Algorithms for Intervector Dependency Utilization in Variable Rate Tree-Structured Vector Quantization*

Ligang Lu† and **William A. Pearlman**

Electrical, Computer and System Enginerring Department
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA
e-mail: pearlman@rpi.edu, luli@indy.tce.com

ABSTRACT

In this paper we improve and extend our previous results in *finite- state variable rate tree-structured vector quantization (FSVRTSVQ)*[1] by developing two new algorithms to increase its coding efficiency. The two new algorithms are derived to utilize the inter-codeword dependency and the tree structure property of the codebook so as to achieve bit rate reduction. The evaluation of both algorithms on various synthesis and real sources has shown that as many as 32.3% of bit rate savings has been obtained over the pruned tree-structured vector quantization (PTSVQ)[3].

Keywords: source compression, vector quantization, tree-structured vector quantization, finite state vector quantization

1 INTRODUCTION

In vector quantization (VQ) of sources with memory, such as video and audio signals, there will exist correlation in the quantizer outputs. On the other hand, the structural design of the tree-structured codebooks determines that highly correlated source vectors will be encoded either by the same codevector or by the codevectors within a small subtree. It implies that if two source vectors are highly correlated, one can expect that a great portion of their path maps will be the same. Therefore, we may utilize the codevector correlation to improve the coding efficiency of variable rate tree-structured vector quantization (VRTSVQ).

[1]This paper is based on Chapter 4 of Ligang Lu's Ph.D. thesis. The work was performed in the Center for Image Processing Research at Rensselaer Polytechnic Institute. This material is based upon work supported by the National Science Foundation under Grant No. NCR-9004758 and ARPA under Contract No. F19628-91-K–0031. The Government has certain rights in this material.

[2]Ligang Lu is now with Coporate Research of Thomson Consumer Electronics, Indianapolis, IN.

In this paper, we will improve and extend our previous results[1] by developing two new algorithms to exploit the inter-codeword correlation in VRTSVQ. Our objective for this work is to investigate simple yet effective algorithms to improve the coding efficiency of VRTSVQ by making use of the inter-codeword dependency. Specifically, we will apply the concepts of the finite state VQ (FSVQ) into VRTSVQ to develop a new coding technique called *finite-state variable rate tree-structured vector quantization* (FSVRTSVQ) which may effectively exploit the inter-codevector correlations. In the following sections, we will start with the introduction of the concepts of the FSVQ and then develop FSVRTSVQ by incorporating FSVQ into VRTSVQ. We will present two algorithms of FSVRTSVQ to utilize the inter-codeword memory. Finally, we will analyze the performance of FSVRTSVQ and examine the results on various sources.

2 PRELIMINARY

In this section we introduce the general concepts and properties of the finite state VQ (FSVQ) for the development that follows.

A finite state vector quantizer[4] has the property that the encoder and decoder used for the current input vector depend on the previous input. That is, a finite state vector quantizer is a vector quantizer with memory. In the finite state vector quantization (FSVQ), there is a set of subcodebooks called *the state codebooks*. Each state codebook corresponds to a particular state of the quantizer, i.e., a particular structure of the quantizer. The union of these state codebooks is *the super codebook*,

$$C = \bigcup_i C_i. \tag{1}$$

Given an input sequence of vectors $X_k, k = 0, 1, \ldots$, the encoder produces both a sequence of channel indices $U_k, k = 0, 1, \ldots$, and a sequence of *states* $S_k, k = 0, 1, \ldots$, where $U_k \in \mathcal{U}$, *the channel alphabet*, and S_k takes a value from a finite set $\mathcal{S}$, *the state set*. Each pair of the previous U_{k-1} and S_{k-1} specifies a particular state of the quantizer which, in turn, determines a subcodebook for the coding of the current input vector X_k. More specifically, the current state S_k of the quantizer is decided by the channel index U_{k-1} of previous coded vector and the previous state S_{k-1},

$$S_k = f(U_{k-1}, S_{k-1}); \tag{2}$$

where f is some mapping called *the next-state transition function*. The encoding of the current input vector X_k is a mapping which depends upon the state S_k by searching the best matching codevector within the subcodebook C_k specified by S_k,

$$U_k = h(X_k, S_k). \tag{3}$$

The channel index is then transmitted to the receiver over the channel. The decoding is another mapping which also uses the current state S_k to reconstruct the vector

$$\hat{X}_k = g(U_k, S_k). \tag{4}$$

The encoding operation of the finite state vector quantizer also obeys the minimum distortion rule,

$$d(X_k, g(U_k, S_k)) = \min_{\hat{X}_k \in C_k} d(X_k, \hat{X}_k). \tag{5}$$

Obviously, the conventional memoryless VQ can be viewed as a special case of a FSVQ which has only one state.

