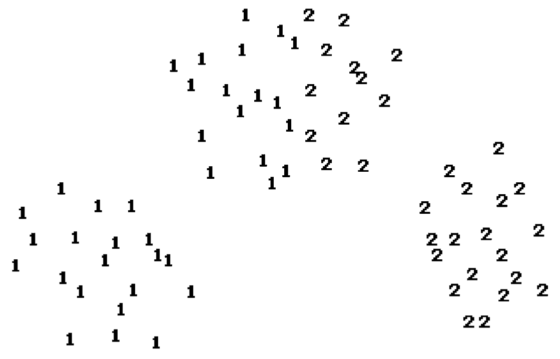


Fig. 1. Set of training samples which is not proper to be divided into two clusters.

However, all of the tree-structured coders mentioned above are fixed M-ary trees, meaning that when a node is split, the training samples contained in the node have to be divided into a fixed number of clusters. The well-known clustering method, the Linde-Buzo-Gray (LBG) algorithm [8], has been widely used for designing the TSVQs. That is, each node is split into a fixed number of child nodes when TSVQ is designed using the LBG algorithm.However, cases usually occur where it is inappropriate to divide the set of training samples into a fixed number of clusters, as illustrated in Fig. 1. The data set in Fig. 1 is not suitable for division into two clusters. That is, if a node contains the data set as in Fig. 1, that node is not suitable for division into two child nodes when the binary TSVQ is designed by using the LBG algorithm. However, the user usually has no idea regarding the number of clusters contained in the splitting node. Thus, the user usually divides each node into two (or M) child nodes, and then the binary (or M-ary) TSVQ is constructed. Actually, each node need not be divided into the same number of child nodes when TSVQ is designed. However, the LBG algorithm cannot help the user to search for the proper number of branches in TSVQ. Furthermore, [9] showed that the LBG algorithm fails to converge to a local minimum under certain conditions. Since the genetic algorithm is good at searching ([10], [11]), this paper proposes a genetic algorithm for searching a proper number of child nodes when a node is split, and then variable-branch tree-structured vector quantizer (VBTSVQ) achieves optimal coding. That is, when a node is split owing to growing the VBTSVQ, the genetic algorithm can determine the number of child nodes of this node required to maximize the value of λ.

One disadvantage of the tree-structured codebook is that the sought codeword usually differs from the codeword used in a full search. That is, the codeword sought in TSVQs is not always the one closest to the input vector. This error occurs when a

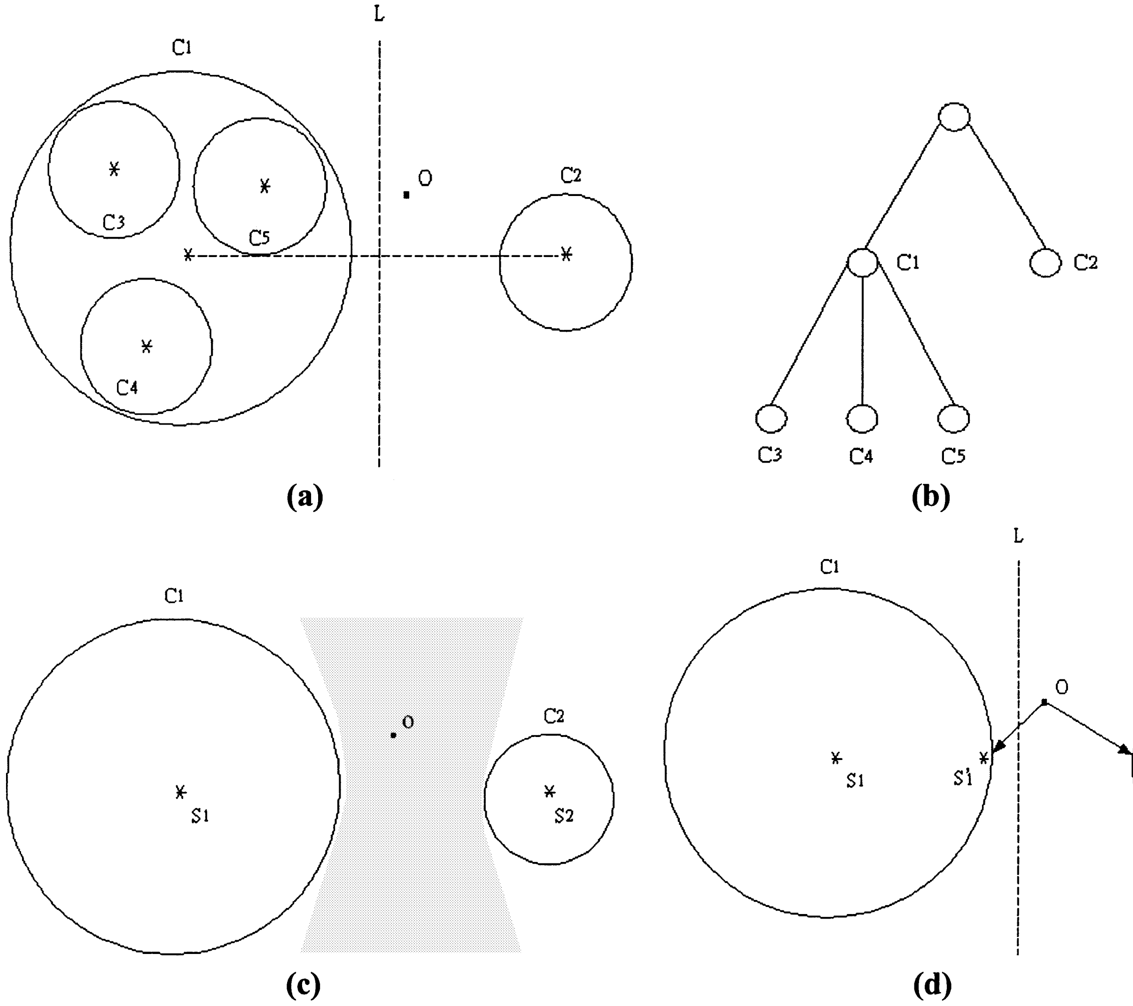


Fig. 2. Example to illustrate the classification error in the danger region.

large cluster represents a node. A single center cannot represent all objects in the large cluster. Fig. 2 shows an example illustrating the drawback of TSVQs. In Fig. 2(a), the data set is first divided into two clusters, C_1 and C_2, and cluster C_1 is then further divided into three clusters, C_3, C_4, and C_5. Fig. 2(b) shows the corresponding tree-structured codebook. If the unknown object O is encoded by the tree-structured codebook, then object O is classified into the cluster C_2, since O is closer to the center of cluster C_2 than the center of cluster C_1. A classification error occurs when C_5 is really the cluster closest to O. This kind of classification error usually occurs when the unknown object O is in the region between the two clusters. The region between the two clusters is defined as the danger region. In Fig. 2(c), the gray region indicates the danger region between the two clusters, C_1 and C_2. A large cluster requires many feature points to prevent a classification error in the danger region. For example, let each cluster contain two feature points, and let two feature points S_1' and S_2' be close to the danger region, as shown in Fig. 2(d). The unknown object O is observed to be closest to point S_1' in C_1 and, thus, O is correctly classified to cluster C_1. Therefore, multiple centers in a large cluster are required to avoid the classification error in the danger region. In [12], a search method involving a binary tree coder was proposed to improve coding quality. When the binary tree coder is traced to find a codeword besides the traced node, the node nearest to the traced node is also traced. Therefore, the search area is enlarged and a better codeword can be found. However, in [12], the cluster that represents the node in the binary tree coder is still the united-center cluster. In [13], the subtractive clustering algorithm was proposed to generate multicenter clusters. In a multicenter cluster, many centers are generated according to the density of each object in the cluster. In [14], the hierarchical subtractive clustering algorithm was proposed. This algorithm partitions the collection of objects into several subcollections and calculates the density functions for the objects in each subcollection. Also, this clustering algorithm with multicenter clusters can handle nonspherical clusters, as described in [14]. In these multicenter clustering algorithms, the search for the centers is based on the density of each object in the cluster, such that objects with high density are preferentially selected as the initial centers in the cluster. However, if object distribution in the cluster is not balanced, then the center distribution in the cluster may be skewed. In this case, centers in a cluster representing a node in TSVQ are not suitable for classifying unknown objects. In VBTSVQ, the multiclassification encoding method is proposed to search for the closest codeword to an unknown object. In VBTSVQ, many classified components can be automatically selected to represent a single cluster. Users are not concerned with the number of classified components in each cluster. Also, the minimal number of the classified components in each cluster can be automatically determined. When an unknown object is encoded in VBTSVQ, it is not compared with the center of a cluster, but rather with classified components in each cluster. Furthermore, these classified components in each cluster are close to the danger regions, and the classification error can be reduced in the danger region. In the experiment performed in this paper, the codeword encoded in VBTSVQ usually is the same as that sought in a full search.

