Fast Codebook Search Algorithms Based on Tree-Structured Vector Quantization

Chin-Chen Chang[1,2], Yu-Chiang Li[2], and Jun-Bin Yeh[2]

[1]Department of Information Engineering and Computer Science,
Feng Chia University, Taichung 40724, Taiwan, R.O.C.
[2]Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi 621, Taiwan, R.O.C.
E-mail: {ccc, lyc, ycp92}@cs.ccu.edu.tw

Abstract. Tree-Structured Vector Quantization (TSVQ) is a highly efficient technique for locating an appropriate codeword for each input vector. The algorithm does not guarantee that the selected codeword is the closest one to the input vector. Consequently, the image quality of TSVQ is worse than that of full-search VQ (FSVQ). Although researchers have proposed multipath TSVQ and DP-TSVQ to enhance the image quality, these methods are still too slow for achieving high image quality. Therefore, this study presents a novel Full Search Equivalent TSVQ (FSE-TSVQ) to obtain efficiently the closest codeword for each input vector. FSE-TSVQ employs the triangle inequality to achieve efficient pruning of impossible codewords. Moreover, this study also develops the Enhanced DP-TSVQ (EDP-TSVQ) algorithm, which achieves a better tradeoff than DP-TSVQ between encoding time and image quality. EDP-TSVQ is a hybrid technique which adds DP-TSVQ's critical function to FSE-TSVQ. EDP-TSVQ always provides an image quality identical to that of DP-TSVQ, but by searching fewer codebook tree nodes. Simulation results reveal that FSE-TSVQ requires only 21% to 38% of the running time of FSVQ. For a high image quality application, the performance of EDP-TSVQ is always better than that of DP-TSVQ. Using the example of a codebook tree with 512 codewords, with the threshold of the critical function set to 0.6, simulation results indicate that EDP-TSVQ requires only 37% of the execution time of DP-TSVQ.

Keywords. Codebook search, tree-structured vector quantization, codebook, vector quantization

1 Introduction

Recent developments in multimedia and computer networks have resulted in the widespread use of electronic file transmissions to replace traditional postal mail. However, due to the large size of multimedia data files and the bandwidth restrictions of computer networks, data transmission is inefficient. Data can be compressed to reduce its size, improving the efficiency of its transmission across computer networks. Vector Quantization (VQ) is a basic efficient coding technique for speech and image compression [1, 16]. In recent years, VQ has been successfully used in numerous applications including image encoding and image recognition [8].

Traditionally, VQ is separated into three phases: *codebook generation*, *vector encoding*, and *vector decoding* [7]. In the first phase, Linde-Buzo-Gray (LBG), one of the best-known algorithms for this purpose, is used to develop the codebook [1]. The LBG algorithm decomposes all training images into a training set of small rectangular blocks (called vectors or codewords). The size of each block is $w \times h = k$. Each vector is therefore a k-dimensional codeword containing k pixel values. Next, LBG randomly selects n codewords as centroids and generates the initial codebook CB ($CB = \{c_i \mid i = 1, 2, \ldots, n\}$, where c_i is a codeword). Subsequent iterations can improve the quality of the codebook. In each iteration, LBG computes the distortions between a training vector and each centroid. Each training vector is placed in the closest c_i class. LBG then calculates the total distortion between each training vector and each centroid, replaces the centroid c_i with the closest training vector, and repeats the process until the reduction in total distortion is insignificant. The distortion between the input vector X and a codeword c_i is defined as their *squared Euclidean distance* (*mean squared error*) or their *Euclidean distance*. Equations (1) and (2) describe respectively the mean squared error (MSE) and the Euclidean distance between two vectors.

$$MSE(X, c_i) = \|X - c_i\|^2 = \sum_{j=1}^{k} (X_j - c_{ij})^2 , \tag{1}$$

$$d(X, c_i) = \|X - c_i\| = \sqrt{\sum_{j=1}^{k} (X_j - c_{ij})^2} , \tag{2}$$

where X_j and c_{ij} represent the value of the jth dimension of X and the jth dimension of c_i, respectively.

In the second phase, the VQ encoder breaks up the source image into a number of k-dimensional vectors. The encoder searches the codebook CB to obtain the closest codeword c_i for each input vector X. VQ then encodes the input vector X into an index i, whose size is only $\log_2 n$ bits. This compression significantly reduces the size of the digital file. This phase is also called *codebook matching*.

In the third phase, the decoder replaces each index value i in the compressed image with the corresponding codeword c_i. In this way, the VQ decoder can easily reconstruct the image.

A major issue in VQ is finding an efficient method of matching the closest codeword to an input vector from the codebook. The full-search VQ (FSVQ) algorithm is the simplest method. However, this exhaustive search approach is time-consuming. Researchers have proposed numerous fast search approaches to speed up the codebook matching process, including TSVQ [1, 2, 3, 6, 9, 11], look-up table [4, 19], Pyramid [12, 17, 20], the transform method [13], and other techniques [5, 10, 14, 15, 18].

Among these methods, TSVQ is a very important and efficient approach. It uses a binary tree to decrease the computational complexity of the search for the closest codeword. The original single-path search TSVQ [1] reduces the computational complexity of codebook matching from $O(kn)$ to $O(k\log n)$. However, the selected codewords are not, in general, the nearest ones to the input vectors. Consequently, the image quality of single-path TSVQ is much poorer than that of FSVQ.

The multipath TSVQ algorithm has been proposed to improve the image quality by increasing the number of search paths traversed to find the closest codeword [3]. For the same purpose, Closest-Couple TSVQ (CC-TSVQ) enlarges the search range of each search path to match more approximate codewords [2] than the single one obtained with multipath TSVQ. Nevertheless, multipath TSVQ and CC-TSVQ do not eliminate the wrong direction paths, which have large distortions between the leaf nodes and the input vectors. Dynamic Path TSVQ (DP-TSVQ) defines a critical function with the intention of improving the accuracy of TSVQ search results [6].

The various TSVQ approaches do not guarantee that the selected codeword is the closest one to an input vector. Multipath TSVQ and DP-TSVQ add more search paths to improve the image quality. However, these methods do not include an efficient

strategy for producing a high-quality image. This study therefore proposes a novel Full Search Equivalent TSVQ (FSE-TSVQ) algorithm for fast encoding. To provide further acceleration of the encoding phase, this study develops an Enhanced DP-TSVQ (EDP-TSVQ) algorithm with a better time-quality tradeoff than DP-TSVQ. To simplify the discussion, this study focuses on accelerating the codebook matching in the encoding phase using TSVQ, assuming that the codebook tree is already given.