3 FINITE STATE VARIABLE RATE TREE-STRUCTURED VECTOR QUANTIZATION

In this section, we develop a new technique to exploit inter-codeword correlation in VRTSVQ by applying the concepts of FSVQ in VRTSVQ. We therefore call this technique *finite state variable rate tree-structured vector quantization* (FSVRTSVQ).

First, to see how the correlation between vectors may be exploited, let us consider a codebook C of a variable rate tree-structured vector quantizer and a real vector source $\mathcal{X}$. Let the vector dimension be K. In the rest of this chapter, we shall call C the super tree codebook and call the codebooks formed by its various branches the subtree codebooks. The super tree codebook C corresponds to a tree T with a finite set of nodes. Associated to each node t_i, there is a codevector v_i. The tree structure of the codebook C is designed to have the successive refining property. To illustrate this property, let us use the partition concept. The root node t_0 of the tree T corresponds to the K dimensional Euclidean space $\mathcal{R}^K$. Each node t of T determines a partition cell which, in turn, corresponds to particular subspace $\mathcal{R}_t^K$ of $\mathcal{R}^K$. For every non-leaf node t, its child node set $C(t)$ represents a set of partition cells that refines the partition cell determined by t. In another word, the subspace $\mathcal{R}_t^K$ corresponding to t is split into a set of disjoint subspaces $\mathcal{R}_k^K$, $k \in C(t)$ with

$$\mathcal{R}_t^K = \cup_{k \in C(t)} \mathcal{R}_k^K. \tag{6}$$

The set of all leaf nodes $\tilde{T}$ represents the finest partition achieved by T,

$$\mathcal{R}^K = \cup_{l \in \tilde{T}} \mathcal{R}_l^K. \tag{7}$$

The encoding of a source vector using C is a successive mapping process in which the source vector is assigned to the partition cells along a path of the tree according to some distortion measure until the source vector is mapped to a partition cell represented by a leaf node. The path from the root to that leaf reflects the successive mapping results and is therefore called *the complete path map.* The complete path map is transmitted to the decoder and the codevector stored at that leaf node is used to reconstruct the source vector. Therefore, if two source vectors X and Y are highly correlated and suppose that Y is

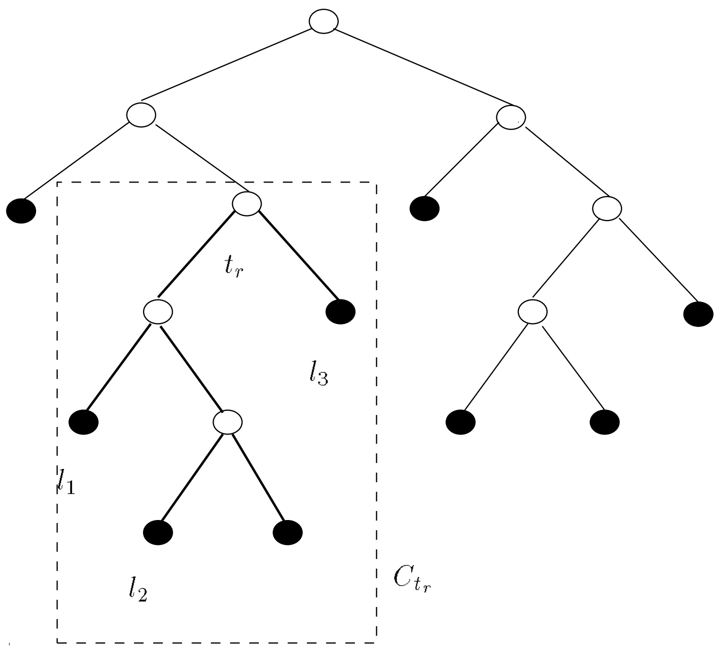

Figure 1: An example of subtree codebook.

encoded by a codevector v_i stored at some leaf node l_i, then X is highly likely to be encoded either by the same codevector v_i or by a codevector v_j stored at another leaf node l_j, where l_i and l_j are within a small subtree as shown in Figure 1. If so, the two path maps will have a large portion of same prefix. This implies that, to encode X, we may not need to search the whole super tree codebook C and transmit the complete path map (or channel index). Instead, we only need to search a small subtree codebook corresponding to a branch of the super tree around the leaf node l_i at which the codevector v_i is stored and transmit the partial path map corresponding to the codevector used for encoding X in this subtree codebook. Hence we can achieve a bit rate saving by exploiting the correlation between X and Y.