The remainder of this paper is organized as follows. Section II presents the design of the VBTSVQ. Section III then describes the design of the multiclassification encoding method in VBTSVQ. Subsequently, experiments are given in Section IV, and, finally,Section V presents conclusions.

II. Design of the VBTSVQ

VBTSVQ consists of two parts, the variable-branch tree encoder and the Huffman tree decoder. Subsection II-A describes the design of the variable-branch tree encoder and the Huffman tree decoder with the rate constraint, while Subsection II-B presents the design of the genetic algorithm, which is used to design the variable-branch tree codebook.

A. Designs of the Variable-Branch Tree Encoder and the Huffman Tree Decoder With the Rate Constraint

The design of the growing method for the variable-branch tree encoder is given before describing the design of the variable-branch tree encoder and the Huffman tree decoder with the rate constraint. The growing method is a "top-down" approach, where we start from the root node and build a tree until the desired rate coder is obtained. In the variable-branch tree codebook, the number of child nodes for each internal node is different, and the code of each leaf node is relied on the Huffman coding [15]. In our growing approach, splitting one node at a time grows the variable-branch tree encoder T. Let S denote the set of all leave nodes in the variable-branch tree T, including node X. Let $P(X)$ represent the probability on the training samples in node X, and let $R(X)$ denote the bits required to represent node X. They are defined as

$$P(X) = \frac{\text{the number of training samples contained in the node} X}{\text{all of the training samples}} \tag{1}$$

$$R(X) = \text{ the number of bits represented by the node } X. \tag{2}$$

Finally, let $D(T)$ and $R(T)$ indicate the average distortion and rate, respectively, measured by T. The tree T' then indicates the tree T after node X is split into nodes, $X_1, X_2, \ldots, X_u$, by the GA1 algorithm, which is described in Subsection II-B. Then

$$D(T) = \sum_{\substack{i\in S\\ i\neq X}} P(i)D(i) + P(X)D(X) \tag{3}$$

$$R(T) = \sum_{\substack{i\in S\\ i\neq X}} P(i)R(i) + P(X)R(X) \tag{4}$$

$$D(T') = \sum_{\substack{i\in S\\ i\neq X}} P(i)D(i) + \sum_{i=1}^{u} P(X_i)D(X_i) \tag{5}$$

$$R(T') = \sum_{\substack{i\in S\\ i\neq X}} P(i)R(i) + \sum_{i=1}^{u} P(X_i)R(X_i). \tag{6}$$

Under the conditions of quasiconvexity, the slope of the distortion-rate function for rates between that of T and T' is

$$\begin{aligned}\lambda &= \frac{\Delta D}{\Delta R} = \frac{D(T) - D(T')}{R(T') - R(T)} \\ &= \frac{P(X)D(X) - \sum_{i=1}^{u} P(X_i)D(X_i)}{[\sum_{i=1}^{u} P(X_i)R(X_i)] - P(X)R(X)}.\end{aligned} \tag{7}$$

Thus, when splitting a node of the tree to get coder of higher rate, the aim is to maximize λ. The design of the growing method is to select the node with the largest λ to be split at a time in T.

In VBTSVQ, the user can control the width and depth of the variable-branch tree codebook. If ΔR in (7) is as large as possible, one node at a time is split into many child nodes in the variable-branch tree codebook. That is, the codebook forms a wide tree. Also, if ΔR is as small as possible, then one node at a time is split into few child nodes, and the codebook tends to grow into a deep tree. Therefore, λ in (7) can be rewritten as

$$\begin{aligned}\lambda_r &= \frac{\Delta D}{(\Delta R)^r} = \frac{D(T) - D(T')}{[R(T') - R(T)]^r} \\ &= \frac{P(X)D(X) - \sum_{i=1}^{u} P(X_i)D(X_i)}{\{[\sum_{i=1}^{u} P(X_i)R(X_i)] - P(X)R(X)\}^r}\end{aligned} \tag{8}$$

where r is a weighting factor. Notably, r is positive in (8). If r is set larger than 1, then a deep tree can be obtained. If r is within (0, 1), a wide tree is thus produced.

Designs of the variable-branch tree encoder T and the Huffman tree decoder H with the rate constraint are described as follows. Before designing the VBTSVQ, the user must give a rate threshold R and a weight r for the variable-branch tree encoder T. T is grown one node at a time by the GA1 algorithm, and the growing process continues until the average rate of T reaches the rate threshold. The algorithm for designing the variable-branch tree encoder T and the Huffman-tree decoder H with the rate constraint is described as follows.

```
Algorithm: VBTSVQ_Rate_Constraint
Input:  The rate threshold $R$, the weight $r$, and the set $L$ of $n$ objects, $O_1, O_2, \ldots, O_n$.
Output: A variable-branch tree encoder $T$ with an average rate of less than or equal to $R$ and the
    Huffman tree decoder $H$.
Step 1.  Set node $B$ as the root node of tree $T$, and node $B$ to contain all objects in $L$. Set
    $\mathrm{Rate} = 0$.
Step 2.  While The value of Rate is smaller than $R$.
    Step 2.1. For each leaf node $X$ in $T$, perform the following.
        Apply the GA1 algorithm to all objects in node $X$ to search for the child nodes
        of $X$ such that $\lambda_r$ is as large as possible.
    Step 2.2. Let $X'$ be the leaf node with the largest $\lambda_r$. Split node $X'$ in $T$ to generate
        the new encoder $T'$. Use all the leaf nodes in $T'$ to construct the
        corresponding Huffman tree decoder $H'$. Traverse each leaf node of $H'$ to
        find the corresponding code of each leaf node in $T'$.
    Step 2.3. $\mathrm{Rate} = R(T')$, where $R(T')$ is defined as (6). Set $T = T'$ and $H =$
$H'$.
Step 3.  Output the variable-branch tree encoder $T$ and the corresponding Huffman tree
decoder $H$.
    Stop.
```

B. GA1 Algorithm

In the design of the variable-branch tree encoder, one leaf node at a time is split using the genetic clustering algorithm GA1 to achieve the optimal coding in the VBTSVQ. Let X be one of the leaf nodes in T and let T' be the encoder T after X is split. The goal of GA1 is not only to search automatically for the number of child nodes of X, but also for the value of λ_r defined in (8) to be large as possible when node X is split.

