A FAST FULL-SEARCH ADAPTIVE VECTOR QUANTIZER FOR VIDEO CODING

Scott E. Budge

Electrical and Computer Engineering Dept.
Utah State University
Logan, UT 84322-4120
scott.budge@ece.usu.edu

Christian B. Peel

Electrical and Computer Engineering Dept.
Brigham Young University,
Provo, UT 84602
chris.peel@ieee.org

ABSTRACT

This paper presents a novel VQ structure which provides very good quality encoding for video sequences and exploits the computational savings gained from a fast-search algorithm. It uses an adaptive-search, variable-length encoding method which allows for very fast matching of a wide range of transmission rates. Both the encoding quality and the computational benefits from the fast-search algorithm are presented. Simulations show that full-search tree residual VQ (FTRVQ) can provide up to 3 dB improvement over a similar RVQ encoder on video sequences.

1. INTRODUCTION

The use of digital video is increasing on the Internet and in other multimedia environments. A significant benefit of using vector quantization for digital video coding is that decoding is a simple table-lookup. This asymmetry of encoding and decoding is particularly applicable to transmission of video over the Internet. In addition, VQ is the theoretically optimal method for quantizing and compressing images [1]. Despite these benefits, to approach optimality a VQ encoder requires large vector sizes and codebooks. This requires large memory and computational resources. To overcome these problems, product code techniques such as tree-structured VQ and residual VQ are often used [2].

Two of the main disadvantages of basic full-searched vector quantization (FSVQ) is that it typically requires a large number of computations per input pixel, and that it is limited to a single compression rate for a given vector size and codebook. Several modifications to the FSVQ algorithm have been proposed to reduce the computational requirements of the method such as residual vector quantization (RVQ), tree-structured vector quantization (TSVQ) and various combinations of these [1]. Other computational savings can be realized using fast-search algorithms [3]. Recently, RVQ has seen commercial application as the basis of Sorenson Vision's Quicktime codec [4]. While these techniques provide low-complexity encoders, they also produce sub-optimal results.

The fixed-rate limitation of FSVQ has been addressed using various adaptive-search and variable-length encoding methods [4, 5]. These methods use rate-constrained buffer-feedback to determine the number of levels in a coding structure to use to encode each input vector. The number of levels, or "coding units" are chosen so as to provide the optimal rate-distortion tradeoff for the encoding. One of the main limitations on the performance of adaptive-search methods is that the underlying structures in these methods are suboptimal.

In this paper we briefly describe rate-constrained adaptive tree-structured RVQ (TRVQ), and then introduce a new structure, full-search tree residual VQ (FTRVQ), which allows most of the performance provided from a full-searched codebook, but also allows for a variable length code for each input vector. Section 2 introduces TRVQ and FTRVQ. A search method that simplifies the complexity of FTRVQ is then described in Section 3. A video coding algorithm used to test the performance of these structures is described in section 4 and simulation results are presented in section 5.

2. RATE-CONSTRAINED ADAPTIVE RVQ CODING

2.1. Tree-Structured RVQ

One structure for a rate-constrained adaptive TRVQ is given in Figure 1.

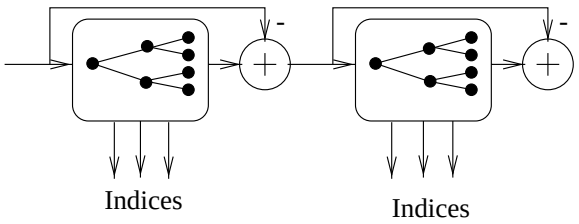

Figure 1: Two stages of TRVQ.

This structure can be thought as a generalization of both rate-constrained TSVQ and RVQ, as described in [5]. In this structure, each level of the tree in each residual stage is considered as resulting in a unique encoding of the input vector. The encoding is the product code created from each of the tree levels up to and including the level under consideration together with all prior residual stages. Each of these encodings are then assigned a cost, and the lowest cost encoding is used to code the input vector.

The cost function used to decide the best encoding is given by

$$J_i = D_i + \lambda R_i, \quad (1)$$

where D_i is the distortion of the i^{th} encoding, R_i is the total length (in bits) of the encoding (including a small header of overhead information to represent i), and λ is a parameter related to the slope of the rate-distortion curve. For a two-stage, three codebook structure (six coding units), a total of six costs are evaluated and the smallest is chosen.

