Comparative analysis of bitplane-based wavelet image coders

L.C.R.L. Feio and E.A.B. da Silva

Abstract: Wavelet image encoders based on bitplanes give excellent results in the compression of still images. The bitplane concept has recently been generalised to vectors, and wavelet image encoders based on vector bitplanes have been proposed, some achieving slightly better rate × distortion performances than scalar encoders. There remains the open question of whether the use of vector bitplanes has the potential of providing more significant rate × distortion improvements over scalar versions. The authors address this question by analysing in detail the performance of adaptations, for the use of vector bitplanes, of four popular wavelet-based bitplane encoders. From this analysis, they determine where the gains in performance of encoders based on vector bitplanes come from. It is concluded that performance improvements may come by increasing the vector dimension, provided that codebooks with good packing properties are used.

1 Introduction

Most image coding techniques can be split into three main stages: at the transformation stage, image pixels are transformed into a set of values (coefficients), which can be more easily coded. After that, in the quantisation stage, the generated values are quantised so that they can be represented using a limited number of symbols. Finally, in the coding stage, symbols resulting from the quantisation stage are assigned to binary codewords.

Wavelet image encoders represent the state-of-the-art in image compression [1–4]. Among them, those using bitplane encoding stand out as the most popular. In this type of coding, the coefficients of the transformed image are encoded through successive approximations: at each iteration a bitplane of the coefficients obtained in the transformation stage is encoded.

These encoders provide excellent performance for still image coding, being part of many important standards, among them JPEG-2000 [5, 6], and the texture coding procedure of MPEG4 [7].

In bitplane coding, a coefficient $-1 < c < 1$ is represented by a sequence $\{s, b_1, b_2, \ldots, b_n \ldots\}$ such that:

$$c = s \sum_{i=1}^{\infty} b_i 2^{-i} \tag{1}$$

where $s \in \{-1, 1\}$ represents the sign of c and $b_i \in \{0, 1\}$. Alternatively, we can have $s = 1$ and $b_i \in \{-1, 1\}$.

In practical applications, the above sum is not infinite, but limited to a value P, which indicates the number of bitplanes used for coding the coefficient. This number P is directly related to the distortion level of the reconstructed image. Among the wavelet image encoders that use this concept, we must mention the EZW [1] (embedded zerotree wavelet) encoder and its variation, the SPIHT [2] (set partition in hierarchical trees) encoder. Another interesting bitplane encoder is the MGE [4] (multigrid embedding) encoder. The idea of bitplanes has been generalised to vectors with the introduction of successive approximation vector quantisation [8]. In this, each vector of coefficients is decomposed as a sum of vectors of decreasing magnitude, as follows [9]:

$$\boldsymbol{v} = \sum_{i=0}^{\infty} \boldsymbol{u}_{n_i} \alpha^{-i} \tag{2}$$

where $\boldsymbol{u}_{n_i}$ is a unit-norm vector that belongs to a given N-dimensional dictionary C_N and $0.5 \leq \alpha < 1$ is a constant depending on the dictionary C_N used.

As in the case of scalar bitplanes (1), the above sum is not infinite, but limited to a value Q, which indicates the number of bitplanes used for coding the vector $\boldsymbol{v}$. Therefore, using Q bitplanes, a vector can be approximated as

$$\boldsymbol{v}_Q = \sum_{i=1}^{Q} \boldsymbol{u}_{n_i} \alpha^{-i} \tag{3}$$

It can be shown that if α is chosen conveniently, the error $\|\boldsymbol{v} - \boldsymbol{v}_Q\|$ tends to zero as the number of factors Q tends to infinite. In [10], sufficient conditions are derived for this to happen:

$$\frac{1}{2\cos[\Theta(C_N)]} \leq \alpha < 1, \quad \Theta(C_N) \leq 45^\circ \tag{4}$$

$$\sin[\Theta(C_N)] \leq \alpha < 1, \quad \Theta(C_N) \geq 45^\circ \tag{5}$$

where $\Theta(C_N)$ is the maximum angle between any vector in the space $\mathbb{R}^N$ and the nearest vector in the dictionary C_N.

In [8] the SAWVQ algorithm, an EZW-like algorithm which replaces the scalar bitplanes, (1), by vector bitplanes, (2), is proposed. The codebooks used were the first shells of regular lattices related to the best known sphere packings in dimensions 4, 8 and 16. A significant improvement has been obtained over the scalar version. Since then, several other bitplane-based wavelet coding algorithms have been proposed, with much better performance than EZW [1].

IEE Proceedings online no. 20040300

doi: 10.1049/ip-vis:20040300

Paper first received 1st August 2002 and in revised form 26th September 2003

The authors are with the Universidade Federal do Rio de Janeiro, PEE/COPPE/DEL/EE, Cx. P. 68504, Rio de Janeiro, RJ, 21945-970, Brazil

These include the SPIHT [2], the SWEET [3] and the MGE [4] algorithms. In this context, it is natural to wonder whether the use of vector bitplanes to replace scalar ones, can give rise to the same sort of improvements as in the case of the EZW algorithm. In other words, it is important to determine if the improvement obtained by the use of vector bitplanes is intrinsic to the concept of vector bitplanes or is related specifically to the structure of the EZW algorithm.

