Optimised feature map finite-state vector quantisation for image coding

C. Zhu, L.-M. Po and Y. Hua

Abstract: An optimised feature map finite-state vector quantisation (referred to as optimised FMFSVQ) is presented for image coding. Based on the block-based gradient descent search algorithm used for motion estimation in video coding, the optimised FMFSVQ system finds a neighbourhood-based optimal codevector for each input vector by extending the associated state codebook stage by stage, thus rendering each state quantiser a variable rate vector quantisation. The optimised FMFSVQ system can be interpreted as a cascade of a finite-state vector quantiser and classified vector quantisers. Furthermore, an adaptive optimised FMFSVQ is obtained. Experiments demonstrate the superior rate-distortion performance of the adaptive optimised FMFSVQ compared with the original adaptive FMFSVQ and the memoryless vector quantisation.

1 Introduction

Finite-state vector quantisation (FSVQ) is a technique that exploits both the intra- and inter-vector correlation to achieve much better rate-distortion performance than ordinary memoryless VQ [1, 2]. The FSVQ has been widely applied in image coding [3-8]. The FSVQ employs a finite number of states that are determined by the reproduction codevectors associated with the neighbouring image block vectors of a current block vector. Each state corresponds to a codebook called a state codebook which is a subset of a super-codebook. Recently, Nasrabadi et al. proposed in [7] a dynamic FSVQ (DFSVQ) with a fixed rate, which generates on-line state codebooks from a universal super-codebook. However, the DFSVQ takes a large amount of time to generate a state codebook on-line, where the state codebook is selected from a super-codebook by a next-state function (a rendering procedure in the DFSVQ) that requires storing large conditional vector transition probability matrices and extensive computations. A feature map FSVQ (FMFSVQ) was introduced by Xu and Kuh [8], which utilises Kohonen's self-organising feature map (SOFM) [9] to construct an 'ordered' super-codebook. With the 'ordered' super-codebook, state codebooks are generated by using a simple neighbourhood topological distance instead of the score function as used in the DFSVQ [7] or the side-match distortion in the side-match FSVQ [5]. As a consequence, the FMFSVQ significantly reduces the computation complexity for the on-line state codebook generation during the coding procedure. However, there is a significant potential for the FMFSVQ to perform better. In this paper, an optimised FMFSVQ is developed by incorporating an effective algorithm used for motion estimation in video coding [10, 11] to achieve a better performance.

2 Review of feature map FSVQ

The self-organising feature map (SOFM), introduced by Kohonen [9], is an unsupervised learning process which can be used for data clustering. Like VQ, the SOFM defines a mapping from an input space $\mathbb{R}^d$ onto a topologically ordered set of nodes where each node is associated with a $K$-dimensional vector. The set of nodes are usually arranged in a much lower dimensional map such as a 2-D plane. A 2-D SOFM is shown in Fig. 1. In the on-line learning process, for an input vector $x$, the winning node closest to the input and its predefined neighbourhood nodes in the map are updated with their corresponding vectors closer to the input. The most attractive characteristic of SOFM is its capability of topological ordering/preserving due to the Neighbourhood learning mechanism. The closer the nodes are to each other in the SOFM in terms of topological distance, the closer the associated codevectors generally are to each other (in terms of Euclidean distance). This property is often called topology preservation or ordering preservation. Therefore, the

![Fig. 1](image-url)
SOFM is a good choice for designing codebooks with ordered structure.

In the feature map FSVQ system, a super-codebook is designed using Kohonen's self-organising feature map (SOFM) [9]. Thus, the designed super-codebook has the advantage of ordered structure in the map. We know that neighbouring block vectors in an image also generally correspond to closer vectors in the Euclidean space. Therefore, the best match reproduction codevectors for neighbouring image block vectors are generally close in terms of topological distance in the SOFM. Fig. 1 illustrates a current image block \(X\) and its four neighbouring blocks \(A, B, C, D\) corresponding to their reproduction code vectors \(Y_A, Y_B, Y_C, Y_D\) of the causal neighbours which are located in a 2-D SOFM, usually with small topological distances.