Let L denote the set of objects contained in node X and the size of L be n. Suppose that L is a large data set. To achieve the performance of GA1 for processing the large data set, the pair-wise nearest-neighbor (PNN) algorithm [16] is first applied to the set L. In PNN, the two closed objects (or clusters) can be merged at a time to form a new cluster, and this merge processing is continued until the desired number of clusters is obtained. Let m clusters, $B_1, B_2, \ldots, B_{\mathrm{m}}$, be obtained after the PNN is applied to L, and let V_{i} indicate the center of cluster B_i, for $1 \leq i \leq m$. Each cluster B_i is regarded to be a component and will not be divided during the GA1 algorithm. That is, only $m (m \ll n)$ components must be further clustered in GA1. Using the PNN algorithm is an attempt to reduce the computation time in GA1. Therefore, GA1 can efficiently process the large data set.

The GA1 algorithm consists of an initialization step and iterations with three phases in each generation. They are described as follows.

1) Initialization Step: In the initialization step of GA1, a population of N strings is randomly generated. The length of each string is m, which is the number of the components obtained in the PNN algorithm. N strings are generated such that the 1 s in the strings are uniformly distributes within $[1, m]$. Each string represents a subset $\{B_1, B_2, \ldots, B_m\}$. If B_i is in this subset, then the ith position of the string will be 1; otherwise, it will be 0. Each B_i in the subset is a seed to generate a cluster.

The method for generating a clustering from the seeds is described before the three phases are elucidated. Let $R = (b_1, b_2, \ldots, b_m)$ be a bit string in the population. Each bit b_i indicates the corresponding component B_i. Then, the string R includes two sets of components, L_1 and L_2, which are defined as

$$L_1 = \{B_i \,|\, b_i = 1, 1 \leq i \leq n_1\} \tag{9}$$

$$L_2 = \{B'_j \,|\, b_j = 0, 1 \leq j \leq n_2\} \tag{10}$$

where $n_1 + n_2 = m$. In L_1, n_1 components, B_i for $1 \leq i \leq n_1$, are used as the seeds to generate n_1 clusters. Initially, each cluster C_i contains only one component, B_i, and then the center S_i of cluster C_i is set to V_i. Then, the components in L_2 are considered one at a time and the Euclidean distances between each component and the centers S_i, for $1 \leq i \leq n_1$, are calculated. Then

$$B'_j \subset C_i \quad \text{if } \|V'_j - S_i\| \leq \|V'_j - S_k\|, \quad \text{for } 1 \leq k \leq n_1. \tag{11}$$

If B'_j is classified into the cluster C_i to form the new cluster $\hat{C}_i$, then the new center $\hat{S}_i$ and the size of cluster $\hat{C}_i$ are updated as

$$\hat{S}_i = \frac{S_i|C_i| + V'_j|B'_j|}{|C_i| + |B'_j|}, \quad |\hat{C}_i| = |C_i| + |B'_j| \tag{12}$$

where $|C_i|$ and $|B'_j|$ indicates the number of objects contained in the cluster C_i and the component B'_j, respectively. After the components in L_2 have all been considered, n_1 clusters, C_i for $1 \leq i \leq n_1$, are obtained from the string R. Notably, the cluster C_i is represented as the ith child node of X.

2) Reproduction Phase: The main issue of the reproduction phase is the design of the fitness function for the string R. We emphasize that the value of λ_r for a split node must be as large as possible during the growth variable-branch tree codebook in the VBTSVQ. Let the string R generate n_1 clusters, C_i for $1 \leq i \leq n_1$. That is, n_1 child nodes of X are generated when node X is split. Let T' be the encoder after node X is split in T. Then, all leaf nodes in T' are used to construct the corresponding Huffman tree decoder H' based on the number of objects contained in each leaf node. Travel to each leaf node in H' to find the corresponding Huffman code for each leaf node in T'. Calculate λ_r when node X in T is split to generate T'. If λ_r is large, then the fitness of string R tends to be large; otherwise, the fitness of string R is small. Thus, the fitness function of string R is defined as

$$\text{Fitness}(R) = \lambda_r. \tag{13}$$

After the fitness of each string in the population is calculated, the reproduction operator is implemented using a roulette wheel with slots sized according to fitness.

3) Crossover Phase: If the crossover operator is applied to a selected pair of strings R and Q, then two random numbers e and f in $[1, m]$ are generated to decide which pieces of the strings are to be interchanged. After the crossover phase, two new strings R' and Q' replace the strings, R and Q in the population. The significance of the crossover phase is that it exchanges seeds between the different strings, to yield the various clusterings.

4) Mutation Phase: During the mutation phase, the bits of the strings in the population are chosen from $[1, m]$ with probability P_m. Each chosen bit is then changed from 0 to 1 or from 1 to 0. That is, if one bit is chosen, then a selected cluster is discarded or produced in a string. After the mutation phase, the new string R' can be obtained and replace the original string R.

The user may specify the number of generations over which he or she wants the genetic algorithm to run. The genetic algorithm runs for this number of generations and retains the string with the best fitness.

The time complexity of GA1 is analyzed as follows. The GA1 algorithm consists of an initialization step and iterations with three phases in each generation. Let the size of data set be n. Before GA1 is applied to the data set, the PNN algorithm takes $\mathbf{O}(n^2)$ time to calculate the distances between pairs of objects and takes $\mathbf{O}(n)$ time to find the minimum. In GA1, let N denote the size of population and m denote the total number of components. It takes $\mathbf{O}(m^2)$ time for each component to find the nearest cluster. The time complexity of the GA1 algorithm is dominated by the calculation of the fitness function. It takes $\mathbf{O}(Nm^2)$ time in the worst case. Suppose the GA1 algorithm is asked to run G generations, the time complexity will be $\mathbf{O}(GNm^2)$. Hence, the time complexity of the whole GA1 algorithm is $\mathbf{O}(n^2 + GNm^2)$.

III. Design of the Multiclassification Encoding Method in VBTSVQ

Subsection III-A proposes the genetic algorithm GA2 to find the classified components in each cluster. Subsection III-B then describes the multiclassification encoding method.

A. GA2 Algorithm

Before the GA2 algorithm is designed, the classified components in a cluster are first defined as follows. Let T be the variable-branch tree codebook, which is constructed by the VBTSVQ_Rate_Constraint algorithm. Each cluster that represents one node, except the root node in T, contains many classified components. Let X be an internal node, and let node X contain u child nodes in T; these u child nodes are represented by u clusters, $C_1, C_2, \ldots, C_u$, respectively. The following describes how to define the classified components in these child nodes of X. Let cluster C_i contain m_i classified components, $\tilde{B}_i^1, \tilde{B}_i^2, \ldots, \tilde{B}_i^{m_i}$, for $1 \le i \le u$. Then, the distance between object O and cluster $C_i, \mathrm{dis}(O, C_i)$, is defined as

$$\mathrm{dis}(O, C_i) = \min_{1 \le j \le m_i} \left\| O - \tilde{V}_i^j \right\| \tag{14}$$

where $\tilde{V}_i^j$ denotes the center of the classified component $\tilde{B}_i^j$. Then, the classified components in the cluster C_i satisfy

$$\mathrm{dis}(O, C_i) < \min_{1 \le j \le u} \mathrm{dis}(O, C_j) \tag{15}$$

for each $O \in C_i$. Thus, when an unknown object reaches node X by tracing the variable-branch tree codebook T, the unknown object should be compared with the classified components in these u child nodes to determine which one of these u child nodes should be traced in the next step.

Three design issues raised by the GA2 algorithm are described as follows. First, the GA2 algorithm searches for the classified components in each cluster, all of the objects in which satisfy (15). Second, the GA2 algorithm required that the density of each classified component is as large as possible and that the classified components in each cluster are close to the danger regions, improving the classification errors in the danger regions. Third, GA2 searches for the minimal number of classified components required in each cluster, reducing the encoding time.