The rest of this paper is organized as follows. Section 2 introduces the background and different versions of TSVQ. Section 3 then describes the first scheme to be developed, which is a fast full search equivalent TSVQ (FSE-TSVQ) algorithm based on the triangle inequality. In addition, Section 4 explains the proposed Enhanced DP-TSVQ (EDP-TSVQ) algorithm, and demonstrates that EDP-TSVQ and DP-TSVQ have identical image qualities. Section 5 provides experimental results and evaluates the performance of the proposed algorithms. Finally, Section 6 presents conclusions.

2 Related Work

2.1 General Remarks

TSVQ [1, 8, 9] is an efficient method for reducing the complexity of a codebook search. A binary tree structure is used to represent the codebook, which is called the *codebook tree*. All internal nodes are used to partition codewords into two groups. Each leaf node stores a codeword. The codebook tree is created in the codebook generation phase.

Moving from the root to the leaf node, the single-path TSVQ algorithm chooses the nearest child node as the descent path for each input vector. Finally, the selected leaf node is treated as the closest codeword. Although single-path TSVQ is fast, the probability of locating the true closest codeword is low. Multipath TSVQ has therefore been proposed to improve image quality [3].

2.2 Multipath TSVQ

The multipath TSVQ algorithm has been proposed to alleviate the problem of poor image quality produced by single-path TSVQ [3]. It follows m paths to search for the closest codeword. At each level of the codebook tree, multipath TSVQ compares $2m$ child nodes and selects m nearest internal nodes for each input vector. At the leaf level, multipath TSVQ examines m candidate codewords and selects from these candidates the one closest to the input vector. Consequently, multipath TSVQ produces a higher image quality than that of single-path TSVQ. If $m = 1$, multipath TSVQ degenerates into single-path TSVQ. The more search paths are traced in TSVQ, the more closer codeword will be found. However, multiple search paths increase the computational complexity significantly, and most search paths used by multipath TSVQ are in fact redundant. Furthermore, multipath TSVQ does not always match the global closest codeword to the input vector.

Example 2.1 Consider the codebook tree in Fig. 1 with eight two-dimensional codewords and input vector $X = (100, 100)$. Except for the root, each node has an integer number outside the node, which indicates the mean squared error between the input vector X and the corresponding node. The two paths followed by two-path TSVQ are *A-D-I* and *B-E-K*. Two-path TSVQ selects the codeword *I* as the closest

codeword to *X*. However, the closest codeword to *X* is in fact the codeword *H*. Moreover, in this example, calculating the path *B*-*E*-*K* is redundant.

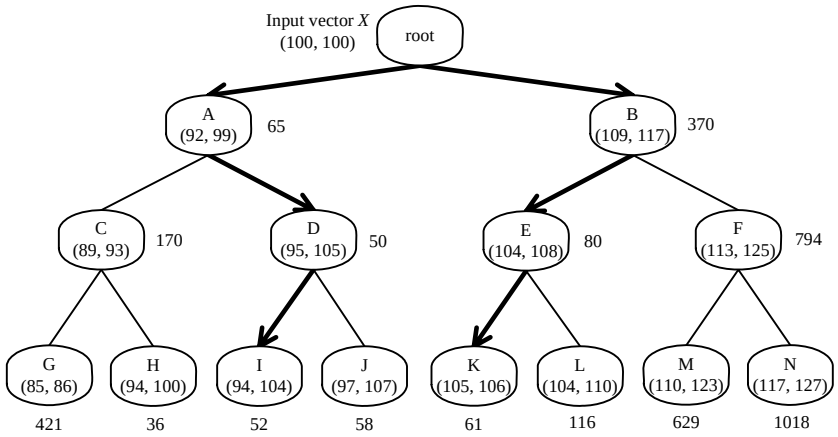

Figure 1. Example of two-path TSVQ

2.3 Dynamic Path TSVQ (DP-TSVQ)

Dynamic Path TSVQ (DP-TSVQ) is an adaptive multipath variation of TSVQ [6]. Unlike the multipath TSVQ algorithm, which fixes the number of the search paths, DP-TSVQ increases the number of search paths dynamically. DP-TSVQ defines a critical function to judge whether to increase the number of search paths. Let *CIN* indicate the current internal node, which DP-TSVQ is traversing. The critical function is defined as follows.

$$F(X, CIN) = \frac{\left| MSE(X, Lchild(CIN)) - MSE(X, Rchild(CIN)) \right|}{MSE(X, Lchild(CIN)) + MSE(X, Rchild(CIN))}, \quad \textbf{(3)}$$

where *X* is the input vector. *Lchild*(*CIN*) and *Rchild*(*CIN*) denote the left child node and right child node of *CIN*, respectively. *MSE*() is the mean squared error between the two vectors, as defined in Eq. (1).

The critical function has the following characteristic: when the value of *F*(*X*, *CIN*) is close to 1, the difference between *MSE*(*X*, *Lchild*(*CIN*)) and *MSE*(*X*, *Rchild*(*CIN*)) is very large; otherwise, these two children of *CIN* have a similar distance to *X*. Accordingly, DP-TSVQ employs the value of the critical function to determine dynamically whether to add paths for further search. The user assigns a value to the threshold *TH*. If $F(X, CIN) \le TH$, DP-TSVQ selects both children of *CIN* to search; otherwise, DP-TSVQ only traverses the nearest child node to *X*.

Example 2.2 Consider the same codebook tree as in Example 2.1, with input vector X = (100, 100). Assume that the threshold of the critical function is 0.7. Initially, DP-TSVQ takes a single search path from the root of the codebook tree. According to Fig. 2, the value of the critical function of the root, 0.677, is smaller than the threshold, that is, the difference between $MSE(X, A)$ and $MSE(X, B)$ is not too large. Therefore, DP-TSVQ adds the second search path, which involves B_node. Similarly, at the next level of the tree, the third path A-C is added to the search. Finally, the three search paths followed by DP-TSVQ are A-D-I, A-C-H and B-E-K. DP-TSVQ matches the closest codeword H to X.