The FMFSVQ employs a very simple method for generating a state codebook on-line. Let \(S_y\) be a set of \(N_y\) codevectors that are the nearest to the codevector \(y\) in topological distance. The state codebook for the current input \(X\) (shown in Fig. 1) consists of the four sets of code vectors: \(S_x, S_y, S_y, S_y\), which has a size of \(4 \times N_y\). However, the number \(N_y\) affects the coding efficiency directly, as can be observed from Table 1 in [8]. Some input vectors may find their best template reproduction codevector in a state codebook of smaller size, while others may find the best template reproduction in a state codebook of larger size. To gain a better bit-rate-distortion performance, therefore, a state codebook with a variable size for each input vector (not for each state) is desired. Moreover, the best match reproduction in the super-codebook for an input vector is desired to be included in the associated state codebook. A scheme aimed at achieving this is developed in the following.

3 Optimised feature map FSVQ

3.1 Block-based gradient descent search algorithm for block motion estimation

In [11], a block-based gradient descent search (BBGDS) algorithm is developed to perform block motion estimation in video coding. In the block motion estimation field for real world video sequences, it is observed that, most motion trajectories are enclosed in a small central area around the central point corresponding to the zero motion vector. Based on the characteristic of central-motion biased vector distribution, the BBGDS takes this advantage by only searching all points around the central point in a block referred to as the checking block in the first step. Checking blocks are usually taken as \(3 \times 3\) or \(5 \times 5\) squares. From the centre-biased checking block, the BBGDS searches for a motion vector along the block-based gradient descent direction where the best match vector corresponding to the minimum distortion is expected to lie. The procedure continues until the winning point associated with a vector is in a central point of the checking block. The BBGDS has shown its competitive performance with reduced computational complexity compared with some other fast motion estimation methods.

Note that the FMFSVQ coding in a 2-D SOFM is very similar to the motion estimation in a search region for video coding [10, 11]. Therefore, we consider applying the fast algorithms for motion estimation to achieve efficient coding in the FMFSVQ. With the FMFSVQ coding in a 2-D SOFM corresponding to the motion estimation in a search region, we aim to apply the BBGDS algorithm in the FMFSVQ for more efficient coding in this paper.

3.2 Block-based gradient descent search algorithm for FMFSVQ

A codevector is called the \(N_y\)-block-based (or \(N_y\)-neighbourhood-based) optimal codevector for \(X\), if the codevector corresponds to the centre point of a checking block of size \(N_y\) in the SOFM and the codevector has the smallest reproduction distortion among all the codevectors in the block. To find such a \(N_y\)-neighbourhood-based optimal codevector for each input vector, an optimised FMFSVQ with the BBGDS algorithm for image coding can be developed. An example of the encoding process in the optimised FMFSVQ is illustrated in Fig. 2. The optimised FMFSVQ can be formulated as follows.

It is known that the FSVQ can be considered as a form of classified VQ (CVQ) [1, 12] with each state implicitly representing a class. Unlike the CVQ, however, the FSVQ does not explicitly produce any class index but a state index based on the previous code vectors. Here, we refer to the information obtained from the current input vector as class, and the information obtained from the previous replications as state. We can see below that the optimised FMFSVQ can be interpreted as a cascade of an FSVQ and CVQs. First, like an ordinary FSVQ, the best match codevector \(y_1\) (1) for the current input vector \(X\) is found in the stage 1 state codebook \(SC_1\), which is generated based on the state information. Instead of taking \(y_1\) as the reproduction codevector for \(X\), we use \(y_1\) as the class information because \(y_1\) is obtained from classifying \(X\) in \(SC_1\). Then a corresponding class sub-codebook (i.e., the 2nd stage state codebook \(SC_2\) in the optimised FMFSVQ) is used to find a possible better reproduction than \(y_1\). If there is a better reproduction codevector \(y_2\) (2) in \(SC_2\), then it is used as improved class information for choosing the next class sub-codebook. The improvement process continues until a class sub-codebook does not contain a better codevector than the current class information \(y_1\).

It can be seen that the reproduction distortion is decreased in the process, during which the bit rate is increased appropriately. The new interpretation may lead to some other design algorithms based on flexible combinations of

![Fig. 2. Example of illustrating optimised FMFSVQ encoding process in finding \(N_y\)-neighbourhood-based optimal codevector for \(X\) with \(N_y = 3 \times 3\) and \(N_y = 5 \times 5\).](image-url)
3.3 Encoding/decoding algorithms

The coding algorithms can be summarized as follows.