The value of the Lagrangian minimization parameter λ is updated using buffer feedback as shown in Figure 2. A

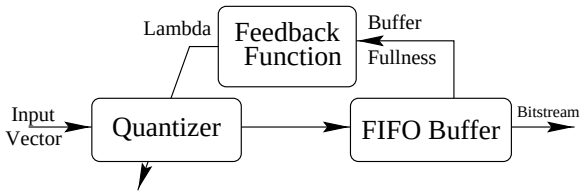

Figure 2: Rate control by buffer fullness feedback.

negative feedback function[6] is used to map the buffer fullness level to a value of λ which increases as the buffer fills. This feedback control causes the encoder to generate fewer bits until an equilibrium is reached. Note that the rate generated by the encoder can be changed at any time by changing the rate at which the buffer is emptied. This rate distortion minimization is related to entropy constrained vector quantization [7].

2.2. Full-Search Tree RVQ

The suboptimal performance of the adaptive TRVQ described previously can be improved upon by replacing the tree search with a set of FSVQ codebooks, each with an increasing number of codevectors. For example, a set of codebooks with 16, 256, and 2048 codevectors can be used in the FTRVQ structure instead of a tree structure using 4 bits per level. Since each succeeding codebook is indexed with an additional 4 bits of address, a cost can be computed which trades the number of bits used by selecting each size codebook with the distortion from encoding with that codebook. This idea can be extended to multiple stages of sets of FSVQ codebooks to reduce the size of the largest FSVQ codebook that must be searched and still provide small distortion encodings in areas of the images which allow it. Note that the residuals used in following stages are based on the best codevector in the last (largest) codebook in the set of FSVQ codebooks.

The significant cost of the FTRVQ structure is that the larger full-search codebooks require a significantly larger number of computations than a tree structure does. Although the cost of computation is decreasing rapidly, it is still desirable that a fast search method be employed to reduce the computational burden.

3. A TREE-LIKE FAST SEARCH ALGORITHM

The set of codebooks used in the FTRVQ algorithm are similar in size relationship to a tree structure. We can take advantage of this similarity to simplify the search by using the best codevector found in the previous "level" codebook as an "anchor point"[1, p. 481] for a search for the next level FS codebook. The first level is full-searched in a normal fashion to establish the first set of anchor points.

Since we know the closest codevector in the previous level of the "full-search tree," the current anchor point, we can construct a table of all possible codevectors in the current level that could be closer to the input vector than the anchor point. By simple geometric argument, illustrated in Figure 3 for an L_2 distortion measure, the region of all possible codevectors closer to the source vector is bounded by a ball with radius twice the distance from the anchor point to the farthest distance in the Voroni cell from the anchor point. In the Figure, x represents the source vector and o represents a possible codevector. The best vector in the previous level is the vector a_4. We can see in the example that o is closer than a_4, even though x is located in a_4's Voroni cell in the previous level codebook.

We can then reduce the search area by testing each of the possible vectors in the set defined above by using the following tests. Define c_i as the current minimum distance codevector in this level, $h_i = d(c_i, x)$ as the minimum distance, and $d(a_k, x)$ as the minimum distance in the previous level.

1. If

$$d(0,x) - h_i \leq d(0,c_j) \leq d(0,x) + h_i, \quad (2)$$

2. and

$$d(a_k,x) - h_i \leq d(a_k,c_j) \leq d(a_k,x) + h_i, \quad (3)$$

then c_j is a potential minimum distance codevector. Equation 2 is the same test presented as Algorithm II in [3]. The implementation of these tests require storage for the

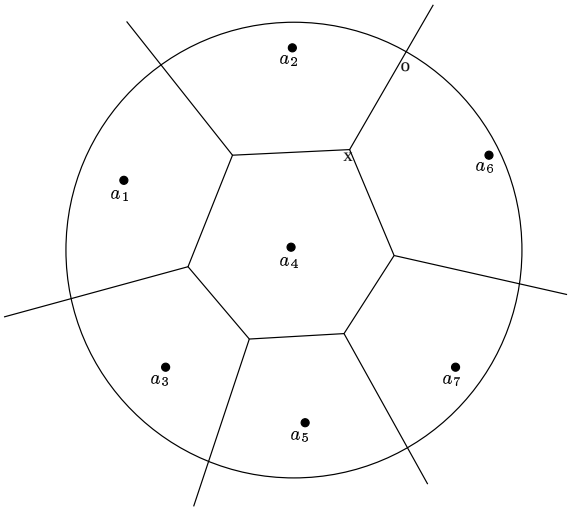

Figure 3: Maximum search region for possible closest codevectors.