The main goal of this paper is to address this issue. We do so by first making a comparative analysis of four different wavelet-based bitplane image encoders: the EZW [1], EZW/C [11], SPIHT [2] and MGE [4] algorithms. This analysis is based on their rate $\times$ distortion performance as well as on the number of bits spent encoding both the location and value of the non-zero coefficients (significance information and the bitplanes themselves). The aim of this analysis is the identification of factors determining differences in the performances of the algorithms. Then, we develop and analyse versions of these encoders using vector bitplanes. Using this analysis we assess quantitatively how the use of vector bitplanes instead of scalar ones affect their performances. Also we identify under which conditions encoders based on vector bitplanes can outperform scalar ones. One should note that algorithms implemented with vector bitplanes have a much higher computational complexity then those using scalar bitplanes. However, when codebooks using regular lattices are employed, one can benefit from several existing fast algorithms to find the nearest vector. For details on these fast algorithms, see [8].

2 Description of the encoders

Since, in the wavelet transforms of natural images, there is a large number of small magnitude coefficients, then their binary representation has a large number of bits equal to zero. Therefore, if there is an efficient way of, instead of coding the zero bits, encoding just the position of the non-zero coefficients, then high compression ratios can be achieved. In fact, this is one of the chief reasons for the efficiency of wavelet-based bitplane encoders.

In bitplane encoding, the first bitplane (bitplane 1), is composed of the most significant bits of the coefficients; the second bitplane (bitplane 2), of the second most significant bits, and so on. A coefficient for a bitplane j is referred to as 'insignificant' if all of its bits for bitplanes $i, i \leq j$, are zero. Otherwise, it is referred to as 'significant'. Two types of information are then encoded: one regarding the position of the significant coefficients, referred to as the 'significance information', and their bitplanes (representing their quantised values).

We then briefly describe the encoders analysed in this work, mainly in terms of how the significance information is encoded. In order to clarify the descriptions, a few definitions are necessary.

- Reference value: is the value that defines a bitplane. The bit corresponding to a coefficient is given by comparison of the absolute value of the coefficient with this reference value. Referring to (1), it could be regarded as a constant multiplying its right-hand side, in order to represent values with magnitude larger than 1. Therefore, its value is chosen before encoding so that the coefficient with maximum magnitude can be represented. For each added bitplane, it is decreased by 2 (or by α, in the vector case, see (2)).
- Significant coefficient: A coefficient is significant when its absolute value is larger than or equal to the reference value.

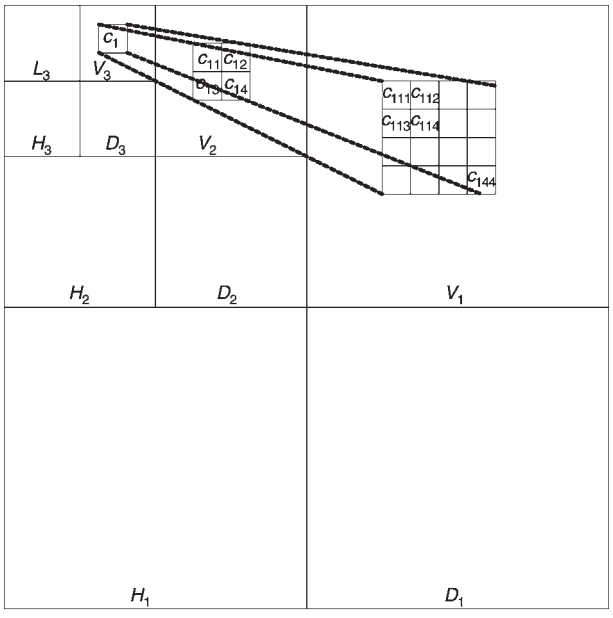


Fig. 1 *Corresponding coefficients in the bands V_i, $i = 1, 2, 3$*

- Insignificant coefficient: A coefficient is insignificant when its absolute value is smaller than the reference value.
- Parent coefficient: The parent coefficient is the coefficient corresponding to the same spacial location that is in the band of the same orientation and immediately lower frequency. This can be exemplified by referring to Fig. 1. The parent coefficient of any of the coefficients c_{11}, c_{12}, c_{13} and c_{14} (band V_2) is the coefficient c_1 (band V_3).
- Children coefficient: The children coefficients are the coefficients corresponding to the same spacial location in the band of the same orientation and immediately higher frequency. In Fig. 1, the children coefficients of c_1 (band V_3) are c_{11}, c_{12}, c_{13} and c_{14} (band V_2). Note that in the case of coefficients that belong to the lower frequency band (e.g. L_3 in Fig. 1), the children are those corresponding to the same spacial location in bands V_3, H_3 and D_3.
- Descendant coefficients: The descendant coefficients are the coefficients corresponding to the same spacial location in all bands of the same orientation and higher frequency. In Fig. 1, descendant coefficients of c_1 (band V_3) are c_{11}, c_{12}, c_{13} and c_{14} (band V_2) and $c_{111}, c_{112}, \ldots, c_{144}$ (band V_1).
- Ascendant coefficient: The ascendant coefficients are the coefficients corresponding to the same spacial location in all bands of the same orientation and lower frequency. In Fig. 1, the ascendant coefficients of any of the coefficients c_{111}, $c_{112}, \ldots, c_{144}$ (band V_1), are coefficient c_{11} (band V_2) and coefficient c_1 (band V_1).
- Zerotree: The use of the concept of zerotree allows exploration of the similarities among bands of the same orientation in the wavelet transform. If we have a zerotree in place of a given coefficient, it means that the coefficient and all its descendants are insignificant. We can also illustrate this process observing Fig. 1. If a zero-valued coefficient c_1 is replaced by a zerotree, it means that coefficients c_{11}, c_{12}, $c_{13}, c_{14}, c_{111}, c_{112}, \ldots$ and c_{144} are also zero.