Like GA1, before the GA2 algorithm is used to find the classified components of the clusters, the PNN algorithm is applied to all the objects to obtain a set of components in each cluster. The GA2 algorithm is described as follows. The GA2 algorithm has two stages. In the first stage of GA2, many nearest-neighbor components are selected from each cluster. These nearest-neighbor components in a cluster are the components that are closest to the other clusters. In general, these nearest-neighbor components in the cluster are also the components closest to the danger regions. The second design issue associated with GA2 concerns the fact that the classified components in each cluster should be close to the danger regions. Thus, the second stage of GA2 not only searches for the classified components of each cluster, but also that these classified components are close to the nearest-neighbor components in the cluster.

The first stage of the GA2 algorithm is as follows.

```
**Input:** $u$ clusters, $C_1, C_2, \ldots, C_u$.
**Output:** Sets of nearest-neighbor components of cluster $C_i$, $S(C_i)$, for $1 \le i \le u$.
**Step 1.** For each cluster $C_i$, for $1 \le i \le u$, do the following.
    **Step 1.1.** Set $S(C_i)$ to empty.
    **Step 1.2.** For each object $O$ in the cluster $C_j$, $i \ne j$, do the following.
        Calculate the Euclidean distances between the object $O$ and each component
        $B_i^k$ in cluster $C_i$. Let $\hat{B}_i^p \in C_i$, and $\hat{B}_i^p$ satisfy $\|\hat{V}_i^p - O\| =$
$\min_{B_i^k \in C_i} \|V_i^k - O\|$. If
        $\hat{B}_i^p$ is not in the set $S(C_i)$, then $\hat{B}_i^p$ is regarded as a nearest-neighbor
        component in cluster $C_i$, and $\hat{B}_i^p$ is added to the set $S(C_i)$.
**Step 2.** Output the $u$ sets, $S(C_i)$ for $1 \le i \le u$. End.
```

Fig. 3 shows an example to illustrate nearest-neighbor components. In Fig. 3, the large black points represent the large components. Let the objects in node X be clustered into four clusters, C_1, C_2, C_3, and C_4, by the GA1 algorithm, as shown in Fig. 3(a). In Fig. 3(b), each component in a solid-line circle represents the nearest-neighbor component.

The second stage of GA2 is a genetic algorithm, which is applied to these u clusters to find the classified components. The method by which the second stage of GA2 searches for the classified components of cluster C_i using the set $S(C_i)$ is described below. The initialization step and three phases in the genetic algorithm are described as follows.

1) Initialization Step: In the initialization step, a population of N binary strings is randomly generated. The length of the binary string m_i is the total number of components in cluster C_i. N binary strings are generated in such a way that the numbers of 1s in the strings uniformly distributes within $[1, m_i]$. Each string represents a subset of C_i. If B_i^k is in this subset, the kth bit in the string will be 1; otherwise, it will be 0. Each component, B_i^k, in the subset indicates a classified component in C_i.

2) Reproduction Phase: Let $R = (b_1, b_2, \ldots, b_{m_i})$ be a binary string in the population. Each bit b_k represents the corresponding component B_i^k. If bit $b_k = 1$, the corresponding component B_i^k is checked to determine whether it is a classified component in the string R. Let the set Y consist of these components such that

$$Y = \left\{ B_i^k \mid b_k = 1 \right\}. \tag{16}$$

If these components in Y are all classified components of cluster C_i, then each object $O_\mathrm{p} \in C_i$ must satisfy

$$\mathrm{dis}(O_p, C_i) < \min_{\substack{1 \le j \le u \\ i \ne j}} \min_{O_q \in C_j} \|O_p - O_q\|. \tag{17}$$

Notably, (17) also satisfies the definition of classified components in (15), since if $O_\mathrm{p} \in C_i$, we have

$$\min_{\substack{1 \le j \le u \\ i \ne j}} \min_{O_q \in C_j} \|O_p - O_q\| \le \min_{\substack{1 \le j \le u \\ i \ne j}} \mathrm{dis}(O_p, C_j). \tag{18}$$

Therefore, if each $O_p \in C_i$ satisfies (17), Y is regarded as the set of all classified components of cluster C_i. The fitness function calculates the number of objects that satisfy (17) in cluster C_i. If object $O_p \in C_i$ satisfies (17), then δ_p is set to 1; otherwise, δ_p is set to 0. Let

$$\delta = \sum_{p=1}^{|C_i|} \delta_p. \tag{19}$$

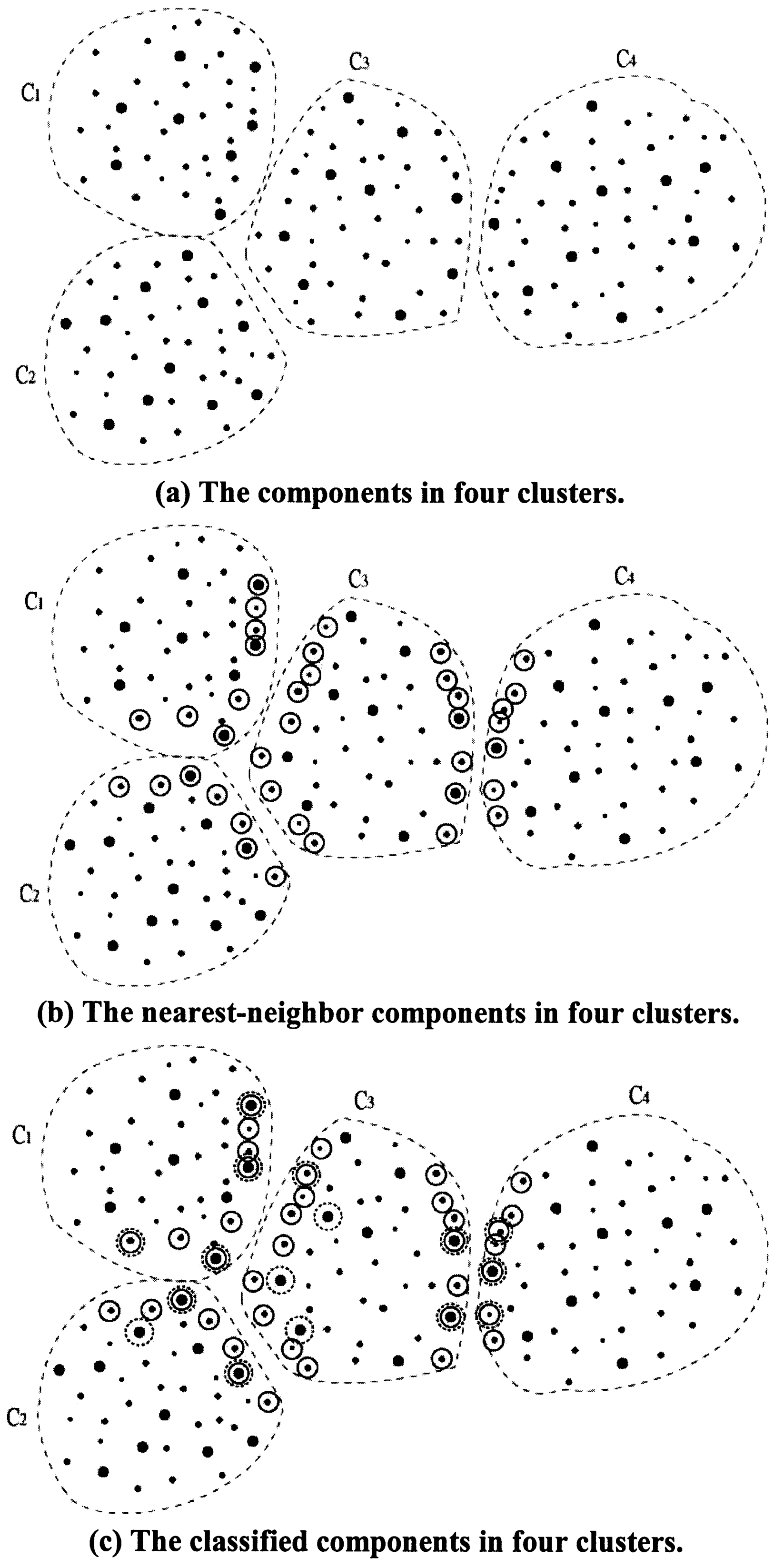

C1
C2
C3
C4
(a) The components in four clusters.
(b) The nearest-neighbor components in four clusters.
(c) The classified components in four clusters.