However, in the coding process, the current input X_k may not necessarily be highly correlated with its predecessor X_{k-1}. In another word, the best codevector in C for encoding X_k may not be within the small subtree around v_i which encoded X_{k-1}. So the key problem now becomes, given the information of the codevector which encoded the previous vector X_{k-1}, how to judiciously determine or predict an appropriate subtree codebook C_k from the super tree codebook C to encode X_k. The encoding of the current input vector X_k using this subtree codebook should achieve a bit rate saving in comparison to encoding X_k using the super tree codebook while maintaining similar coding fidelity.

In the following sections, we will develop two algorithms to utilize the inter-codeword correlation in adjacent source vectors by incorporating the concepts of FSVQ into VRTSVQ.

3.1 Algorithm 1

Let the tree T correspond to the super tree codebook C. To each node t_i of the tree T, we assign a state S_{t_i} which uniquely specifies a subtree codebook C_{t_i}. This subtree codebook C_{t_i} corresponds to the branch of T rooted at the node t_i. Moreover, if T has L leaf nodes, then for convenience, we can label the leaf nodes of T by an integer set $\mathcal{I} = (1, 2, \ldots, L)$.

To ensure that the decoder can correctly trace the state of the encoder without side information, the prediction of the current subtree codebook can not depend on the current input, that is, the next-state transition function f can only depend on the previous coded source vector, i.e., the codevector that encoded the previous input or equivalently its path map (channel index), and the previous state. If the statistics of the source are known or are well represented by the training set, one reasonable approach to determine the current subtree codebook is to use the information of the conditional probabilities of the codevectors. Let $\mathcal{P}(v_i|v_j)$ be the probability that the current input will be encoded by the codevector v_i given that the previous input is coded by v_j, where $i, j = 1, 2, \ldots, L$, and L is the leaf number of the tree T. Then the next-state transition function f and the next state S_k can be defined as follows.

Given that the previous codevector is v_j, the next state transition function first examines the conditional probabilities $\mathcal{P}(v_1|v_j), \mathcal{P}(v_2|v_j), \ldots, \mathcal{P}(v_L|v_j)$. Suppose that m_j of them are greater than zero, clearly, $0 \leq m_j \leq L$. If $0 < m_j \leq m$, where m is a pre-selected integer, let $v_{i_1}, v_{i_2}, \ldots, v_{i_{m_j}}$ be the m_j codevectors corresponding to the m_j non-zero conditional probabilities $\mathcal{P}(v_{i_1}|v_j)$, $\mathcal{P}(v_{i_1}|v_j)$,. . ., $\mathcal{P}(v_{i_{m_j}}|v_j)$, respectively. Then starting from the m_j leaf nodes, $l_{i_1}, l_{i_2}, \ldots, l_{i_{m_j}}$, at which v_{i_1}, v_{i_2}, . . ., $v_{i_{m_j}}$ are stored, trace back to their parent nodes, until reaching the first node t_r which is the common ancestor of them; then set the next state

$$S_k = S_{t_r}. \tag{8}$$

The state codebook C_k specified by S_k is the subtree codebook C_{t_r} corresponding to the subtree rooted at the node t_r. This subtree has the property that it is the smallest branch of T that includes $l_{i_1}, l_{i_2}, \ldots, l_{i_{m_j}}$ as a part of its leaf nodes. Since for the trees we have defined, the root node t_0 is the common ancestor of all leaf nodes, in the worst case, the subtree codebook C_k is equal to the supertree codebook C.

If $m_j > m$ or $m_j = 0$, i.e., if there are more than m possible candidates for encoding the current input X_k or if there is no appropriate prediction of the codevector for encoding X_k, then simply set the next state

$$S_k = S_{t_0}, \tag{9}$$

where t_0 is the root of the super tree T, i.e., the next state codebook or the subtree codebook C_k is the super tree codebook C. X_k is encoded using C_k, i.e., the best matching codevector in C_k is sequentially searched and either the corresponding partial path map or the complete path map is transmitted to the receiver. Since the current state S_k and the corresponding subtree codebook C_k are determined depending only on the previous codevector and the conditional probabilities, the decoder can trace the encoder's state without side information provided that the decoder also has the information of the codevector conditional probabilities.