2.1 Encoders using scalar bitplanes

The first encoder to be analysed is the EZW (embedded zerotree wavelet) [1], which was the first successful encoder of this type reported. The reference value is initially defined as half the magnitude of the largest magnitude coefficient. All the coefficients having magnitude larger than the

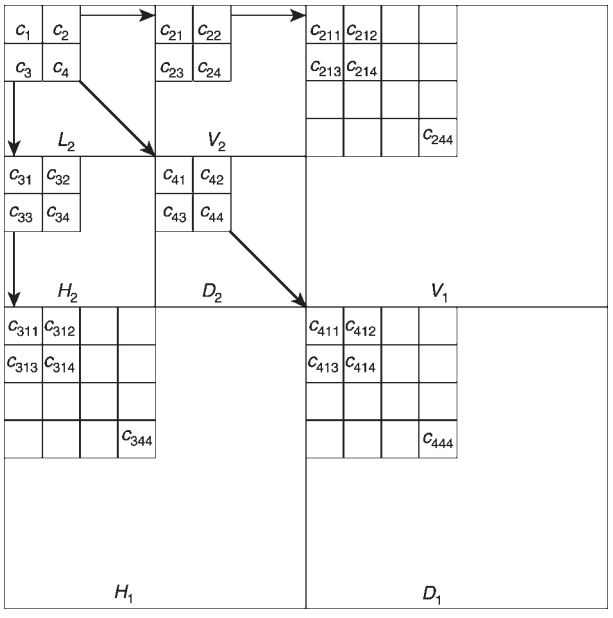


Fig. 2 *Set of SPIHT coefficients for images with a two-stage wavelet transform*

reference value are significant; otherwise, they are insignificant. The signs of the significant coefficients are encoded. Then the reference value is halved, the coefficients previously found significant are refined, and the signs of the coefficients that just became significant are sent. This process is repeated until a stop criterion is met. Also, the significance information is encoded using the concept of zerotrees. All the symbols generated are encoded using an adaptive arithmetic coder [12]. For details the reader is referred to [1]. Another coder analysed is an improvement of the EZW algorithm proposed in [11], referred to as EZW/C. Unlike EZW, in EZW/C there are no symbols such as a zerotree.

Statistical conditioning is used instead. More specifically, the context used by the arithmetic coder to encode each bit of the bitplanes depends on the significance information of the parent of the corresponding coefficient. We used a large number of contexts (e.g. 102 contexts for a six-level wavelet transform), depending on the current pass, the wavelet decomposition level where the coefficient is, significance of the neighbouring coefficients, and significance of parent coefficient. We also perform statistical conditioning to encode the signs of the coefficients, depending on the wavelet decomposition level where the coefficient is, the signs of the neighbouring coefficients, and the sign of the parent coefficient.

We also analyse the SPIHT (set partition in hierarchical trees) [2] algorithm, which is also an improvement of the EZW algorithm, having a considerably superior performance. Its main difference from the previous algorithms is that, while the EZW and EZW/C algorithms assume that all coefficients are significant and indicate explicitly the insignificant ones, the SPIHT algorithm assumes, initially, that all coefficients are insignificant and indicates the ones that become significant. It has a list of insignificant pixels (LIP), a list of insignificant sets (LIS) and a list of significant pixels (LSP). Initially, the LIP keeps the coordinates of all coefficients that are in the lowest frequency band, the LIS keeps the coordinates of coefficients that are in the lowest frequency band, but that have descendants in the other bands, and the LSP is initialised empty. Figure 2 shows this process. This process is carried out through a set of lists of coefficients that allow one to implement, in a very efficient way, the zerotree concept. It should be noted that the SPIHT algorithm also differs from EZW in the form that the coefficients in the lower frequency band are treated. In Fig. 2 we can see that, for the group of four coefficients c_1, c_2, c_3, c_4, coefficient c_1 does not have children, the children of c_2 are c_{21}, c_{22}, c_{23}, c_{24}, the children of c_3 are c_{31}, c_{32}, c_{33}, c_{34} and the children of c_4 are c_{41}, c_{42}, c_{43}, c_{44}. This being the case, we initialise the list LIP with elements c_1, c_2, c_3 and c_4 and the list LIS only with c_2, c_3 and c_4, since c_1 does not have descendants. Initiating the encoding process, the reference value is set to 2^n, where the largest magnitude coefficient can be represented by n bits. Thus, for bitplane n_j, the LIP is scanned and if a coefficient becomes significant for this bitplane its coordinates are moved to the LSP. Then, the LIS is scanned, and if a given coefficient is significant, all of its descendants have to be tested. If one of them is significant, their coordinates are moved to the LSP, else their coordinates are moved to the LIP. The coefficients that are in LSP, except those that were placed there during the current pass, are then refined. Finally, the next bitplane is scanned, with the reference value being set to 2^{n_j-1}. This process is repeated until a stop criterion is met. All the symbols generated are encoded using an adaptive arithmetic coder [12]. For more details see [2].