DP-TSVQ performs fast in a low threshold situation, but produces a low image quality. In the above example, if TH = 0.5, $F(X, A)$ = 0.545 > TH. DP-TSVQ selects the codeword I rather than the codeword H as the closest codeword to X. When the value of the threshold equals one, the image quality is as good as that of FSVQ. However, DP-TSVQ does not limit the number of search paths. The running time of DP-TSVQ with TH = 1 is even longer than that of FSVQ.

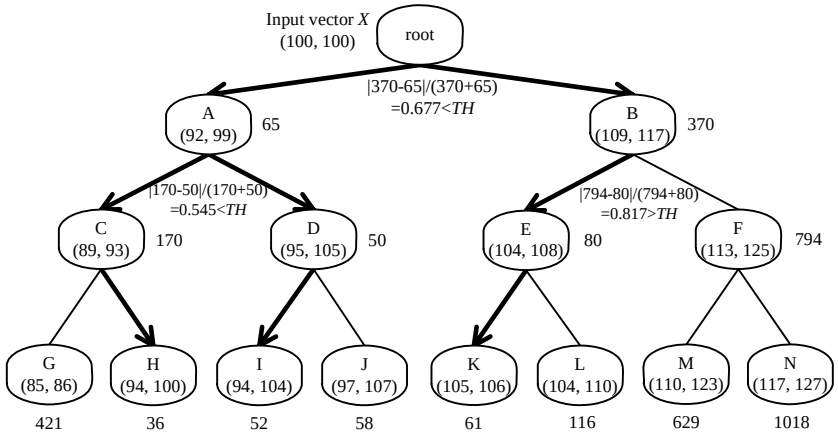

Figure 2. Example of DP-TSVQ with TH = 0.7

3 Full Search Equivalent TSVQ (FSE-TSVQ)

As described in the previous section, the searches performed by various TSVQ algorithms do not always result in the selection of the best codeword. This study develops a fast full search equivalent TSVQ algorithm (FSE-TSVQ) to obtain the global closest codeword.

Definition 1 Let *IN* be an internal node of the FSE-TSVQ codebook tree. The set of codewords $CB_{IN} = \{c_i | \ i = 1, 2, \ldots, p\}$, where c_i is a codeword in the subtree of *IN*. Let $R(IN)$ indicate the radius of *IN*. $R(IN) = \max_{c_i \in CB_{IN}} \{d(IN, c_i)\}$, where $d(IN, c_i)$ is the Euclidean distance defined in Eq. (2).

Each node of the FSE-TSVQ codebook tree is identical to that of the DP-TSVQ codebook tree, except that each internal node of the FSE-TSVQ codebook tree stores one additional value, the radius. FSE-TSVQ applies the triangle inequality to avoid searching useless paths, which could not improve the image quality. Therefore, FSE-TSVQ uses Euclidean distance rather than mean squared error to measure the distortion.

Definition 2 Let *X* be the input vector, *LC* be the local closest codeword and *GC* be the global closest codeword. Let *CB* be the codebook and CB_v be the set of codewords which FSE-TSVQ has visited, where $CB = \{c_i | \ i = 1, 2, \ldots, n\}$ and $CB_v = \{c_i | \ i = 1, 2, \ldots, q\}$. The distances of *GC* and *LC* to *X* are $d(X, GC) = \min_{c_i \in CB} \{d(X, c_i)\}$ and $d(X, LC) = \min_{c_i \in CB_v} \{d(X, c_i)\}$, respectively.

FSE-TSVQ employs single-path TSVQ to find the first local closest codeword *LC*. Let *CIN* represent the current internal node, which FSE-TSVQ is traversing, and let its two child nodes be *Lchild*(*CIN*) and *Rchild*(*CIN*). According to the triangle inequality, no codeword in the *Rchild*(*CIN*) subtree is closer to *X* than *LC* if

$d(X, LC) + R(Rchild(CIN)) < d(X, Rchild(CIN))$.

Figure 3 depicts the above inequality in two-dimensional space. If a codeword is closer to *X* than *LC*, the codeword must be found inside the dotted circle. In the

figure, no node in the right subtree of *CIN* is closer to *X* than *LC*. By Definition 2, the distance between *X* and *GC* is not greater than $d(X, LC)$ so that

$$d(X, GC) + R(Rchild(CIN)) < d(X, Rchild(CIN)).$$

Therefore, the path terminates in *Rchild*(*CIN*). Otherwise, FSE-TSVQ performs a depth-first traverse of the next level of *CIN*. The value $d(X, LC)$ is called the *bound* value. If FSE-TSVQ matches a codeword which is nearer to *X* than the previously found *LC*, FSE-TSVQ updates the bound value $d(X, LC)$ to perform further pruning of impossible paths.

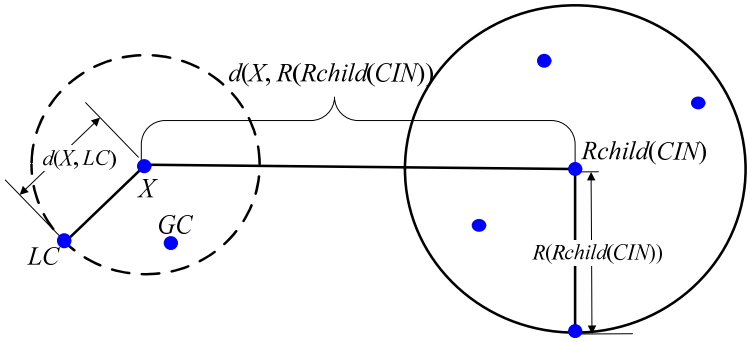

Figure 3. Two-dimensional representation of the inequality $d(X, LC) + R(Rchild(CIN)) < d(X, Rchild(CIN))$

A pseudo-code representation of the FSE-TSVQ algorithm is as follows.