As in most, if not all, cases of VQ designs and applications, we have a training set assumed to represent the source statistics. The conditional probability of each codevector can be estimated by its relative frequency of occurrence given its predecessor during the VRTSVQ codebook design procedure. Suppose X_{k-1} is encoded by v_j, i.e.,

$$\hat{X}_{k-1} = v_j, \tag{10}$$

and let $N_{i|j}$ denote the number of pairs such that $\hat{X}_{k-1} = v_j$ and $\hat{X}_k = v_i$; let N_j be the total number of source vectors mapped to v_j, then the estimate of the conditional probability $\mathcal{P}(v_i|v_j)$ is

$$\mathcal{P}(v_i|v_j) = \frac{N_{i|j}}{N_j}. \tag{11}$$

The design algorithm of FSVRTSVQ can be summarized as follows.

FSVRTSVQ Design Algorithm 1

- **Step 1.** (Design the supertree codebook)
  Given a training vector set which represents the statistic characteristics of the source, design a variable rate tree-structured codebook; label each leaf node with an integer j, $j = 0, 1, \ldots, L$.

- **Step 2.** (Estimate conditional probability)
  Obtain the estimates of the conditional probabilities $\mathcal{P}(v_i|v_j), i, j = 0, 1, \ldots, L$ of each codevector associated with leaf nodes.

- **Step 3.**(Establish a look-up table)
  For each reproduction codevector v_j, find m_j, the number of all non-zero conditional probabilities and the corresponding codevectors. Store the m_j integer labels of the leaf nodes corresponding to the m_j codevectors in a table.

- **Step 4.** (Implement the next state transition function)
  Implement the next state transition function as a table look-up procedure and a node common ancestor search procedure.

To save the memory required for the storage of the table, we can calculate the subtree for each codevector before hand and store only the root node of each subtree instead. Let $\bar{m}$ be the average number of $m_j, j = 1, 2, \ldots, L$, then the storage requirement can be reduced from $\bar{m}L$ to L integer numbers. The disadvantage for only storing the root node of each subtree is that if m is changed we have to recalculate the subtrees and store the new root nodes. But even in the former case, the memory increase for storing the integer table for a codebook with practical size can be managed small through the choice of m.

If the estimation of the conditional probabilities is sufficiently accurate, the performance of this algorithm can be guaranteed no worse than the original VRTSVQ. Otherwise, the quantizer might be unable to track the input. In the following, we present another algorithm as an alternative.

3.2 Algorithm 2

Actually, the best prediction of the subtree codebook for current input X_k should not only depend on the information of the previous coded input, but also based upon the knowledge of X_k itself. Although, utilizing the information of the current input will require additional bits to transmit the side information, the overall gain may be much more than the extra cost. In the following, we propose an algorithm wherein the next state transition function exploits both the information of the previous coded input and the knowledge of the current input to determine the current state of the quantizer.

Since, in Algorithm 1, the m_j codevectors with the positive conditional probabilities are the likely candidates to encode the current input, intuitively the best codevector to represent the current input is the one that has the minimum distortion with X_k. Hence, Algorithm 1 can be modified as follows.

Given that the previous vector X_{k-1} is encoded by the codevector v_j, the encoder examines the conditional probabilities $\mathcal{P}(v_1|v_j), \mathcal{P}(v_2|v_j), \ldots, \mathcal{P}(v_L|v_j)$. Suppose m_j terms of them are greater than zero, $0 \leq m_j \leq L$. We design the next state transition function f and determine the current state S_k as follows,