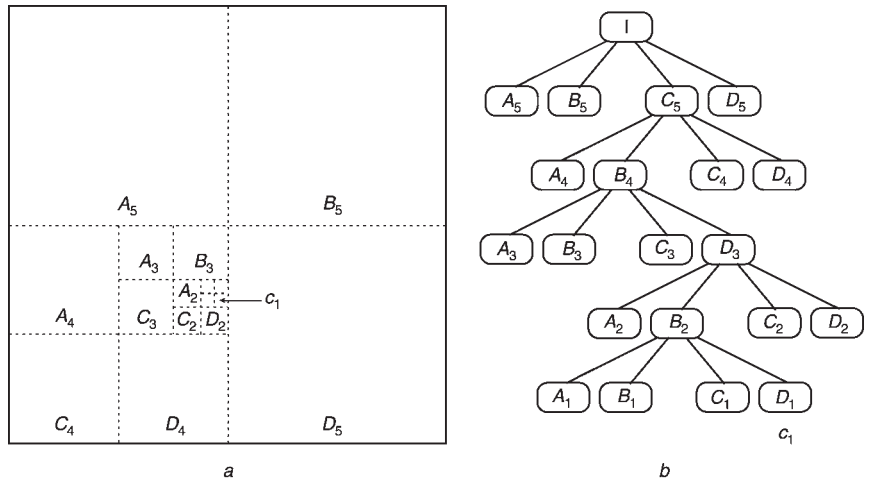


Fig. 3 *Example of a quadtree decomposition (a), and example of tree (b)*

Table 1: Summary of the main characteristics of implementation of the algorithms

Algorithm	Image transform	Quantisation	Significance information code	Alphabet	Symbols code	Reference value
EZW	wavelet	successive approximation of scalars	zerotrees	2, 3 or 4 symbols	Adaptive arithmetic coder	$T \leftarrow T/2$
SAWVQ	wavelet	successive approximation of vectors	zerotrees	3 or $m+1$ symbols	Adaptive arithmetic coder	$T \leftarrow T * \alpha$, $0.5 \leq \alpha < 1$
EZW/C	wavelet	successive approximation of scalars	statistical conditioning	2 symbols	Adaptive arithmetic coder	$T \leftarrow T/2$
SAWVQ/C	wavelet	successive approximation of vectors	statistical conditioning	2 or $m+1$ symbols	Adaptive arithmetic coder	$T \leftarrow T * \alpha$, $0.5 \leq \alpha < 1$
SPIHT	wavelet	successive approximation of scalars	lists structure	2 symbols	Adaptive arithmetic coder	$T \leftarrow T/2$
SPIHTVQ	wavelet	successive approximation of vectors	lists structure	2 or $m+1$ symbols	Adaptive arithmetic coder	$T \leftarrow T * \alpha$, $0.5 \leq \alpha < 1$
MGE	wavelet	successive approximation of scalars	Quadtree decomposition	2 symbols	Adaptive arithmetic coder	$T \leftarrow T/2$
MGEVQ	wavelet	successive approximation of vectors	Quadtree decomposition	2 or $m+1$ symbols	Adaptive arithmetic coder	$T \leftarrow T * \alpha$, $0.5 \leq \alpha < 1$

m is the cardinality of the dictionary

Another example is the MGE (multigrid embedding) [4] algorithm. It has a performance similar to that of the SPIHT algorithm. In MGE, the zerotree paradigm is substituted by a quadtree composition. When a square region contains a significant coefficient, it is split in four, and the process is repeated until all significant coefficients are localised. If, in a bitplane, all coefficients in a square region are insignificant, then the region is not split, and only a single bit is used to represent them. This is exemplified in Fig. 3, where a quadtree decomposition that localises element c_1 is shown along with the corresponding tree. More specifically, in this algorithm, like in the SPIHT, the initial reference value is given by 2^n, where n represents the most significant bitplane. Then, for a bitplane n_j, the region to be coded is tested for significant coefficients. If there is a significant coefficient, it is divided into four square subregions. This division stops when a predetermined region size is reached; after that, the signs of all coefficients in it are encoded. Then, the coefficients that have already been found significant in previous bitplanes are refined, and we proceed to the next bitplane, with the new reference value, 2^{n_j-1}. This process is repeated until a stop criterion is met. All the symbols generated are encoded using an adaptive arithmetic coder [12]. For more details see [4].