Input: (1) A codebook tree of TSVQ with *n* codewords in the leaf level (each internal node storing an extra value of radius)

(2) A set of input vectors

Output: An encoded image

Procedure:

```
01 for each input vector X do {
02      Single-Path_TSVQ(X);  // obtain the initial LC and push internal nodes onto the stack
03      while top > -1 {                    // 0 ≤ top ≤ log2n - 1
04              CIN := stack[top];  top--; // pop off of the top of the stack
05              while (true) {
06                      D_left := d(X, Lchild(CIN));
07                      D_right := d(X, Rchild(CIN));
08                      if child of CIN is an internal node {
09                              dist[0] := R(Lchild(CIN)) + bound - D_left;
10                              dist[1] := R(Rchild(CIN)) + bound - D_right;
11                              if dist[0] ≤ 0 && dist[1] ≤ 0 {
12                                      break;   }
13                              elseif dist[0] ≤ 0 && dist[1] > 0 {
14                                      CIN := Rchild(CIN);      }
15                              elseif dist[0] > 0 && dist[1] ≤ 0 {
16                                      CIN := Lchild(CIN);      }
17                              elseif dist[0] > 0 && dist[1] > 0 {
18                                      if dist[0] ≥ dist[1] {
19                                              stack[++top] := Rchild(CIN);
20                                              CIN := Lchild(CIN);      }
21                                      else {
22                                              stack[++top] := Lchild(CIN);
23                                              CIN := Rchild(CIN);      }          }
24                              continue;          }
25                      if bound > min(D_left, D_right) {  // child of CIN is a leaf node
26                              bound := min(D_left, D_right);
27                              LC = D_left < D_right ? Lchild(CIN) : Rchild(CIN);              }
28                      break;
29              }
30      }
31 GC := LC;
32 return the index value of GC;
33 }
```

In Line 2, the function Single-Path_TSVQ(*X*) applies the single-path TSVQ algorithm to match a local closest codeword to *X*, which is the initial index value of *LC*. Then the function visits each branch of the path to determine whether the leaf nodes in the subtree of each branch could include *GC*. Single-Path_TSVQ(*X*) obtains the initial *LC*, computes the initial bound value, and pushes the nodes of branches onto the stack along with the path. Lines 11-23 employ the triangle inequality to

remove impossible paths. In Lines 11 and 12, unlike *m*-path TSVQ and DP-TSVQ, FSE-TSVQ could skip both the child nodes of *CIN*. Lines 25-26 update the bound value, and Line 27 updates *LC*. The pseudo-code of the function Single-Path_TSVQ(*X*) is as follows.

Function Single-Path_TSVQ(*X*)

```
01 CIN := root; top := -1; j := 0; // top is the current location of the top of the stack
02 while X is an internal node {
03        D_left := d(X, Lchild(CIN));
04        D_right := d(X, Rchild(CIN));
05        if (D_left < D_right) {
06                  stack[++top] := Rchild(CIN);
07                  CIN = Lchild(CIN);       }
08        else {
09                  stack[++top] := Lchild(CIN);
10                  CIN = Rchild(CIN);       }
11 }
12 if (D_left < D_right) {  bound := D_left;       }
13 else                    {  bound := D_right;     }
14 LC := CIN;
15 for i := 0 to top {  // top := log2n - 1
16        dist[0] = R(stack[i]) + bound - d(X, stack[i]);
17        if ( dist[0] > 0 ) {
18                  stack[j] := stack[i];
19                  j++;
20        }
21 }
22 top := j - 1;
```

Lines 5-10 push the branch nodes onto the stack along with the path. Lines 15-21 keep any branch nodes in the stack whose subtrees could include *GC*.

Example 3.1 Consider the same codebook tree as in Example 2.1, with input vector X = (100, 100). According to Fig. 4, except for the root, each node has a real number outside the node. Each of these values represents the Euclidean distance between the input vector X and the corresponding node. Table 1 lists the radius of each internal node. Initially, FSE-TSVQ employs a single search path to find the local closest codeword I and computes its Euclidean distance from X as the bound value ($\sqrt{52}$). FSE-TSVQ is a depth-first algorithm and the right child of *A*_node has been traversed. Since $\sqrt{52} + \sqrt{218} > \sqrt{170}$, FSE-TSVQ visits the second path, which includes *C*_node (the left child of *A*_node), until the leaf node is reached. FSE-TSVQ obtains the second local closest codeword H and updates the bound value as $\sqrt{36}$.

Then FSE-TSVQ determines whether the right subtree of the root needs to be traversed. Since $\sqrt{36}+\sqrt{164}<\sqrt{370}$, FSE-TSVQ skips the *B*_subtree and terminates.

FSE-TSVQ uses the pre-computed and pre-stored radii in the codebook tree to reduce the complexity of the closest codeword search. Moreover, the image quality of FSE-TSVQ equals that of FSVQ. The process of computing the radius of each internal node needs to execute only once in the codebook generation phase, in contrast with the encoding phase, where the FSE-TSVQ algorithm can repeat execution indefinitely. Therefore, the cost of computing radii can be disregarded.

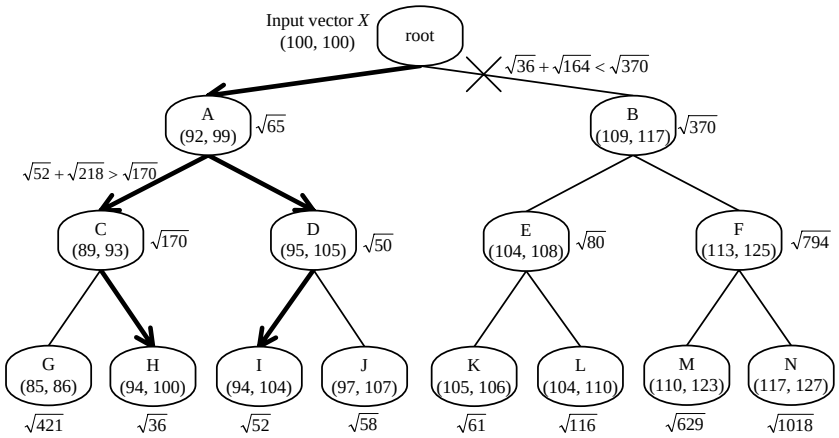

Figure 4. Example of FSE-TSVQ

Table 1. The radius of each internal node

Internal node	A	B	C	D	E	F
Radius	$\sqrt{218}$	$\sqrt{164}$	$\sqrt{74}$	$\sqrt{16}$	$\sqrt{5}$	$\sqrt{20}$

4 Enhanced Dynamic Path TSVQ (EDP-TSVQ)

Using a user-defined threshold, DP-TSVQ reduces the image quality to shorten the encoding time. It dynamically increases the number of search paths to match the closest codeword as nearly as possible. In environments of low computational capability, DP-TSVQ is very efficient. However, DP-TSVQ consumes even more time than FSVQ to encode a high-resolution image. Although FSE-TSVQ decreases the computational complexity of searching for the nearest neighbor codeword, it does not further accelerate the process in the case where the image quality is slightly degraded. A higher image quality requires a longer encoding time. This study therefore proposes an Enhanced DP-TSVQ (EDP-TSVQ) algorithm with a better tradeoff between encoding time and image quality than DP-TSVQ.