If $m_j > m$ or $m_j = 0$, then as in Algorithm 1, set the current state as

$$S_k = S_{t_0}, \tag{12}$$

where t_0 is the root of T. The corresponding state codebook is the super tree codebook,

$$C_k = C. \tag{13}$$

That is when there are more than m possible codevector candidates for encoding X_k or when no appropriate prediction of the codevector for encoding X_k, the next state transition function specifies the super tree codebook to encode the current input. Otherwise, if $0 < m_j \leq m$, denote the m_j codevectors $v_{i_1}, v_{i_2}, \ldots, v_{i_{m_j}}$, corresponding to the m_j non-zero conditional probabilities $\mathcal{P}(v_{i_1}|v_j), \mathcal{P}(v_{i_2}|v_j), \ldots, \mathcal{P}(v_{i_{m_j}}|v_j)$, respectively. The next state transition function chooses the codevector v_{i^*} that has the minimum distortion with the current input X_k from the m_j codevectors which are the likely candidates to encode the current inputs X_k . Then the leaf node l_{i^*} at which v_{i^*} is stored and the corresponding state $S_{l_{i^*}}$ can be determined. That is

$$v_{i^*} = \arg \min_{j \in \{i_1, i_2, \ldots, i_{m_j}\}} d(X_k, v_j) \tag{14}$$

and

$$S_k = S_{l_{i^*}}. \tag{15}$$

Once S_k is determined, the corresponding subtree codebook C_k can be uniquely determined. Clearly

$$C_k = C_{l_{i*}}. \tag{16}$$

Furthermore, $C_{l_{i*}}$ has only one codevector, v_{i^*}, so there is no need to transmit any partial path map for v_{i^*}. Instead, we need to transmit $\lceil \log_2 m_j \rceil$ bits side information to specify the codevector v_{i^*} among the m_j probable codevectors to encode the current vector. This side information also enables the decoder to uniquely determine the current state S_k so as to trace the state of the encoder. Since knowing the previous codevector v_j, the decoder can also find the number m_j by examining the pre-stored table. If $0 < m_j \leq m$, the decoder expects only $\lceil \log_2 m_j \rceil$ bits of side information, otherwise it looks forward to a complete path map from the transmitter. Therefore X_k can be correctly reconstructed either by v_{i^*} specified by the $\lceil \log_2 m_j \rceil$ bits side information or by a codevector represented by a path map, and both the encoder and the decoder can be kept in the same track.

The design procedure for Algorithm 2 is similar to that of Algorithm 1 summarized in Section 3.1 except for Step 4. Now, the next state transition function is implemented as a table look-up procedure and a minimum distortion codevector search procedure. For the sake of clarity, we enumerate the steps of Algorithm 2 in the following.

FSVRTSVQ Design Algorithm 2

- **Step 1.** (Desgin the supertree codebook)
  Design a variable rate tree-structured codebook from an appropriate training source which sufficiently represents the statistic characteristics of the source to be compressed. Label the leaf nodes of the tree by an integer set $\mathcal{I} = \{0, 1, \ldots, L\}$.

- **Step 2.** (Estimate conditional probability)
  For every reproduction codevector v_j at the leaf node j, $j = 0, 1, \ldots, L$, estimate the conditional probabilities $\mathcal{P}(v_i | v_j)$, $i = 0, 1, \ldots, L$.

- **Step 3.** (Establish a look-up table)
  For each reproduction codevector v_j, $j = 0, 1, \ldots, L$, find m_j, the number of all positive conditional probabilities and the associated reproduction codevectors. Store the m_j integer labels of the leaf nodes corresponding to the m_j reproduction codevectors in a table.

- **Step 4.** (Implement the next state transition function)
  Implement the next state transition function as a table look-up procedure and a minimum distortion codevector search procedure.

This algorithm predicts a subtree codebook to encode the current input X_k based upon the information not only of the past coded input, but also of the present input X_k. Since the decoder has no information about X_k, the side information becomes a necessity. However, if the $\lceil \log_2 m_j \rceil$ bits is less than the complete path map L_{i^*} from the root node to the leaf node where v_{i^*} is stored, we have a bit rate saving of $L_{i^*} - \lceil \log_2 m_j \rceil$. Therefore, for sources having high correlation, we can expect that Algorithm 2 may achieve significant performance gain over the conventional VRTSVQ. Furthermore we can also expect that Algorithm 2 will have more efficient performance than Algorithm 1. This is not only because Algorithm 2 utilizes the current input information but also because it more efficiently exploits the previous information. For example, suppose that $m_j = 8 < m$, then Algorithm 2 only needs to transmit 3 bits side information. However, Algorithm 1 needs to track back to the common ancestor of the 8 corresponding leaf nodes. For the binary tree case, it will require to track back at least 3 steps to find a common ancestor for 8 leaf nodes. Consequently the resulting partial path map will, in the most cases, be longer than 3 bits. Therefore, Algorithm 2 can be expected to have better coding efficiency than Algorithm 1.