Fig. 3. Example to illustrate the classified components in four clusters.

The first design issue concerning the fitness function emphasizes that all the classified components can represent all the objects in the cluster. Thus, δ must be as large as possible. If $\delta = |C_i|$, the components in Y can represent all objects in cluster C_i.

The second design issue of the fitness function emphasizes that the density of each classified component should be as large as possible because a classified component with a high density can represent many objects in a cluster. Furthermore, the classified components are required to be close to the nearest-neighbor components in a cluster to reduce the error of classification in the danger regions. Thus, Ω is defined as

$$\Omega = \sum_{j=1}^{|S(C_i)|} \max_{1\leq k\leq |Y|} \frac{D_i^k}{\left\| \hat{V}_i^j - V_i^k \right\|} \tag{20}$$

where $\hat{V}_i^j$ is the center of the nearest-neighbor component $\hat{B}_i^j, D_i^k$ is the density of the component $B_i^k \in Y$ and D_i^k is defined as

$$\begin{aligned} D_i^k &= \text{number of objects contained in } B_i^k, \\ &= \left| B_i^k \right|. \end{aligned} \tag{21}$$

Notably, if the component B_i^k in Y is the same as the nearest-neighbor component $\hat{B}_i^j$, then $||\hat{V}_i^j - V_i^k||$ is set to a small positive value ε in (20).

The third design issue concerning the fitness function emphasizes that the number of classified components in Y must be as small as possible to reduce the encoding time in VBTSVQ. To summarize the three aforementioned design issues, the fitness function for string R is defined as

$$\text{Fitness}(R) = \begin{cases} \frac{\delta}{|C_i|} & \text{if } \delta < |C_i| \\ 1 + \frac{\Omega}{|Y|} & \text{if } \delta = |C_i| \end{cases}. \tag{22}$$

In (22), if $\delta < |C_i|$, the components in Y cannot represent all objects in cluster C_i, and the Fitness(R) is then set to a small value within $[0, 1]$; otherwise, Fitness(R) exceeds 1. After the fitness of all binary strings in the population is calculated, the reproduction operator is implemented as using a roulette wheel with slots sized according to fitness.

3) Crossover and Mutation Phases: The crossover and mutation phases in the genetic algorithm are similar to that in GA1. Two random numbers in $[1, m_i]$ are generated to decide which pieces of two strings are to be interchanged in the crossover phase. Also, the crossover operator is done with probability P_c. Furthermore, in the mutation phase, the bits of each string R in the population are chosen from $[1, m_i]$ with probability P_m. Each chosen bit is then changed from 0 to 1 or from 1 to 0.

After the GA2 algorithm is applied to cluster C_i, the string R'' with the best fitness is reserved. Let the set Y contain components, each of which has a corresponding bit that equals to 1 in R''. Then, each component B_i^k in Y is set as the classified component $\tilde{B}_i^k$ in cluster C_i. Fig. 3(c) shows an example to illustrate the classified components. In Fig. 3(c), the components in the dotted circle are classified components.

In the following, the time-complexity of GA2 is analyzed when the GA2 algorithm is used to find the classified components of cluster C_i. Let there be u clusters, and the number of all objects contained in these u clusters be n. Let the number of objects contained in C_i be q. The time complexity of the first stage in GA2 is dominated by Step 1, which takes $\mathbf{O}(nq)$ time to calculate the Euclidean distances between pairs of objects to determine the nearest-neighbor components of cluster C_i. In the second stage, let N denote the size of population. The time complexity of the second stage is dominated by the calculation of the fitness function. It takes $\mathbf{O}(nq)$ time to calculate the value of δ. Thus, the reproduction phase takes $\mathbf{O}(Nnq)$ time in the worst case. Suppose the genetic algorithm is asked to run G generations, the time complexity will be $\mathbf{O}(GNnq)$. Hence, the time complexity of the whole clustering algorithm is $\mathbf{O}(GNnq)$.

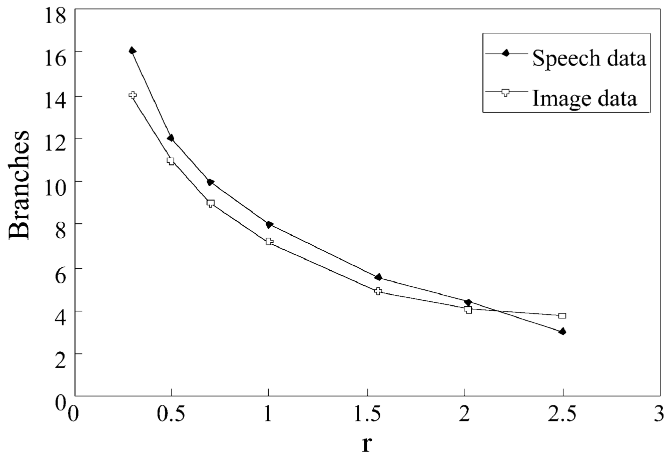


Fig. 4. Relation between the values of r and the average number of branches in the variable-branch tree codebook.

B. Multiclassification Encoding Method

If an unknown object O is encoded in VBTSVQ, then O is traced from the root node to the leaf nodes in the variable-branch tree codebook, it is then encoded using the codeword in the closest leaf node. Let X be an internal node that consists of u child nodes in the variable-branch tree codebook. The set of objects in the ith child node of X is represented as the cluster C_i, for $1 \leq i \leq u$. The following describes how to classify the unknown object O into one of these u child nodes, when the object O reaches the internal node X by tracing the variable-branch tree codebook. The traditional encoding method, the united-center encoding method, uses the Euclidean distance between the object O and the center S_i to measure the closeness of object O to cluster C_i. In the proposed encoding method, the distances between O and the classified components in cluster C_i are used to measure the closeness of O to the cluster C_i. Let cluster C_i contain m_i classified components, $\tilde{B}_i^1, \tilde{B}_i^2, \ldots, \tilde{B}_i^{m_i}$, for $1 \leq i \leq u$, and let

$$\gamma = \arg \min_{1\leq i\leq u} \text{dis}(O_x, C_i) \tag{23}$$

where $\text{dis}(O_x, C_i)$ is defined in (14). Then, the unknown object O is classified into the cluster C_γ. Therefore, the object O proceeds to the γth child node of node X to continue to trace the variable-branch tree codebook. This tracing process is continued until O reaches one of the leaf nodes in the variable-branch tree codebook; then, the codeword contained in the leaf node is regarded as the encoding result of object O.

The time complexity of the multiclassification encoding method is described as follows. Suppose that the total number of leaf nodes in the variable-branch tree codebook is n; the average number of child nodes of each internal node in the variable-branch tree codebook is u, and the average number of classified components in each node is q. Then, the time complexity is approximately to $\mathbf{O}(qu \log_u(n))$ for the proposed encoding method of VBTSVQ.