DP-TSVQ is fast if the threshold is low, and FSE-TSVQ always has the best image quality. The proposed EDP-TSVQ algorithm attempts to combine the advantages of DP-TSVQ and FSE-TSVQ. EDP-TSVQ can be viewed as adding the critical function of DP-TSVQ into FSE-TSVQ. First, like FSE-TSVQ, EDP-TSVQ employs the triangle inequality to prune several unattainable paths. Furthermore, EDP-TSVQ calculates the value of the critical function in each internal node, as does DP-TSVQ, to determine whether the distances of the two child nodes to *X* are similar. EDP-TSVQ combines the FSE-TSVQ and DP-TSVQ algorithms to produce a better time-quality tradeoff. EDP-TSVQ only needs to modify Lines 17-23 of the pseudo-code of FSE-TSVQ as follows.

```
01      elseif dist[0] > 0 && dist[1] > 0 {
02              fvalue := Critical(CIN);
03              if dist[0] ≥ dist[1] {
04                      if (fvalue ≤ TH ) {
05                              stack[++top] := Rchild(CIN);    }
06                      CIN := Lchild(CIN);     }
07              else {
08                      if (fvalue ≤ TH ) {
09                              stack[++top] := Lchild(CIN);    }
10                      CIN := Rchild(CIN);     }       }
```

Line 2 adds a critical function Critical() of DP-TSVQ. Lines 4-5 and 8-9 remove the paths whose critical function value is greater than the threshold. Similarly, EDP-

TSVQ also needs to modify Lines 15-21 of the pseudo-code of the function Single-Path_TSVQ(*X*) as follows.

```
01 for i := 0 to top {
02       fvalue := Critical(parent node of stack[i]);
03       dist[0] = R(stack[i]) + bound - d(X, stack[i]);
04       if ( dist[0] > 0 && fvalue ≤ TH ) {
05                stack[j] := stack[i];
06                j++;
07       }
08 }
```

According to the EDP-TSVQ algorithm, FSE-TSVQ is a special case of EDP-TSVQ. When $TH = 1$, the set of search paths traversed by EDP-TSVQ and FSE-TSVQ are identical.

Definition 3 Let V_{DP} and V_{EDP} denote the two node sets in which the nodes are visited by DP-TSVQ and EDP-TSVQ, respectively, where V_{DP} and V_{EDP} contain at least a leaf node.

Lemma 1 Give a codebook tree and the input vector *X*. If DP-TSVQ and EDP-TSVQ have an identical *TH*, then $V_{EDP} \subseteq V_{DP}$.

Proof. Let *CIN* indicate the currently visited node in the codebook tree. Assume that a child node N_c of *CIN* satisfies that $N_c \in V_{EDP}$ and $N_c \notin V_{DP}$. According to the EDP-TSVQ algorithm, since $N_c \in V_{EDP}$, $F(X, CIN) \le TH$ or N_c is closer to *X* than its sibling node. Let us consider the case where DP-TSVQ and EDP-TSVQ have an identical *TH*.

(1) According to the DP-TSVQ algorithm, while $F(X, CIN) \le TH$, $N_c \in V_{DP}$.

(2) If N_c is closer to *X* than its sibling node, the node must be traversed by DP-TSVQ. So $N_c \in V_{DP}$.

By the description of the two cases, we have $N_c \in V_{DP}$. This contradicts the original assumption. Therefore, if DP-TSVQ and EDP-TSVQ have an identical *TH*, then $V_{EDP} \subseteq V_{DP}$. Q.E.D.

Definition 4 Given an original image with size $m \times n$, the mean squared error between the original image and the encoded (reconstructed) image is defined as follows.

$$MSE = \frac{1}{m \times n} \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - y_{ij})^2 , \qquad (4)$$

where x_{ij} and y_{ij} denote the pixel values of the original image and the reconstructed image, respectively. The quality of the reconstructed image is measured by the peak signal-to-noise ratio (PSNR). The definition of PSNR is as follows.

$$PSNR = -10 \log_{10} \frac{MSE}{255^2}. \quad (5)$$

Definition 5 Give an input vector X. Let LC_{EDP} and GC_{EDP} be the local closest codeword and the global closest codeword of EDP-TSVQ, respectively, where $LC_{EDP} \in V_{EDP}$ and $GC_{EDP} \in V_{EDP}$. Let CB_{EDPc} be a set of codewords which EDP-TSVQ can visit, and let CB_{EDPh} be a set of codewords which EDP-TSVQ has visited. Here $CB_{EDPh} \subseteq CB_{EDPc}$, where $CB_{EDPc} = \{c_i | \ i = 1, 2, \ldots, p\}$, $d(X, GC_{EDP}) = \min_{c_i \in CB_{EDPc}} \{d(X, c_i)\}$, $CB_{EDPh} = \{c_i | \ i = 1, 2, \ldots, q\}$ and $d(X, LC_{EDP}) = \min_{c_i \in CB_{EDPh}} \{d(X, c_i)\}$.

Since $CB_{EDPh} \subseteq CB_{EDPc}$, we get $d(X, GC_{EDP}) \leq d(X, LC_{EDP})$.

Theorem 1 We are given a codebook tree and the input vector X. If DP-TSVQ and EDP-TSVQ have an identical TH, then DP-TSVQ and EDP-TSVQ have identical PSNR values (that is, the same image quality).