4 SIMULATION RESULTS

In this section, we examine the performance of FSVRTSVQ on various sources. We first evaluate the performance of FSVRTSVQ on some well known synthetic sources. The first source tested is an AR(1) source with the correlation coefficient 0.9. In the test, the codebooks of VRTSVQ at various rates were designed from a training set of 100,000 vectors with size of 8, generated by the AR(1) source. The generalized BFOS pruning algorithm[3] was applied to the initial trees to obtain the best subtrees from the initial tree. Both algorithms of FSVRTSVQ were implemented and applied on these tree-structured codebooks to encode another test source consisting of 30,000 vectors outside of the training set. The results are presented in Figure 2. For comparison, we have also plotted the performance of the pruned VRTSVQ on the test source. The results have shown that, although at low rates, where the trees are relatively small, both algorithms of FSVRTSVQ have no obvious advantage over VRTSVQ, Algorithm 2 does achieve substantial gain over VRTSVQ at medium rates. It has achieved as much as 15.2% bit rate savings, or equivalently more than 1 dB gain, over the pruned VRTSVQ.

The second synthetic source we used to evaluate FSVRTSVQ is an AR(2) source with $a_1 = 1.515$ and $a_2 = -0.752$. Again pruned VRTSVQ codebooks at various rates were obtained from a training set having 100,000 vectors generated from the AR(2) source. The two FSVRTSVQ algorithms were used in coding a test source comprising 30,000 vectors drawn separately from the AR(2) source. The results are shown in Figure 3.

As on the AR(1) source, the results of FSVRTSVQ on the AR(2) source have shown the same conclusion. While Algorithm 1 has only gained a little over the pruned VRTSVQ at the medium rates, Algorithm 2 has obtained significant bit rate savings.

The second set of tests were conducted on two real sources. The first test source consists of 27,960 vectors with vector dimension of 16. The samples were taken from motion compensated frame difference images of the sequence *Salesman*. The variable rate tree-structured codebooks of various rates were designed by the training source generated from motion compensated frame difference images of the sequence *Miss America*. We applied both VRTSVQ and FSVRTSVQ to encode the test source using these code-

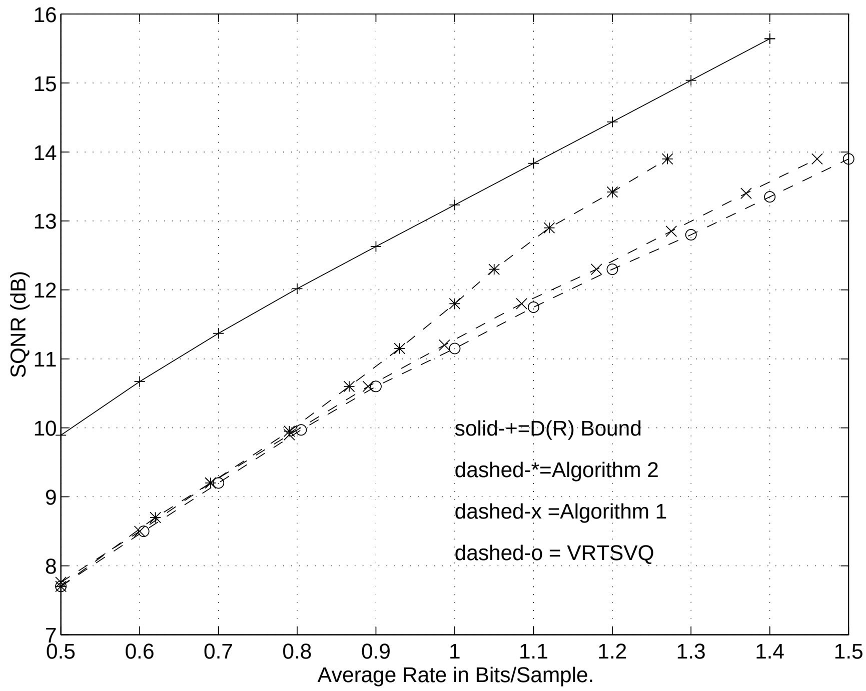

Figure 2: The Performance of FSVRTSVQ on an AR(1) Source with $a = 0.9$ and Vector Size 8.