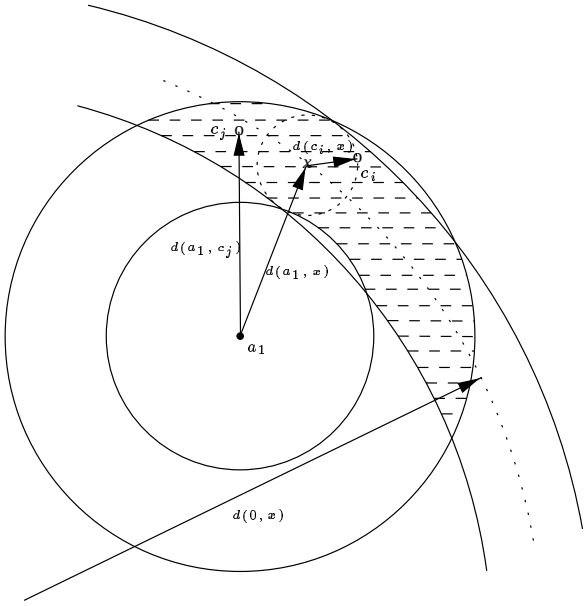

Figure 4: Search region for possible closest codevectors.

$d(a_k, c_j)$ and $d(0, c_j)$, which can be computed and stored before encoding.

Figure 4 illustrates the area of potential closest codevectors. In the Figure, the dotted circle represents the area of closest vectors, and the dashed area is the area restricted by (2) and (3). Note that c_j is still tested for distance, even though it is not closer than c_i.

4. A VIDEO CODING ALGORITHM USING RATE-DISTORTION ADAPTIVE-SEARCH METHODS

Simulations on video sequences were performed with an adaptive-search, locally rate-distortion optimal, mean-removed quantizer described in [4]. The target channel is constant-bit-rate, so a virtual buffer is used to constrain the rate. The bits produced by the quantization are fed to the buffer, before being passed to the channel at a constant rate.

A simple quad-tree decomposition is used to maintain good performance at low rates. A basic block size of 16x16 pixels is used, with the following options: (a) Transmit only a mean for the 16x16 block. (b) Break the basic block into 16x8 blocks and send a mean for each of those. (c) Break one or both of the 16x8 blocks into 8x8 blocks and either send just a mean for them, or means and VQ indices. When coding a motion-compensated residual, the option of doing nothing is also considered.

An important feature of this algorithm is its ability to encode at multiple rates. This is achieved merely by extracting data from the buffer at the desired rate. This change in data rate can occur at any time during encoding. Huffman codes are used to encode the motion offsets, means, a header indicating how many VQ coding units to use, and the mode. Though a slight benefit is observed for training Huffman tables for each rate, we used tables trained a low rates to encode at all rates.

Block-based motion compensation is used to remove temporal redundancy. Simple segmentation into 16x16 or 8x8 blocks is provided as in the H.263 standard [8]. A locally rate-distortion-optimal decision [9] is made between the following four encoding-mode options: (a) No-update (or conditional-update), indicates that the corresponding block from the previously decoded frame is to be used, (b) Intra coding indicates that the block is to be coded without the benefit of motion compensation, (c) A motion residual using 16x16 motion compensation is coded, (d) A motion residual resulting from 8x8 motion compensation is coded.

During decoding, the Huffman code for the mode is first decoded, indicating which of the four encoding options are used. If motion is used, one or four motion offsets are decoded. If VQ is used, a header is first decoded, then a mean and several VQ indices, depending on the value indicated by the header [9]. Finally, a direct sum of a mean vector, VQ codevectors, and a motion compensation vector is made to obtain the decoded vector.