Proof. According to Lemma 1, owing to DP-TSVQ and EDP-TSVQ having an identical TH, we have $V_{EDP} \subseteq V_{DP}$. Let LN be a leaf node, where $LN \in (V_{DP} - V_{EDP})$. Let $Anc(LN)$ be an arbitrary ancestor node of LN. Assuming that LN is closer to X than GC_{EDP}, we get $d(X, LN) < d(X, GC_{EDP})$. During the codebook tree traversal of EDP-TSVQ, since $LN \notin V_{EDP}$, there exists an $Anc(LN)$ satisfies the inequality $R(Anc(LN)) + d(X, LC_{EDP}) < d(X, Anc(LN))$. Also, we have

$$\begin{aligned} d(X, LC_{EDP}) &< d(X, Anc(LN)) - R(Anc(LN)) \\ &\leq d(X, Anc(LN)) - d(Anc(LN), LN) \leq d(X, LN). \end{aligned} \quad (6)$$

By the assumption, we have $d(X, LN) < d(X, GC_{EDP})$ and $d(X, GC_{EDP}) \leq d(X, LC_{EDP})$. Then $d(X, LN) < d(X, LC_{EDP})$, which contradicts Eq. (6). So no leaf node in $(V_{DP} - V_{EDP})$ is closer to X than GC_{EDP}. EDP-TSVQ and DP-TSVQ select the identical codeword GC_{EDP} to treat as the closest codeword to X. Therefore, if DP-TSVQ and EDP-TSVQ have an identical TH, then DP-TSVQ and EDP-TSVQ have identical PSNR values.

Q.E.D.

5 Experimental Results

This simulation used a 550 MHz AMD Athlon PC with 512MB of main memory, running the Windows 2000 Professional operating system, to compare the performances of FSE-TSVQ, EDP-TSVQ, FSVQ and DP-TSVQ. All algorithms were coded in Visual C++ 6.0. Each experimental image involved 512×512 pixels with 256 gray levels. Four training images (Barbara, Boat, Lena, and Toys) were used to generate the codebook trees. Four test images (Airplane, Baboon, Girl, and Lena) were employed to test the search efficiency of our proposed algorithms. The Lena image was used in both the training set and the test set. These images were divided into various 4×4 blocks. Therefore, each block was represented by a 16-dimension input vector.

We used an array to implement the perfect binary codebook tree of DP-TSVQ. The set of codewords located on the leaf nodes of the codebook tree was exactly the same as the codebook of FSVQ. The codebook tree of FSE-TSVQ and EDP-TSVQ was identical except that an extra array was used to store the radius values of each internal node for FSE-TSVQ. This experiment considered four codebook sizes (256, 512, 104, and 2048).

Table 2 lists the image quality of four reconstructed images. The difference in PSNR between DP-TSVQ (or EDP-TSVQ) and FSVQ was used to evaluate the image quality. PSNR is defined in Eq. (5) and the difference is defined as follows.

$$Diff(P_d) = P_{FS} - P_d, \tag{7}$$

where P_{FS} represents the PSNR value of FSVQ, and P_d is the PSNR value of DP-TSVQ (or EDP-TSVQ). DP-TSVQ and EDP-TSVQ have identical values of $Diff(P_d)$ when they have the same threshold values. This result is in agreement with Theorem 1. In Table 2, the values of $Diff(P_d)$ are between 0.0085dB and 0.0005dB when TH = 0.3. The image quality of DP-TSVQ is close to that of FSVQ. If the threshold value TH is set to 0.6, the $Diff(P_d)$ values are not greater than 8×10^{-5}dB. Therefore, the encoding result of DP-TSVQ with TH = 0.6 is sufficient for use in an environment requiring high image quality. In the four test images, $Diff(P_d) = 0$ when $TH \geq 0.8$.

Table 2. The values of $Diff(P_d)$ and the PSNR values of FSVQ

Method		DP-TSVQ or EDP-TSVQ			
Image	*TH*	Codebook size			
		256	512	1024	2048
Airplane $Diff(P_d)$	0	0.66825	0.69377	0.72594	0.83914
	0.1	0.06992	0.06535	0.07414	0.10404
	0.2	0.01647	0.01153	0.01264	0.01551
	0.3	0.00404	0.00184	0.00226	0.00257
	0.4	0.00151	0.00006	0.00020	0.00066
	0.5	0.00026	0.00002	0.00004	0.00007
	0.6	0.00001	0	0	0
	0.7	0	0	0	0
FSE-TSVQ or FSVQ (PSNR)		29.80893	30.47185	31.12653	31.77125
Baboon $Diff(P_d)$	0	0.36229	0.40200	0.46716	0.55731
	0.1	0.03585	0.03435	0.04064	0.04901
	0.2	0.00489	0.00379	0.00387	0.00564
	0.3	0.00051	0.00063	0.00077	0.00096
	0.4	0.00014	0.00003	0.00012	0.00008
	0.5	0.00006	0.00001	0	0.00004
	0.6	0	0	0	0.00001
	0.7	0	0	0	0
FSE-TSVQ or FSVQ(PSNR)		23.68842	24.08075	24.44554	24.79215

Method		DP-TSVQ or EDP-TSVQ			
Image	*TH*	Codebook size			
		256	512	1024	2048
Girl $Diff(P_d)$	0	0.72359	0.80675	0.84170	0.84772
	0.1	0.14346	0.14269	0.15480	0.14668
	0.2	0.03778	0.03784	0.03200	0.02541
	0.3	0.00845	0.00706	0.00522	0.00552
	0.4	0.00106	0.00014	0.00111	0.00198
	0.5	0.00010	0.00001	0.0003	0.00014
	0.6	0	0	0	0.00010
	0.7	0	0	0	0
FSE-TSVQ or FSVQ(PSNR)		31.08119	31.78266	32.51071	33.14924
Lena $Diff(P_d)$	0	0.50468	0.53282	0.54427	0.57655
	0.1	0.08200	0.08210	0.08397	0.09038
	0.2	0.01857	0.01206	0.01505	0.01702
	0.3	0.00334	0.00219	0.00236	0.00313
	0.4	0.00106	0.00051	0.00024	0.00087
	0.5	0.00023	0.00016	0.00004	0.00011
	0.6	0.00008	0	0.00001	0
	0.7	0.00005	0	0	0
FSE-TSVQ or FSVQ(PSNR)		31.02359	31.91049	32.79162	33.58789

Table 3 presents the average number of nodes, which needs to be computed for an input vector, of the four methods using various threshold requirements and various codebook sizes. The number of nodes given in Table 3 includes the internal nodes and the leaf nodes. The last two rows of Table 3 list the computed number of nodes for FSE-TSVQ and FSVQ respectively. Except for $TH = 0$, the node computation requirement of EDP-TSVQ is always less than that of DP-TSVQ. This result is in agreement with Lemma 1. A higher threshold value implies a more significant difference in the node computation requirement between DP-TSVQ and EDP-TSVQ. For example, the number of nodes to be computed by EDP-TSVQ is 49-80% of the number computed by DP-TSVQ when $TH = 0.3$. The proportion falls to 24-49% when $TH = 0.6$. In Table 3, the darker color grids at the bottom denote that the encoded images have image qualities as high as those produced by FSVQ.