TABLE I
CODING QUALITIES OF VBTSVQ WHEN THE VARIOUS VALUES OF r ARE USED IN λ_r. (a) SPEECH DATA. (b) IMAGE DATA

(a)

Experiments	Rate (bits/sample)	SNR (dB)						
		r=0.3	r=0.5	r=0.7	r=1	r=1.5	r=2	r=2.5
(1)	0.3	12.3	12.1	11.8	11.4	11.0	10.8	10.5
(2)	0.6	19.7	19.4	19.1	18.9	18.5	18.3	18.1
(3)	0.9	26.2	25.9	25.4	25.1	24.7	24.4	24.2
(4)	1.2	32.4	31.9	31.4	30.8	30.0	29.3	28.8
(5)	1.5	39.1	38.3	37.3	36.2	35.1	34.0	33.2

(b)

Experiments	Rate (bits/pixel)	PSNR (dB)						
		r=0.3	r=0.5	r=0.7	r=1	r=1.5	r=2	r=2.5
(1)	0.15	28.8	28.7	28.7	28.6	28.6	28.5	28.4
(2)	0.25	29.5	29.5	29.4	29.4	29.3	29.3	29.2
(3)	0.35	30.6	30.6	30.4	30.4	30.3	30.3	30.2
(4)	0.45	31.4	31.4	31.3	31.3	31.2	31.2	31.1
(5)	0.55	33.1	32.8	32.6	32.4	32.3	32.2	32.1

IV. EXPERIMENTS

This section compares the performance of VBTSVQ with that of the other TSVQs. Furthermore, the multiclassification encoding method and other encoding method are compared below.

A. Data Sets

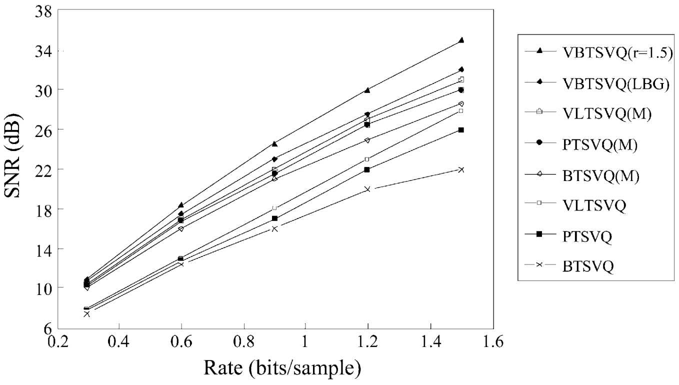
In the experiments, two data sets, namely speech and image data sets, were used to test the performance of VBTSVQ and the other methods, respectively. In the image data set, five 512×512 (pixels) images with 256 gray levels were employed as the training images, and were divided into 4×4 blocks to design the VBTSVQ and the other TSVQs. The "Lena" image was not used in training to test the performance of these methods. In the speech data set, a total of 100 000 spectral feature vectors of speech were used as training objects to design the VBTSVQ and the other methods. These spectral feature vectors were taken from 1000 continuous speeches, given by five males and five females. Each spectral feature vector contained 64 sample points. Furthermore, 50 000 spectral feature vectors were extracted as testing objects not used in training.

B. Performance of VBTSVQ

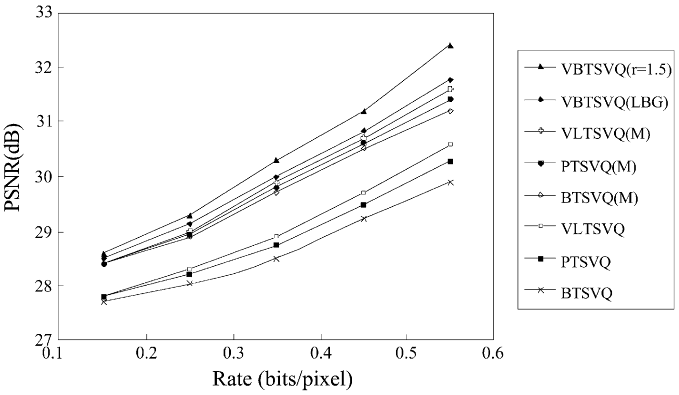
VBTSVQ uses the GA1 algorithm to construct the variable-branch tree codebook. The parameters used in GA1 were population size 100, crossover rate 80%, and mutation rate 5%. Five hundred generations were run and the best solution was retained. Before GA1 was applied to the training data set, the PNN algorithm was applied to all training objects to obtain a small set of components. In the experiments, the size of the component set was set to a fifth of the total number of training objects. In GA1, the weight r is used to control the width of the variable-branch tree codebook. Fig. 4 shows the average number of branches of an internal node in the variable-branch tree codebook when various values of r are used in (8). If r is large, then a narrow variable-branch tree codebook is obtained; otherwise, a wide variable-branch tree codebook is obtained. Table I lists the coding qualities of VBTSVQ for various r in λ_r. In Table I, the coding quality of VBTSVQ is enhanced with r because the variable-branch tree codebook tends to grow into a wide tree with decreasing r, and, thus, the unknown object is compared

TABLE II
ENCODING TIME OF VBTSVQ WHEN THE VARIOUS VALUES OF r ARE USED IN λ_r

Data sets	Ave. Encoding Time (second)						
	r=0.3	r=0.5	r=0.7	r=1	r=1.5	r=2	r=2.5
Speech data	8.23	8.01	7.89	7.72	7.65	7.54	7.46
Image data	8.13	7.98	7.72	7.62	7.53	7.44	7.34



(a)



(b)

Fig. 5. Comparison of the performance of the VBTSVQ and the other TSVQs. (a) Speech data. (b) Image data.