2.2 Encoders using vector bitplanes

In [8], a version of the EZW algorithm using vector bitplanes has been proposed, the SAWVQ (successive approximation wavelet vector quantisation) algorithm. This algorithm achieves an improvement of performance of almost 1 dB over the EZW algorithm for the 'Lena' 512×512 image at 0.5 bits/pixel. Using the same philosophy, we have proposed vector extensions of the algorithms, EZW/C, SPIHT and MGE, referred to as SAWVQ/C, SPIHTVQ and MGEVQ, respectively.

The main differences among scalar and vector versions are the following:

(*a*) the coding entities are vectors of coefficients instead of individual coefficients;
(*b*) the bitplanes are then vector bitplanes, represented by the set of indexes n_i for bitplane i (see (2)). The indexes n_i refer to vectors $\boldsymbol{u}_{n_i}$ drawn from a dictionary of orientation vectors;
(*c*) the initial reference value is α times the largest magnitude of the vectors of coefficients;
(*d*) The significance of a vector is determined by comparing its magnitude with the reference value.

Is important to note that:

(*a*) In the scalar case, each symbol of a bitplane has only two possibilities, '0' or '1'. In contrast, in the vector case, each symbol of a bitplane has as many possibilities as the cardinality of the dictionary C_N. Therefore, the number of bits per vector tends to be large in the vector case.
(*b*) On the other hand, since each vector represents N coefficients, the number of bits per coefficient may not increase; this will depend on the relative efficiency of the bitplane representation; in addition, the number of bits spent encoding the significance information will tend to be N times smaller, for there are N times less vectors than coefficients.

In Table 1 the main implementation characteristics of each of the algorithms analysed are summarised.

In the next Section we make a comparative analysis of the performance of the algorithms presented above. We first compare the rate $\times$ distortion performances of the algorithms in terms of peak signal to noise ratio (PSNR) obtained versus the bitrate. Second, we compare, for both the scalar and vector cases, the number of bits spent encoding the significance information with those spent encoding the non-zero coefficients.

3 Comparative analysis of the algorithms

For comparison among the several algorithms we use 'Lena', 'Barbara' and 'Boats' images, all with dimensions 512×512. They were encoded at three different rates: 0.2, 0.5 and 1 bit/pixel. The wavelet transform was based on the biorthogonal '*F*' filter bank defined in [13], with six decomposition levels. In the implementation using vector

Table 2: Scaling factors for dictionary and image

	Lena	Barbara	Boats
A2	$\alpha = 0.56$	$\alpha = 0.55$	$\alpha = 0.56$
D4	$\alpha = 0.58$	$\alpha = 0.60$	$\alpha = 0.58$
E8	$\alpha = 0.61$	$\alpha = 0.61$	$\alpha = 0.59$
Λ16	$\alpha = 0.63$	$\alpha = 0.62$	$\alpha = 0.64$

bitplanes, we used four different codebooks: in dimension 2, the regular hexagon of unit norm (A_2); in dimension 4, the first shell of the D_4 lattice; in dimension 8, the first shell of the E_8 lattice; and, in dimension 16, the first shell of the Λ_{16} lattice (for more information on these dictionaries see [14]).

An important parameter in vector bitplane decompositions is the α factor, since it has a profound effect on their performance. Since the aim of this work is study the behaviour of successive approximation vector quantisation when applied in bitplane-based wavelet coding algorithms, we opted to use the scaling factor α which gave the best results for each image and lattice.

The criterion used to choose the optimum scaling factor α was the following: for each image and for each lattice, the α value was varied in the range [0.5, 1] with increments of 0.01; the one yielding the best PSNR was chosen. The results obtained can be seen in Table 2.

In a practical case, this solution clearly presents problems due to the great computational complexity that would be required in order for the encoder to test for the optimum α for each image. However, as can be noted from Table 2, the variation of optimum α, for different lattices, is very small. Then, in practice, we can use single α for each lattice, with only a small penalty in performance. The results presented here, however, used the αs from Table 2 because using them gives a more accurate idea of the best possible performance that can be achieved by encoders using vector bitplanes.

3.1 Rate × distortion performance of the algorithms

In this section we make a comparison, in terms of rate × distortion performance, of the scalar and vector versions of the algorithms described in the previous section. Figures 4–6 show the performance for scalar and vector versions of the algorithms EZW, EZW/C, SPIHT and MGE for the images 'Lena', 'Barbara' and 'Boats', of dimensions 512 × 512. The vector versions use the first shell of the Λ16 lattice as codebook. We also present two Tables with PSNR values for all the images for the rates 0.2, 0.5 and 1 bit/pixel. The performances of scalar and vector versions are shown in Tables 3 and 4, respectively. In Figs. 4–6 we can observe that, in general, the encoders based on vector bitplanes tend to outperform the ones that are based on scalar bitplanes. Exceptions are the SPIHT and MGE algorithms for the 'Lena' image, where, for some rates, the scalar versions outperform the vector ones. In Fig. 4 we can also observe that the rate–distortion characteristic of the EZW algorithm exhibits a rather unexpected non-monotonic behaviour. For a detailed explanation, as well a solution for it, the reader should see [15].