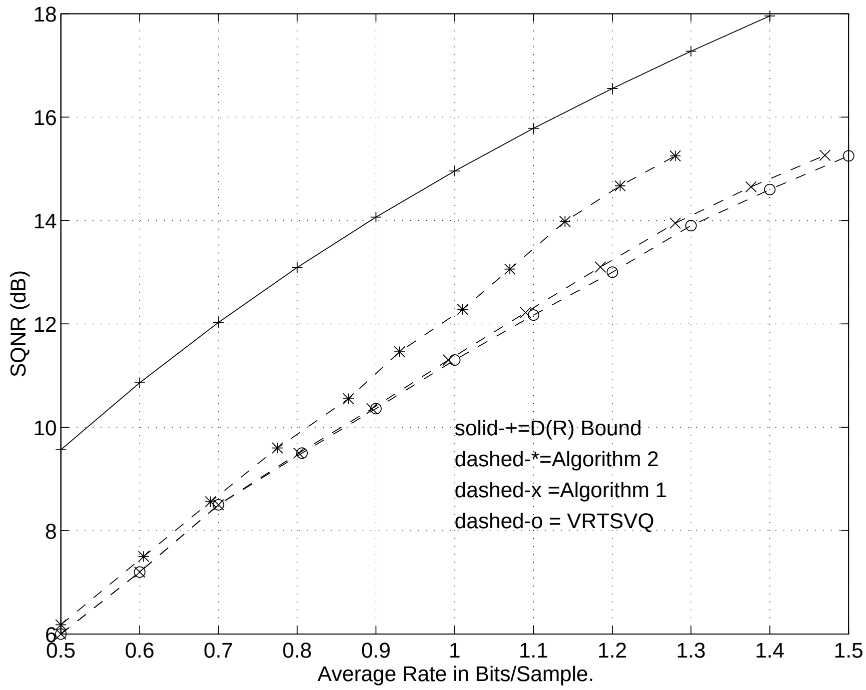

Figure 3: The Performance of FSVRTSVQ on an AR(2) Source with $a_1 = 1.515$ and $a_2 = -0.752$ and Vector Size 8.

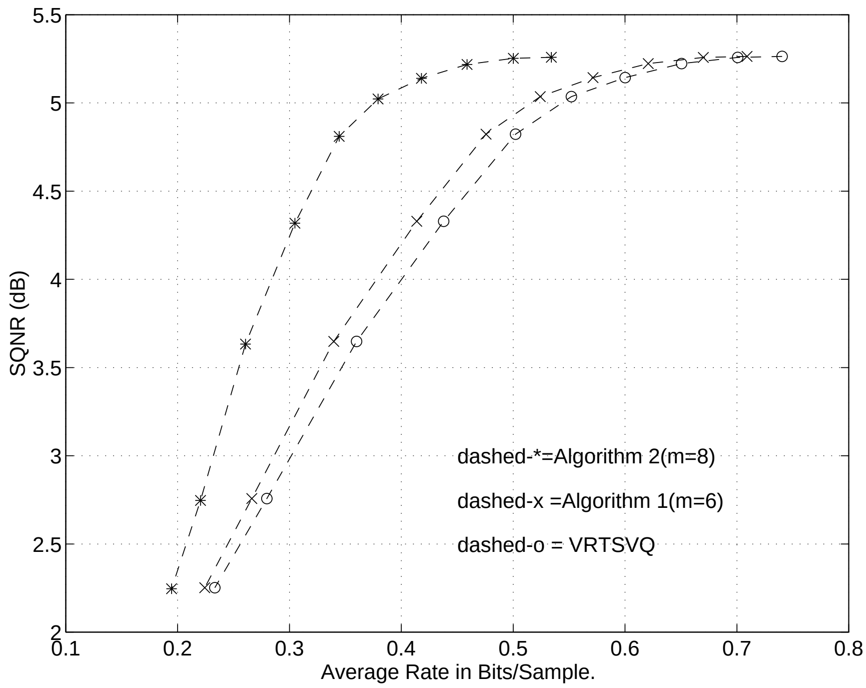

Figure 4: The Performance of FSVRTSVQ on a Real Source and Vector Size is 16.

books. In the simulation, Algorithm 1 and Algorithm 2 of FSVRTSVQ were used with $m = 6$ and $m = 8$, respectively. For comparison, the results are plotted in Figure 4. From the results, we can see that both algorithms uniformly outperform VRTSVQ at all rates while Algorithm 2 again has the best performance. Algorithm 2 has achieved a bit rate savings from 16.6% to 27.9%.