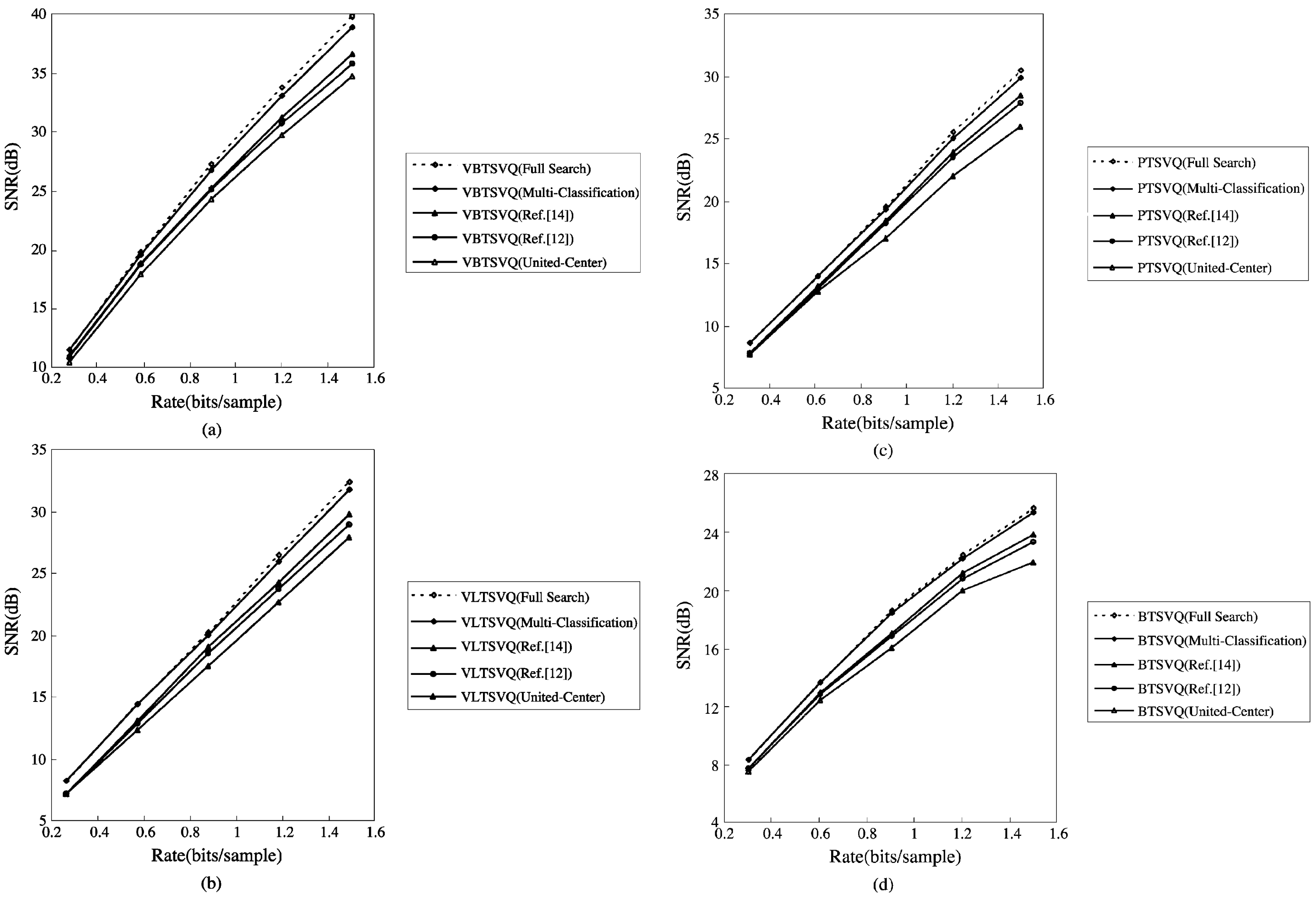


Fig. 6. Performance comparison between the multiclassification encoding method and the other encoding methods for speech data. (a) The performance of the various encoding methods in VBTSVQ. (b) The performance of the various encoding methods in VLTSVQ. (c) The performance of the various encoding methods in PTSVQ. (d) The performance of the various encoding methods in BTSVQ.

with more nodes at a time to determine which node should be traced in the wide variable-branch tree codebook. Clearly, if the variable-branch tree codebook is too wide, then the coding quality of VBTSVQ approaches that obtained by the full search method. Table II lists the encoding times when various values of r are used to design the variable-branch tree codebook. Although the coding quality of VBTSVQ obtained using the wide variable-branch tree codebook is higher than that using the narrow variable-branch tree codebook, the encoding time required by the wide variable-branch tree codebook exceeds that required by the narrow variable-branch tree codebook. If the user wishes to minimize the encoding time, a narrow variable-branch tree codebook is preferred; otherwise, a wide variable-branch tree codebook is used to enhance the coding quality. Fig. 5 compares the VBTSVQ with the other three methods, Balakrishnan's method [8], Chou's method [6], and the balanced binary-tree VQ, which were implemented for comparison with the proposed method. These three methods were called VLTSVQ, PTSVQ, and BTSVQ, respectively. In PTSVQ, the tree was grown to a height of 12 layers before pruning. Also, the three TSVQs were easily designed using M-ary trees, namely VLTSVQ(M), PTSVQ(M), and BTSVQ(M), where M denotes the average number of child nodes in the variable-branch tree codebook designed by VBTSVQ. In Fig. 5, r was set to 1.5 to design a variable-branch tree codebook, and the average number of child nodes of an internal node in the variable-branch tree codebook was approximately six for speech training data set and five for image training data set. Thus, M was set to 6 and 5 in these three TSVQ(M)s for testing speech and image data sets, respectively. Also, the Huffman coding was used in the TSVQ(M)s, as VBTSVQ. Fig. 5 fairly compares these methods: the GA1 algorithm first was used to design the variable-branch tree codebook, and then the other TSVQs were applied separately to obtain equal sized codebooks using the LBG algorithm. VBTSVQ(LBG) uses the LGB algorithm to design the variable-branch tree codebook, with the same structure obtained using VBTSVQ$(r = 1.5)$. Additionally, VBTSVQ and the other methods use the same traditional united-center encoding method to encode an unknown object. In the united-center encoding method, the unknown object needed only to be compared with the center in each internal node, such that the unknown object can decide which child node should be forwarded. In Fig. 5, the VBTSVQ$(r = 1.5)$ has higher coding quality than the other methods for a given bit rate, because the GA1 algorithm can determine the appropriate number of child nodes of each internal node required to optimize the coding in the variable-branch tree codebook, while the number of child nodes of each internal node is fixed in the other TSVQs. VBTSVQ actually outperforms the other fixed-branch TSVQs. Furthermore, although VBTSVQ$(r = 1.5)$ and aVBTSVQ(LBG) have the same variable-branch tree codebook structure, the coding quality