5. EXPERIMENTAL RESULTS

The performance of the fast full-search tree algorithm of Section 3 was tested using 824,380 16-dimensional vectors derived from 8 images of various types. Both the performance improvement in the number of distance computa-

tions and memory requirements was investigated. The fast-search method requires that a table of distances between the previous level codevectors and the current level codevectors be computed and stored, and a table of "energy" values (distances from the origin) also be computed and stored. The total number of possible storage locations for the k^{th} level of the full-search tree is given by

$$mem_k = 2^{bits(k)+bits(k-1)} + 2^{bits(k)} \qquad (4)$$

where $bits(k)$ is the number of bits in the k^{th} level. The first term in (4) is required by (3), and the second is required by (2). The total number of distance computations (compares) for FSVQ, without the fast-search algorithm, is given by $2^{bits(k)}$ compares.

Table 1: Distance Computation and Memory

"Tree" Level	Coding Units	Codebook Vectors	Average Compares (% of Max)	Average Memory (% of Max)
1	1	16	100	100
2	2	256	16.6	90
3	3	4096	13.5	68

Table 1 lists the performance of the fast-search algorithm on the test set for 4-bit increases per level. Note that the number of compares required is a little more than 10% of the maximum, meaning that between two to three bits of equivalent search time is saved by the fast-search. The distribution of compares for the 4096 codevector level is given in Figure 5. The distribution indicates that the number of compares is highly peaked about the average, suggesting that a suboptimal variation of this fast search may perform better than a tree-search, but with a smaller fixed number of distance compares.

When we store distances for only the codevectors in the region illustrated in Figure 3, the reduction in the first term of (4) is also reported. The advantage of restricting the distances in the table to possible codevectors does not seem to be impressive in our empirical study, with a memory requirement of about 70–90% of the maximum storage required.

Table 2: Memory Requirements

Algorithm	Coding Units	Levels	Memory
RVQ.6	6	1	24576
RVQ.12	12	1	49152
TRVQ2.12	12	2	417792
TRVQ3.12	12	3	4472832
FTRVQ2.12	12	2	417792
FTRVQ3.12	12	3	4472832

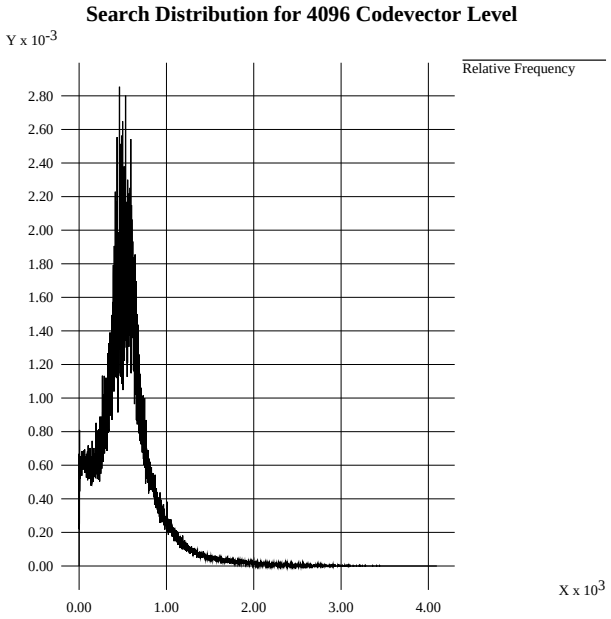

Figure 5: Distribution of distance computations required for a 4096-codevector level.

Six different codebook structures were used to test coding performance. For a more detailed description of the structures, see [5]. The codebooks are listed in Table 2, along with the number of tree levels and the size of the codebook in bytes. The label used in subsequent plots is also listed. Separate codebooks for inter-block and intra-block coding were trained. A training set of 400 frames of 752x480 video containing natural and synthetic scenes was used.

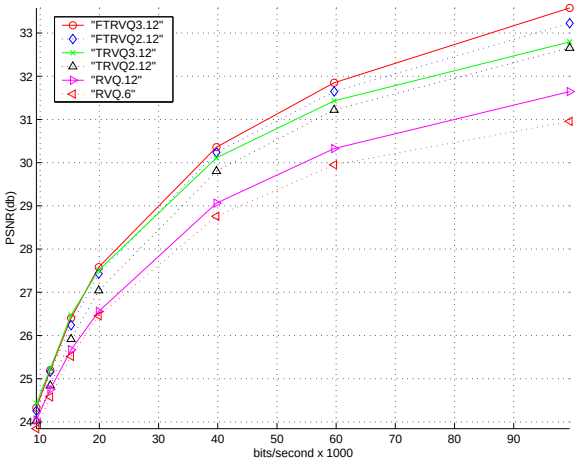

Figure 6: Rate-PSNR curve for `foreman` sequence.