From Table 4 we note that the best performances, in the vector case, are obtained with vectors of highest dimension, that is, using the codebook Λ_{16}. An important fact to be observed is that, provided that an arithmetic encoder is used, and for a given vector dimension, the differences in performance among the several vector versions of the algorithms are negligible.

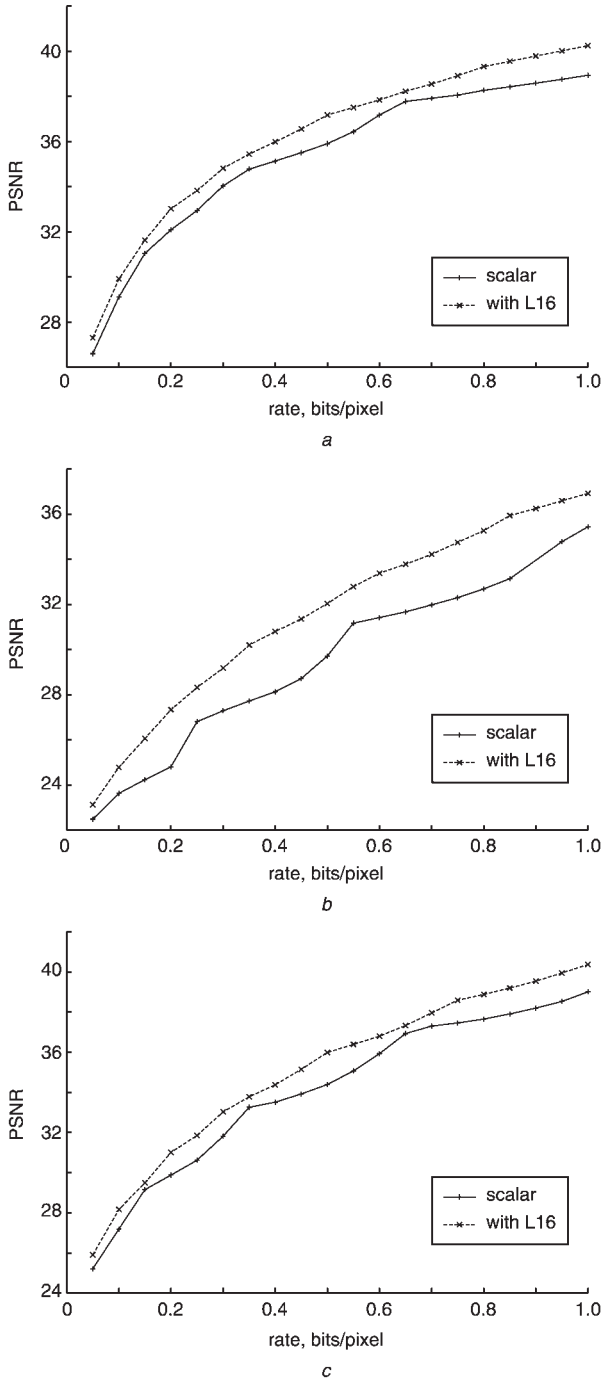


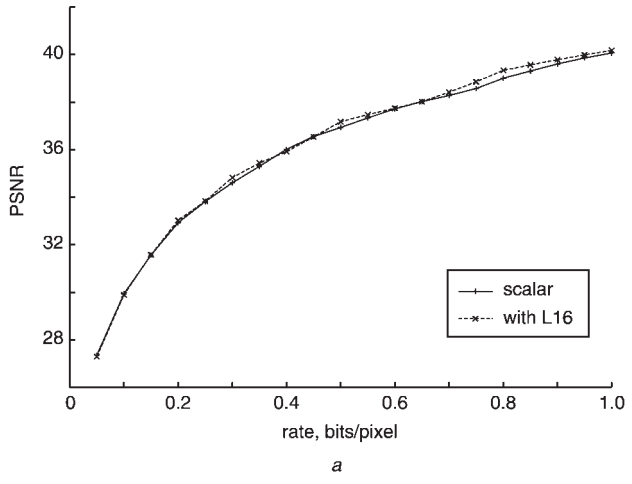
Fig. 4 *Performance of the algorithms EZW and SAWVQ with dictionary Λ16 as a function of the rate*

a 'Lena' image
b 'Barbara' image
c 'Boats' image

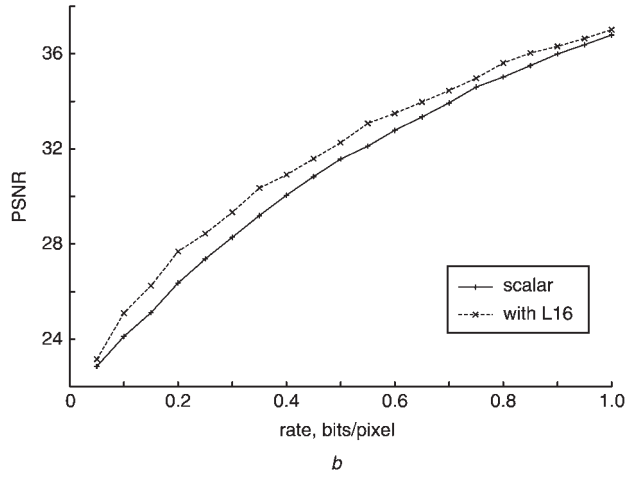
3.2 Comparative analysis of the number of bits spent in coding significance information and non-zero coefficients