The second real test source is a motion compensated frame difference image generated from the sequence *Miss America*. The variable rate tree-structured codebooks were designed by a training source obtained from the motion compensated frame difference images generated from the sequence *Salesman*. Both algorithms of FSVRTSVQ and VRTSVQ were applied to encode the test source using these tree-structured codebooks. The results are plotted in Figure 5. Again from the results we can conclude that both algorithms outperform VRTSVQ at all rates. The performances of Algorithm 1 are obtained with $m = 4$. For this testing source, the bit rate savings of Algorithm 1 ranges from 6.3% to 10.1%. Algorithm 2 with $m = 8$ has the best performance at all rates, achieving a bit rate reduction from 19.8% to 32.3%.

5 CONCLUSIONS

In this paper, we have developed a new technique called FSVRTSVQ to exploit inter-codeword correlation. We have proposed two simple algorithms of FSVRTSVQ. One advantage of FSVRTSVQ is that, by utilizing the tree structure, the subtree codebooks in FSVRTSVQ are naturally embedded in the supertree codebook. Therefore there is no need to design the subtree codebooks separately and the increase of the storage requirement is also small.

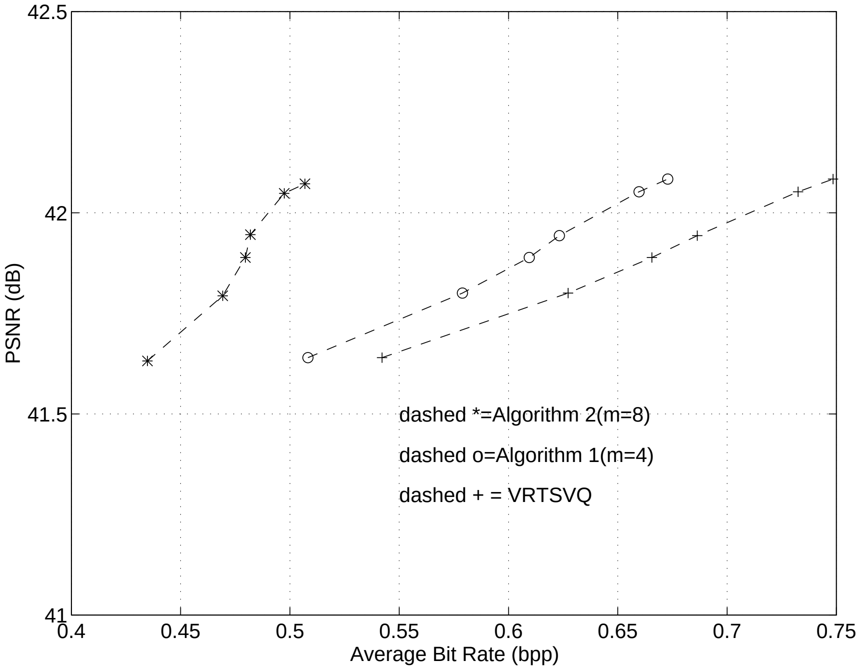

Figure 5: The Performance of FSVRTSVQ on a Motion Compensated Frame Difference Image of the Sequence *Miss America*, The Vector Size is 16.

The results of FSVRTSVQ on all sources tested have consistently shown that Algorithm 2 performed very effectively, especially on the real sources generated from the motion compensated frame difference images of video sequences. As is true in the general situations, if the training set indeed represents the statistical characteristics of the source, FSVRTSVQ can be expected to significantly outperform VRTSVQ. In addition, for sources having high intervector correlation, such as motion compensated frame difference images, and for relative large tree-structured codebooks, FSVRTSVQ could achieve substantial bit rate savings while maintaining the coding fidelity.

6 REFERENCES

1. Ligang Lu and William A. Pearlman " Video Coding Using Finite State Variable Length Tree-structured Vector Quantization," *Proc. of Information Science and Systems*, Baltimore, March 24-26, 1993.

2. Ligang Lu, "Advances in Tree-structured Vector Quantization and Adaptive Video Coding," Ph.D. thesis, Rensselaer Polytechnic Institute, August, 1995.

3. P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with application to tree-structured source coding and modeling," *IEEE Trans. on Info. Theory,* Vol. 35, No. 2, pp. 299-315, 1989.

4. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.