3.3.1 Encoding algorithm:

0. Select $N_c$ (e.g., $1, 5, 3 \times 3$ or $5 \times 5$) for the size of the initial state codebook and $N_p$ (e.g., $3 \times 3$ or $5 \times 5$) for the size of the checking block.

1. Let $m$ be the stage number of the state codebook. Set $m = 1$ initially. For a current input vector $X$ with four neighbouring reproduction vectors $x_0, x_1, y_0, y_1$ organise the first stage state codebook $SC_1 = \bigcup_{y \in \{0, 1\}^2} S_y$ by removing duplicate codewords, where $S_y$ is the set of $N_c$ codewords nearest to $y$ in topological distance. Note that the size $M_1$ of $SC_1$ is usually smaller than $4 \times N_c$.

2. In $SC_1$, find the best match codeword $x_1(1)$ for $X$ corresponding to the minimum distortion. The index of $x_1(1)$ in $SC_1$ is sent to the decoder with $\lfloor \log_2 M_1 \rfloor$ bits, where $|r|$ returns the least integral value greater than or equal to $r$.

3. Set $m = m + 1$. Let $Q_m$ consist of the nearest $N_p$ code-vectors around $x_1(m - 1)$ in topological distance.

4. If $Q_m \subseteq \bigcup_{y \in \{0, 1\}^2} S_y$, then $x_1(m - 1)$ is an $N_p$-neighbourhood-based optimal codeword, go to step 1 for encoding the next input vector. Otherwise, go to the next step.

5. Construct the $m$th stage state codebook $SC_m = Q_m - (Q_m \cap \bigcup_{y \in \{0, 1\}^2} S_y)$ with the size $M_m$. Determine from $SC_m$ the best match codeword $x_1(m)$ that is the closest to $X$ in Euclidean distance.

6. If $d(X, x_1(m - 1)) < d(X, x_1(m))$, where $d(\cdot, \cdot)$ represents the Euclidean distance between two vectors, then $x_1(m) - 1$ is an $N_p$-neighbourhood-based optimal codevector. Then go to step 1 to encode the next input vector. Otherwise, transmit the index of $x_1(m)$ in $SC_m$ using $\lfloor \log_2 M_m \rfloor$ bits preceded by a non-ending flag $w$, and go to step 3.

3.3.2 Decoding algorithm:

1. Let $m$ be the stage number of the state codebook. Set $m = 1$ initially. For the current decoded vector $X$ with four neighbouring reproductions $x_0, x_1, y_0, y_1$, the stage 1 state codebook $SC_1$ is organised as in step 1 of encoding.

2. According to the size $M_1$ of $SC_1$, the index length for $y_1(1)$ is determined. Then $y_1(1)$ can be restored from $SC_1$ with the corresponding index.

3. Set $m = m + 1$. Let $Q_m$ consist of the nearest $N_p$ code-vectors around $y_1(m - 1)$ in topological distance.

4. If $Q_m \subseteq \bigcup_{y \in \{0, 1\}^2} S_y$, the current vector is reconstructed by $y_1(m - 1)$, and go to step 1 to decode the next vector. Otherwise, go to step 5.

5. If the ending flag $w$ follows the previous index, the current vector is reconstructed by $y_1(m - 1)$, and go to step 1 to decode the next vector. Otherwise, go to step 6.

6. Construct the $m$th stage state codebook $SC_m = Q_m - (Q_m \cap \bigcup_{y \in \{0, 1\}^2} S_y)$, $y_1(m)$ from $SC_m$ using the following index whose length is determined by the size $M_m$ of $SC_m$. Note that the rounding flag $w$ preceding the index should be skipped. Go to step 3.

3.4 Some remarks of optimized FMFSVQ system

It is clear that in the optimised FMFSVQ coding system, different input vectors may require different number of stages and different sizes of state codebook at each stage. Therefore, the optimised FMFSVQ is a variable rate FSVO. It should be noted that there is no fixed state codebook for each state. Each state quantiser in the optimised FMFSVQ system is a variable rate VQ. For input vectors in the same state, only the first stage state codebook is the same but the state codebooks of the following stages may be completely different. This is a feature distinctive from other variable rate FSVQs [3-6].