Now we evaluate the behaviour of the algorithms in terms of the bits spent in the coding process. We do so by comparing the number of bits spent to encode the significance information and bitplanes of non-zero coefficients.

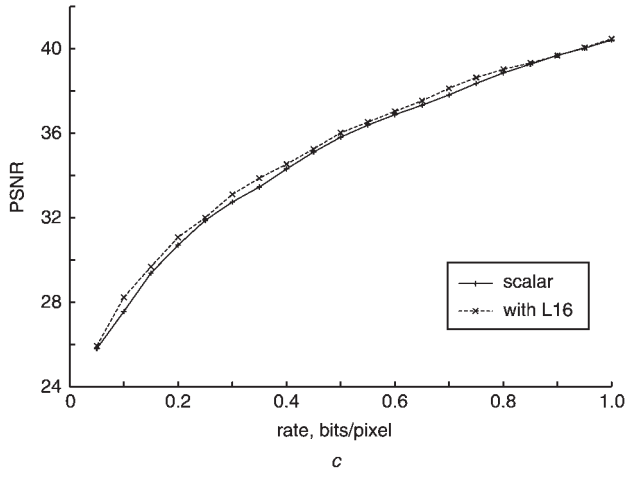
We carry out this analysis using the images 'Lena' and 'Boats' having dimensions 512 × 512. They were encoded using the same PSNR. The PSNRs used were the ones obtained for the EZW algorithm at a 0.5 bit/pixel. They were 35.91 dB for 'Lena' and 34.39 dB for 'Boats'. The corresponding bitrates are shown in Tables 5 and 6 for 'Lena' and 8 and 9 for 'Boats'.



a



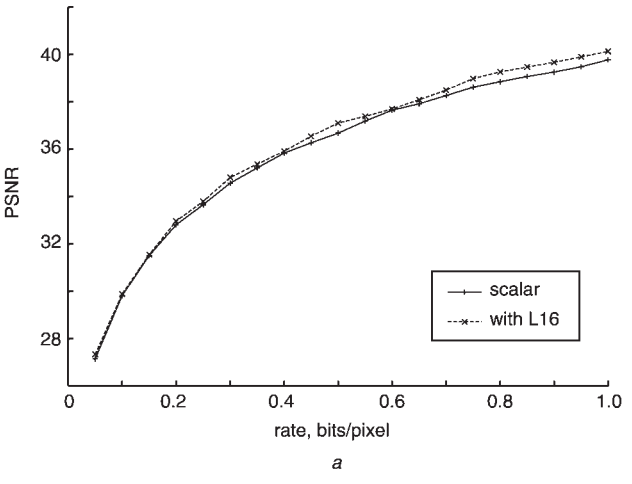
b



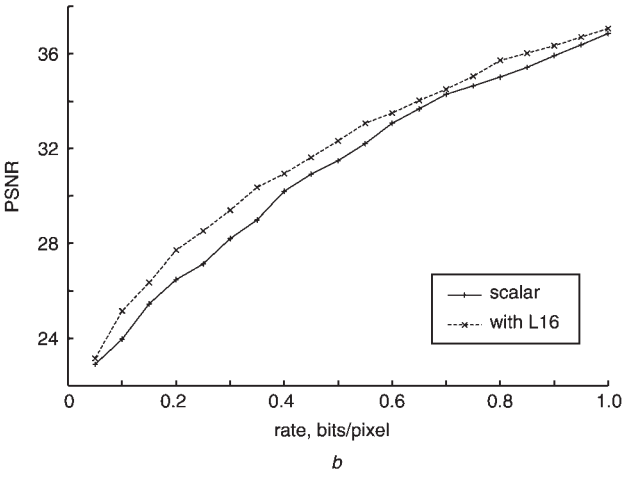
c

Fig. 5 *Performance of algorithms SPIHT and SPIHTVQ with dictionary Λ16 as a function of the rate*