Figure 6 shows the rate-distortion characteristics when

using these codebooks to encode the `foreman` sequence. Figure 7 shows the performance obtained when encoding the `mthr_dotr` sequence. These are the H.263 test sequences temporally downsampled to 10 frames per second. These figures illustrate the benefit of TRVQ and FTRVQ. As we increase the amount of memory used for the codebooks (increase the number of tree levels), the performance increases. In addition, as we increase the number of full-search codebooks used, the performance also increases.

Inspection of these figures reveals that the benefit from encoding with the more complex TRVQ and FSVQ structures increases with rate. For example, for the `mthr_dotr` sequence, we obtain 0.6 db PSNR improvement from TRVQ2 over RVQ, and one db improvement at high rates. For FSTRVQ3, the improvement ranges from one to three db. We also note that at low rates, the bit allocation approaches the mean-only case for intra coding, or the predicted-only case for inter-frame coding, and TRVQ and RVQ give the same results.

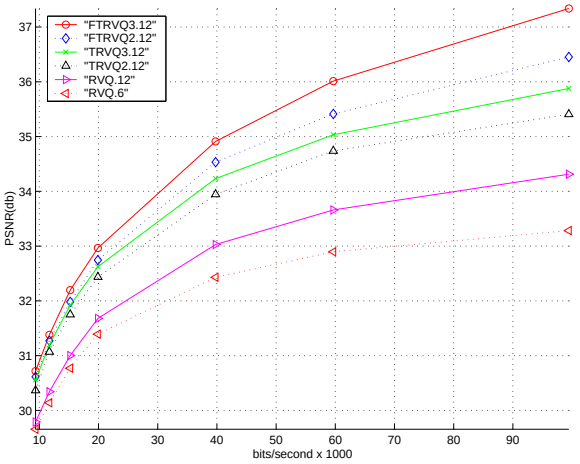

Figure 7: Rate-PSNR curve for `mthr_dotr` sequence.

6. CONCLUSION

In conclusion, this paper presents the FTRVQ algorithm, including a fast search method to reduce the computational requirements needed to full-search the "tree" codebooks. Performance on video sequences as compared to other adaptive-search, variable-length encoding methods was also presented. Although the computational requirements of FTRVQ are still high, the fast search reduced the requirements by almost 90% over ordinary full searches. Experiments indicate the FTRVQ algorithm performed better than equivalent-codebook-size TRVQ by up to 1 dB at 56k modem rates for low-motion sequences. As memory and computational capabilities increase, the FTRVQ algorithm has potential for high-quality video compression at low bit-rates.

7. REFERENCES

[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1992.

[2] C. F. Barnes, S. A. Rizvi, and N. M. Nasrabadi, "Advances in residual vector quantization, a review," vol. 5, pp. 226–262, February 1996.

[3] 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, pp. 413–416, July 1992.

[4] K. Liang, C.-M. Huang, A. K. Huber, and P. D. Israelsen, "Variable block size multistage VQ for video coding," in *Conference of the IEEE Industrial Electronics Society (IECON)*, Nov. 1999.

[5] C. B. Peel, X. Liu, and S. E. Budge, "Adaptive-rate tree-structured residual vector quantization," in *Proceedings of the IEEE ICASSP*, pp. 1887–1890, IEEE, June 2000.

[6] J. Choi and D. Park, "A stable feedback control of the buffer state using the controlled Lagrange multiplier method," *IEEE Trans. Image Processing*, vol. 3, pp. 546–558, Sept. 1994.

[7] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy constrained vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 31–42, Jan. 1989.

[8] *ITU–T Recommendation H.263 "Video coding for low bitrate communication"*, June 1996.

[9] C. B. Peel, S. E. Budge, K. Liang, and C.-M. Huang, "Locally optimal, buffer-constrained motion estimation and mode selection for video sequences," in *Proceedings of the IEEE ICASSP*, vol. 6, IMDSP-7.7, pp. 3369–3372, IEEE, Mar. 1999.