The block-based gradient descent search (BGDS) algorithm was originally developed to perform fast motion estimation in video coding [11]. It has been shown [11] that, compared with some other search schemes, the BGDS method can find a block-based optimal solution faster with reduced computational complexity in terms of fewer distance calculations/comparisons. In essence, finding the best-matched motion vector in video coding is equivalent to finding the best-matched codevector in FMFSVQ image coding. The (original) FMFSVQ makes a full search in a predefined area for the best-match codevector while the optimised FMFSVQ employs an efficient gradient search scheme. Referring to Fig. 2, we can easily see that to find the same neighbourhood-based codeword the optimised FMFSVQ with the BBGDS scheme will use fewer distance calculations and comparisons than the FMFSVQ. In other words, to obtain the same quality compressed images, the optimised FMFSVQ will generally take fewer distance calculations/comparisons than the (original) FMFSVQ, while the optimised FMFSVQ will achieve a lower bit rate. In the optimised FMFSVQ, the number of distance calculations required for encoding each input vector is the sum of the sizes of all stages of state codebooks needed, which is variable with respect to different input vectors and different neighbours. We can see that, just like the bit rate, the number of distance calculations/comparisons is associated with the size of the state codebook. The smaller the size of the state codebook, the fewer the number of distance calculations/comparisons required and the lower the bit rate. Therefore the number of distance calculations/comparisons for the optimised FMFSVQ algorithm can be generally inferred from the associated bit rates.

3.5 Adaptive optimised FMFSVQ

When the input $X$ is not highly correlated with its neighbouring vectors, the first stage state codebook may not contain any good reproduction. Because of the local order preservation in the SOFM, the BBGDS algorithm may lead to a locally optimal code vector of low quality. A common treatment is to perform a full search VQ in the super-codebook when a reproduction corresponding to a small distortion cannot be found in a state codebook. This scheme has been used in the adaptive DHFSVQ [7] and the adaptive FMFSVQ [8]. Similarly, an adaptive optimised FMFSVQ can be obtained. That is, if the distortion between a neighbourhood-based optimal codevector and a current input vector is greater than a threshold $D^*$, then the super-codebook is searched for the best-match codevector. Obviously, a control flag is needed to inform the decoder. Increasing the size of the first stage state codebook by 1, we use the last index of the codebook to represent the control flag, as done in [7]. Furthermore, when the number of bits required to find a neighbourhood-based optimal codevector in the optimised FMFSVQ encoding process is
larger than that required by using full search VQ together with the control flag, we should also use the full search VQ in the super-codebook.

4 Experimental results

Computer simulation was performed to evaluate the performance of the optimised FMFSVQ in comparison with the FMFSVQ and the memoryless VQ in terms of mean squared error (MSE) and bit rate. Two super-codebooks of size 256 (16 × 16 in the 2-D SOFM) and 625 (25 × 25 in the 2-D SOFM) were created using Kohonen's SOFM algorithm on eight 512 × 512 monochrome images of 8-bit grey levels. The image block size is 4 × 4, corresponding to 16-dimensional vectors. Four other images with the same size and same grey levels ('Lena', 'Peppers', 'Airplane' and 'Splash') were used for testing.

In the adaptive FMFSVQ, a flag should be appended to identify which codebook (a small state codebook or the large super-codebook) is used for coding each vector. In this paper, we use the last index of a state codebook (increasing its original size by 1) as the flag for a repro-

| Table 1: Performance comparison of (nonadaptive) FMFSVQ, (nonadaptive) optimised FMFSVQ and memoryless VQ in terms of bit rate (bpp) and MSE distortion |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| N_s | N_r = 3 × 3 | N_r = 5 × 5 | Super-codebook |       |
| 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 |
| Lena | 0.4380 | 0.5000 | 0.3886 | 0.4045 | 0.5000 |       |
| Airplane | 0.4380 | 0.5000 | 0.3679 | 0.3875 | 0.5000 |       |
| Peppers | 149.06 | 191.19 | 124.86 | 113.44 | 78.17 |       |
| Splash | 109.73 | 90.06 | 96.39 | 91.24 | 66.71 |       |
| Bit rate | 0.4380 | 0.5000 | 0.3524 | 0.3636 | 0.5000 |       |
| MSE | 64.395 | 50.928 | 55.943 | 53.695 | 40.965 |       |

The super-codebook of size 256 is used. All the bit rates are calculated with each coding index being represented by the least integer number of bits.