Table 3. Average computed number of nodes for an input vector including internal nodes and leaf nodes

Codebook size	256		512		1024		2048	
Method / Threshold	DP-TSVQ	EDP-TSVQ	DP-TSVQ	EDP-TSVQ	DP-TSVQ	EDP-TSVQ	DP-TSVQ	EDP-TSVQ
0	16.00	16.00	18.01	18.01	20.02	20.02	22.05	22.05
0.1	22.42	21.67	28.13	26.43	35.85	32.67	46.92	40.34
0.2	31.42	27.99	44.70	38.18	66.60	49.69	104.21	67.05
0.3	43.18	34.48	68.41	46.85	114.41	68.02	199.95	96.84
0.4	58.25	41.13	99.83	58.51	179.26	87.36	332.11	128.74
0.5	79.74	48.15	147.69	70.70	272.55	108.45	523.02	163.22
0.6	116.47	55.87	221.35	84.35	429.24	132.07	841.94	201.72
0.7	173.22	63.76	338.48	84.46	667.9	156.39	1325.03	241.09
0.8	237.21	71.23	469.87	94.35	934.68	179.25	1863.56	277.58
0.9	329.09	78.22	656.86	122.95	1312.26	201.19	2622.85	312.18
1	510.00	84.93	1022.00	134.18	2046.00	221.13	4094.00	344.54
FSE-TSVQ	84.93		134.18		221.13		344.54	
FSVQ	256.00		512.00		1024.00		2048.00	

Figures 5-8 depict the average running time for the four test images. The *y*-axes denote the average running time, and the *x*-axes denote various threshold values. Figure 5 shows that the running time of FSE-TSVQ is 38% of that of FSVQ with a codebook size of 256. The larger the codebook size, the more significant the differences in PSNR. In Fig 8, the running time of FSE-TSVQ decreases to 21% of that of FSVQ. When $TH \geq 0.5$, FSE-TSVQ is superior to DP-TSVQ in both running time and image quality. DP-TSVQ requires more running time than FSVQ when $TH > 0.7$. The performance of EDP-TSVQ and DP-TSVQ is almost the same when $TH \leq 0.2$. EDP-TSVQ is superior to DP-TSVQ when $TH > 0.2$. As depicted in Figs. 5-8, EDP-TSVQ offers a better tradeoff between time and quality than DP-TSVQ. When *TH* is set to 0.6 to obtain a high-quality reconstructed image with a codebook size of 256, the running time of EDP-TSVQ is only 51% of that of DP-TSVQ as represented in Fig. 5. Similarly, in Figs. 6-8, if $TH = 0.6$, the running time of EDP-TSVQ is only 37%, 32% and 25% of that of DP-TSVQ, respectively.