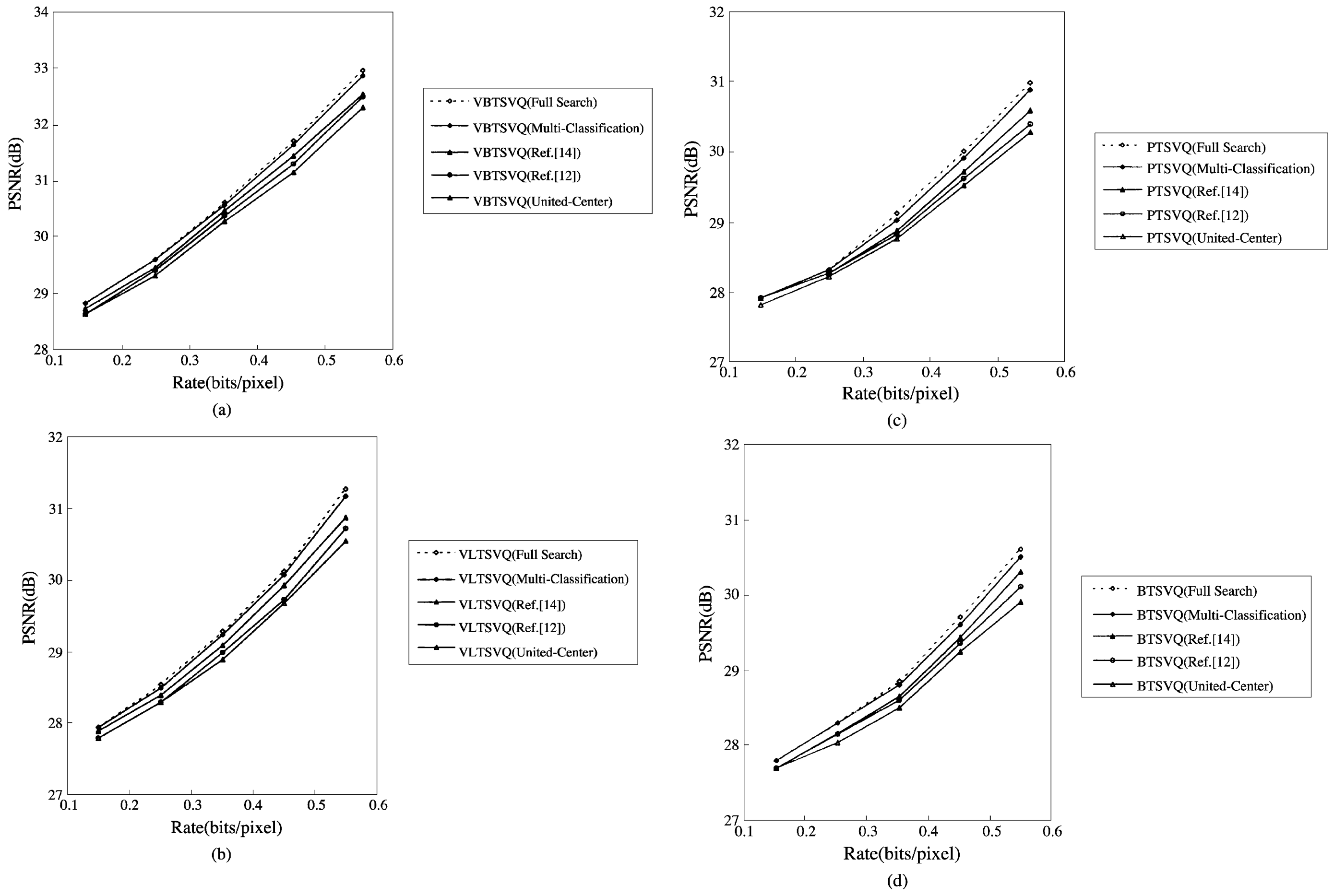


Fig. 7. Performance comparison between the multiclassification encoding method and the other encoding methods for image data. (a) The performance of the various encoding methods in VBTSVQ. (b) The performance of the various encoding methods in VLTSVQ. (c) The performance of the various encoding methods in PTSVQ. (d) The performance of the various encoding methods in BTSVQ.

obtained using VBTSVQ($r = 1.5$) is better than that obtained using VBTSVQ(LBG) because the GA1 algorithm can search for a better clustering result than the LBG algorithm for a given data set.

C. Performance of the Multiclassification Encoding Method

The efficiency of the multiclassification encoding method was compared with that of the united-center encoding method, Chang's method [12] and Tao's method [14]. The parameters used in GA2 were population size 100, crossover rate 80%, and mutation rate 5%. The algorithm was run for 500 generations and the best solution was retained. In Tao's method, the number of cluster centers was set to the average number of classified components in that cluster in the multiclassification encoding method. Figs. 6 and 7 compare these encoding methods. In Figs. 6 and 7, the multiclassification encoding method outperforms the other encoding methods. Figs. 6 and 7 present two phenomena. First, the multiclassification encoding method and the method in [14] outperform the united-center encoding method and that in [12], because the latter two use only one center to represent all objects in a cluster, and a classification error generally arises when the cluster is large or nonspherical. Second, the coding quality of the multiclassification encoding method is better than that of the method in [14]. In [14], the cluster centers are determined according to the density of each object in the cluster. The initial cluster centers thus are sensitive to object distribution. In the multiclassification encoding method, classified component selection in each cluster is determined based on the density of each component, and the classified components in each cluster are close to the other clusters. The classification error in the danger regions between any two pairs of clusters thus can be reduced. Table III lists the search rates of the various encoding methods in VBTSVQ, where search rate indicates the percentage similarity between the sought codeword and that sought by the full search method. The performance of the multiclassification encoding method is close to that of the full search method, as shown in Figs. 6 and 7.

V. Conclusion

TSVQ has the advantage of more efficient codebook searches than traditional full-search vector quantizers. However, dividing a set of training samples into a fixed number of clusters in TSVQ is sometimes inappropriate. Specifically, such division generally increases either the bit rate or the average distortion, or both. This paper thus proposed VBTSVQ. The GA1 algorithm is proposed to automatically search for the appropriate number of branches of each splitting node to maximize the value of λ, which is the slope of the distortion-rate function. Moreover, the threshold r also is provided to control the width and depth of the variable-branch tree codebook in VBTSVQ. Although the GA1

TABLE III
SEARCH RATE OF THE COMPARISON OF THE VARIOUS CODING METHODS IN VBTSVQ($r = 1.5$). (a) SPEECH DATA. (b) IMAGE DATA

(a)

Methods	Search Rates(%)				
	(1)	(2)	(3)	(4)	(5)
United-Center Encoding Method	93.5	91.4	89.1	87.9	85.2
Ref. [12]	97.4	95.5	93.5	91.2	90.1
Ref. [14]	98.9	97.1	95.2	92.9	91.8
Multi-Classification Encoding Method	100.0	99.3	98.7	98.1	97.5

(b)

Methods	Search Rates(%)				
	(1)	(2)	(3)	(4)	(5)
United-Center Encoding Method	94.2	92.7	90.4	88.3	86.8
Ref. [12]	97.8	95.8	94.1	92.5	90.3
Ref. [14]	99.1	97.9	96.4	93.9	92.5
Multi-Classification Encoding Method	100.0	100.0	99.3	98.8	98.1

algorithm requires more time to design the VBTSVQ than the LBG algorithm requires to design the fixed-branch TSVQ, VBTSVQ outperforms the fixed-branch TSVQ, as shown in our experiments.

This paper proposes the multiclassification encoding method that uses the classified components. The GA2 algorithm not only searches for classified components to represent each cluster, but also requires that these classified components are close to other clusters. Hence, the classification error is reduced in the danger regions. Also, users do not care about the number of classified components in each cluster, because the GA2 algorithm can seek the minimum number of classified components of each cluster. In the experiments conducted here, the search rate of the multiclassification encoding method exceeds that of the other encoding methods.

REFERENCES

[1] A. Buzo, A. H. Gray Jr., R. M. Gray, and J. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 562–574, May 1985.

[2] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol. 73, pp. 1551–1588, Nov. 1985.

[3] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inform. Theory*, vol. 35, pp. 299–315, Mar. 1989.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, ser. The Wadsworth Statistics/Probability Series. Belmont, CA: Wadsworth, 1984.

[5] E. A. Riskin and R. M. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Trans. Signal Processing*, vol. 39, pp. 2500–2507, Nov. 1991.

[6] M. Balakrishnan, W. A. Pearlman, and L. Lu, "Variable-rate tree-structured vector quantizers," *IEEE Trans. Inform. Theory*, vol. 41, pp. 917–930, July 1995.

[7] U. Bayazit and W. A. Pearlman, "Variable-length constrained-storage tree-structured vector quantization," *IEEE Trans. Image Processing*, vol. 8, pp. 321–331, Mar. 1999.

[8] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, Jan. 1980.

[9] S. Z. Selim and M. A. Ismail, "K-means-type algorithm: Generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 81–87, Jan. 1984.

[10] M. Srinivas and M. Patnaik, "Genetic algorithm—A survey," *IEEE Computer*, vol. 27, pp. 17–26, June 1994.

[11] D. E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[12] C. C. Chang and T. S. Chen, "New tree-structured vector quantization with closed-coupled multipath searching method," *Opt. Eng.*, vol. 36, no. 6, pp. 1713–1720, 1997.

[13] S. L. Chiu, "Fuzzy model identification based on cluster estimation," *J. Intell. Fuzzy Syst.*, vol. 2, no. 23, pp. 267–278, 1994.

[14] C. W. Tao, "Unsupervised fuzzy clustering with multicenter clusters," *Fuzzy Sets Syst.*, vol. 128, pp. 305–322, 2002.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[16] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust. Speech Signal Processing*, vol. ASSP-32, pp. 1568–1575, Sept. 1989.

Shiueng-Bien Yang received the B.S. and the Ph.D. degrees from the Department of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan, R.O.C., in 1993 and 1999, respectively.

He is currently an Associate Professor with the Department of Computer Science and Information Engineering, Leader University, Tainan City, Taiwan. His research interests include pattern recognition, speech coding, image processing, and neural networks.