| Table 2: Performance comparison of adaptive FMFSVQ, adaptive optimised FMFSVQ, and memoryless VQ in terms of bit rate (bpp) and MSE distortion |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| N_s | N_r = 3 × 3 | N_r = 5 × 5 | Super-codebook |       |
| 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 | 5 × 5 | 7 × 7 |
| Lena | 0.2846 | 0.3822 | 0.4343 | 0.3172 | 0.5000 |       |
| Airplane | 0.3053 | 0.4073 | 0.4617 | 0.3124 | 0.5000 |       |
| Peppers | 95.15 | 68.48 | 61.03 | 57.86 | 78.17 |       |
| Splash | 89.32 | 74.11 | 70.96 | 70.55 | 66.71 |       |
| Bit rate | 0.2972 | 0.3771 | 0.4319 | 0.3134 | 0.5000 |       |
| MSE | 60.30 | 47.17 | 44.22 | 43.96 | 40.96 |       |

The super-codebooks of size 256 and 625 are used, respectively. The bit rate for the memoryless VQ with the super-codebook of size 625 is $\frac{\log_{2}625}{16} = 0.625$ bpp.
duction codevector that is selected from the super-codebook. Then the bit rate for the adaptive FMFSVQ is given by
\[
R = \frac{P_{1c}}{4 \times 4} \times \left( \frac{\log_2(4 \times N_c + 1)}{4 \times 4} \right) + (1 - P_{1c}) \times \left( \frac{\log_2(M)}{4 \times 4} \right) + 1
\]

where one bit is used as the flag, e.g. bit '0' for the small codebook and '1' for the super-codebook. For \( N_c = 1 \), \( N_c = 5 \), or \( N_c = (2n + 1) \times (2n + 1), \) \( n = 1, 2, \ldots \), with a centre in the block of size \( N_c \), it is easy to see that the flag scheme corresponding to \( N_c = 1 \) is more efficient than that associated with \( N_c = 5 \) in terms of bit rate in the case of \( P_{1c} \) being close to 1.

The experimental results of the optimised FMFSVQ in comparison with the FMSVQ and the ordinary memoryless VQ are summarised in Table 1. We can clearly see that the optimised FMFSVQ outperforms the FMSVQ. However, the results for both the nonadaptive versions are undesirable. Table 2 presents the results of the adaptive versions. For the adaptive FMSVQ [8] and the adaptive optimised FMFSVQ, the distortion threshold \( D^* \) was set to 3000, as in [7]. \( N_c = 1 \) and \( N_c = 3 \times 3 \) were employed for the adaptive optimised FMFSVQ. \( N_c = 1, N_c = 5 \) and \( N_c = 3 \times 3 \) were considered separately for the adaptive FMFSVQ. It can be seen that the adaptive optimised FMFSVQ achieves a much better performance than the adaptive FMFSVQ. Compared with the memoryless VQ, the adaptive optimised FMFSVQ obtains similar MSE distortions with a bit rate reduction ~40% or more.

In addition, we tested the robustness of the rate-distortion performance to the threshold \( D^* \) for the adaptive optimised FMFSVQ and the adaptive FMSVQ. Our experiment showed that, compared with the adaptive FMSVQ, the adaptive optimised FMFSVQ is more robust to the choice of \( D^* \) varying from 2000 to 24,000. This is because the optimised FMFSVQ can find at least a suboptimal (neighbourhood-based optimal) reproduction for each input vector regardless of \( D^* \). This result suggests that the optimised FMFSVQ has a better potential for mitigating the deterioration caused after a bad reproduction has to be made for an input vector [1].

5 Conclusions

By incorporating the block-based gradient descent search (BBGDS) algorithm [11] used for motion estimation in video coding, the optimised FMFSVQ system has been developed for image coding. The optimised FMFSVQ can be interpreted as a cascade of an FMSVQ and classified VQs (CVQs). The experimental results have clearly demonstrated that the optimised FMFSVQ and its adaptive scheme achieve much better rate-distortion performance than the original FMSVQ and the adaptive FMSVQ, respectively. The adaptive optimised FMFSVQ can obtain a similar MSE distortion as the memoryless full search VQ with a significant reduction in bit rate. Furthermore, the adaptive optimised FMFSVQ is found to have better robustness to the threshold \( D^* \) than the adaptive FMSVQ. Finally, we would like to point out that some other fast algorithms for motion estimation, such as 2-D logarithmic search, three-step search, conjugate direction search and hierarchical block matching [10], can also be applied in the FMFSVQ for efficient coding.

6 References