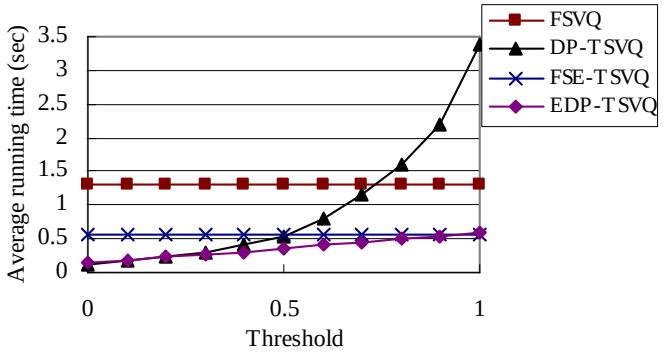

Figure 5. Average running time with 256 codebook size

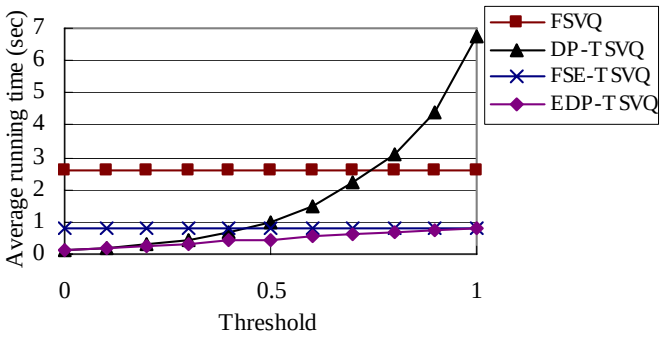

Figure 6. Average running time with 512 codebook size

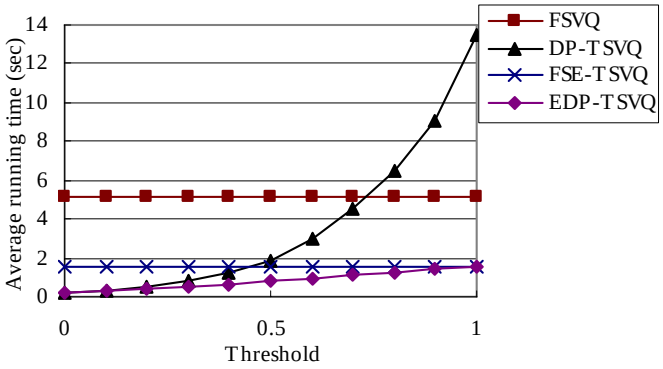

Figure 7. Average running time with 1024 codebook size

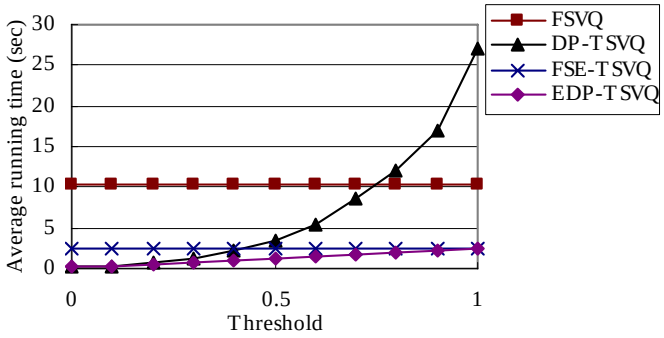

Figure 8. Average running time with 2048 codebook size

6 Conclusions

VQ is a widely used data compression technique. TSVQ plays an important role among the various VQ methodologies. The existing methods such as single-path TSVQ, multipath TSVQ and DP-TSVQ still have some drawbacks. Single-path TSVQ provides poor image quality, while multipath TSVQ and DP-TSVQ waste too much execution time attempting to improve the image quality to approach that of FSVQ. This research has developed a novel full search equivalent TSVQ (FSE-TSVQ), which stores a radius value for each internal node and employs the triangle inequality to filter out many impossible codewords, so that the search space is narrowed. FSE-TSVQ achieves image quality identical to that of FSVQ while requiring only 21-38% of the computation time of FSVQ.

Furthermore, for the environments of low computational capability or a slight reduction in image quality, this study proposes the hybrid method EDP-TSVQ, which combines FSE-TSVQ and DP-TSVQ to obtain a better tradeoff between time and quality than that of DP-TSVQ. In a low threshold requirement situation, the performance of EDP-TSVQ and DP-TSVQ is almost equivalent. EDP-TSVQ is always faster than DP-TSVQ when $TH > 0.2$. In addition, EDP-TSVQ and DP-TSVQ produce the same image quality. Therefore, the EDP-TSVQ algorithm offers a better time-quality tradeoff than DP-TSVQ.

References

[1] A. Buzo, A. H. Gray, R. M. Gray, and J. D. Markel, "Speech coding based upon vector quantization," *IEEE Trans. Acoustics, Speech, Signal Processing*, Vol. 28, No. 5, pp. 562-574, 1980.

[2] C. C. Chang and T. S. Chen, "New tree-structured vector quantization with closest-coupled multipath searching method," *Optical Engineering*, Vol. 36, No. 6, pp. 1713-1720, 1997.

[3] R. F. Chang, W. T. Chen, and J. S. Wang, "Image sequence coding using adaptive tree-structured vector quantization with multipath searching," *IEE Proc. I*, Vol. 139, No. 1, pp. 9-14, 1992.

[4] C. C. Chang, J. S. Chou, and T. S. Chen, "An efficient computation of Euclidean distances using approximated look-up table," *IEEE Trans. Circuits Systems Video Technology*, Vol. 10, No. 4, pp. 594-599, 2000.

[5] C. C. Chang and I. C. Lin, "Novel full-search for speeding up image coding using vector quantization," *Real-Time Imaging*, Vol. 10, No. 2, pp. 95-102, 2004.

[6] C. C. Chang and F. C. Shiue, "Tree structured vector quantization with dynamic path search," In *Proc. Intl. Workshop Multimedia Network Systems* (*MMNS*), *Aizu, Japan*, pp. 536-541, Sep. 1999.

[7] T. S. Chen and C. C. Chang, "Diagonal axes method (DAM): a fast search algorithm for vector quantization," *IEEE Trans. Circuits Systems Video Technique*, Vol. 7, No. 3, pp.555-559, 1997.

[8] A. Gersho and R. M. Gray, "Vector quantization and signal compression," Kluwer Academic Publishers, 1992.

[9] R. M. Gray and Y. Linde, "Vector quantizers and predictive quantizers for Gauss-Markov sources," *IEEE Trans. Communications*, Vol. 30, No. 2, pp. 381-389, 1982.

[10] C. M. Huang, Q. Bi, G. S. Stiles, and R. W. Harris, "Fast full search equivalent encoding algorithms for image compression using vector quantization," *IEEE Trans. Image Processing*, Vol. 1, No. 3, pp. 413-416, 1992.

[11] I. Katsavounidis, C. C. J. Kuo, and Z. Zhang, "Fast tree-structured nearest neighbor encoding for vector quantization," *IEEE Trans. Image Processing*, Vol. 5, No. 2, pp. 398-404, 1996.

[12] C. H. Lee and L. H. Chen, "A fast search algorithm for vector quantization using mean pyramids of codewords," *IEEE Trans. Communications*, Vol. 43, No. 2/3/4, pp. 1697-1702, 1995.

[13] Z. M. Lu, S. C. Chu, and K. C. Huang, "Equal-average equal-variance equal-norm nearest neighbor codeword search algorithm based on ordered hadamard transform," *Intl. Jour. Innovative Computing, Information and Control*, Vol. 1, No. 1, pp. 35-41, 2005.

[14] J. Mielikainen, "A novel full-search vector quantization algorithm based on the law of cosines," *IEEE Signal Processing Letters*, Vol. 9, No. 6, pp. 175-176, 2002.

[15] D. Mukherjee and S. K. Mitra, "Successive refinement lattice vector quantization," *IEEE Trans. Image Processing*, Vol. 11, No. 12, pp. 1337-1348, 2002.

[16] N. M. Nasrabadi and R. A. King, "Image coding using vector quantization: a review," *IEEE Trans. Communications*, Vol. 36, No. 8, pp. 957-971, 1988.

[17] Z. Pan, K. Kotani, and T. Ohmi, "An improved fast encoding algorithm for vector quantization using 2-pixel-merging sum pyramid data structure," *Pattern Recognition Letters*, Vol. 25, No. 4, pp. 459-468, 2004.

[18] J. S. Pan, Z. M. Lu, and S. H. Sun, "An efficient encoding algorithm for vector quantization based on subvector technique," *IEEE Trans. Image Processing*, Vol. 12, No. 3, pp. 265-270, 2003.

[19] S. A. Rizvi and N. M. Nasrabadi, "An efficient Euclidean distance computation for vector quantization using a truncated look-up table," *IEEE Trans. Circuits Systems Video Technology*, Vol. 5, No. 4, pp. 370-371, 1995.

[20] B. C. Song and J. B. Ra, "A fast search algorithm for vector quantization using L_2-norm pyramid of codewords," *IEEE Trans. Image Processing*, Vol. 11, No. 1, pp. 10-15, 2002.