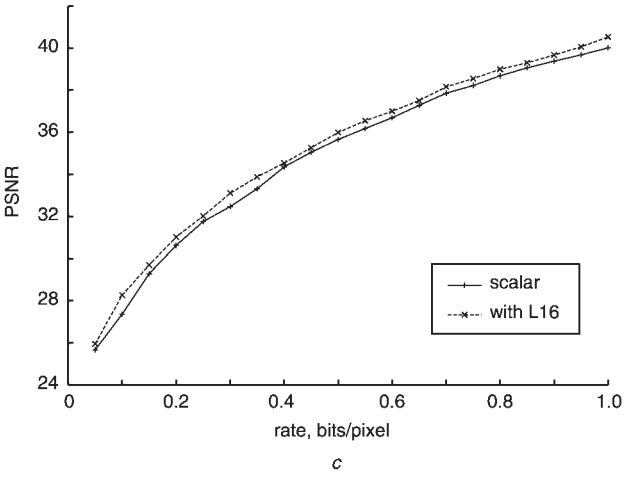
a 'Lena' image
b 'Barbara' image
c 'Boats' image



a



b



c

Fig. 6 *Performance of algorithms MGE and MGEVQ with dictionary Λ16 as a function of the rate*

a 'Lena' image
b 'Barbara' image
c 'Boats' image

Table 3: Comparison among PSNR (dB) for different algorithms using scalar bitplanes

Algorithm	Lena			Barbara			Boats		
	0.2	0.5	1	0.2	0.5	1	0.2	0.5	1
EZW	32.09	35.91	38.94	24.81	29.72	35.44	29.88	34.39	39.01
EZW/C	32.30	36.85	39.82	25.81	30.45	35.23	30.05	35.47	40.32
SPIHT	32.93	36.94	40.07	26.37	31.57	36.79	30.71	35.82	40.42
MGE	32.81	36.68	39.78	26.48	31.49	36.88	30.63	35.66	40.01

Table 4: Comparison among PSNR (dB) for the different algorithms using vector bitplanes

Algorithm		Lena			Barbara			Boats		
		0.2	0.5	1	0.2	0.5	1	0.2	0.5	1
SAWVQ	A2	31.89	36.29	39.10	26.14	29.94	34.52	29.70	34.73	38.59
	D4	32.52	36.60	39.96	26.44	30.77	35.85	30.73	35.04	40.06
	E8	32.79	36.74	39.92	26.67	31.75	36.16	30.40	35.49	39.89
	Λ16	33.03	37.18	40.25	27.34	32.04	36.92	31.01	35.99	40.37
SAWVQ/C	A2	32.03	36.38	39.31	26.26	30.48	35.18	29.96	34.88	38.93
	D4	32.62	36.74	40.08	26.64	31.24	36.14	30.77	35.21	40.13
	E8	32.79	36.84	40.00	26.95	31.90	36.57	30.42	35.56	40.00
	Λ16	33.01	37.19	40.24	27.44	32.38	37.32	31.01	36.01	40.44
SPIHTVQ	A2	32.26	36.49	39.58	26.41	30.88	35.46	30.18	35.02	39.41
	D4	32.72	36.55	40.02	26.83	31.25	36.48	30.78	35.33	40.15
	E8	32.84	36.77	40.17	27.22	32.05	36.62	30.69	35.70	40.00
	Λ16	33.03	37.18	40.18	27.68	32.26	37.01	31.08	36.03	40.47
MGEVQ	A2	32.09	36.32	39.22	26.44	30.87	35.43	30.05	34.86	39.13
	D4	32.62	36.56	39.93	26.82	31.30	36.41	30.73	35.24	39.93
	E8	32.81	36.76	39.99	27.29	32.05	36.65	30.64	35.6	39.95
	Λ16	32.97	37.11	40.13	27.71	32.33	37.07	31.03	35.99	40.53

Table 5: Bitrate for the scalar algorithms for 'Lena' image

Algorithm	Bitrate
EZW	0.5000
EZW/C	0.4435
SPIHT	0.3905
MGE	0.4080

Table 6: Bitrate for the vector algorithms for 'Lena' image

Algorithm	Bitrate			
	A2	D4	E8	Λ16
SAWVQ	0.4633	0.4410	0.4320	0.3930
SAWVQ/C	0.4465	0.4375	0.4300	0.3960
SPIHTVQ	0.4535	0.4210	0.4060	0.3990
MGEVQ	0.4615	0.4300	0.4110	0.3995

We make a direct comparison among the vector versions, as well as an individual comparison for each algorithm, among the scalar and vector versions for dimensions 2, 4, 8, 16.

In Figs. 7*a* and 8*a* we show comparisons of the bits spent by each of the algorithms to encode both significance information and the non-zero coefficients, using scalar bitplanes. In these Figures, the bits spent on coding just the sign of the coefficients are included above the bits spent encoding the non-zero coefficients (represented by the hatched areas). An exception is the EZW coder, where the sign of the non-zero coefficients is encoded together with the significance information. Observe that:

- The EZW algorithm has the worst performance of the four, spending more bits than the others to encode both the significance information and the bitplanes of the coefficients.
- The EZW/C algorithm is worse than SPIHT and MGE, both in the encoding of significance information and in the encoding of bitplanes. Thus, we can conclude that the prediction used by the SPIHT and MGE algorithms is more efficient than the statistical conditioning implemented in EZW/C.
- The SPIHT and MGE algorithms are very similar, although SPIHT has a slightly superior performance in the encoding of the significance map. One explanation for this behaviour lies in the fact that, unlike SPIHT, MGE does not exploit the interband dependency of the coefficients.
- Despite small differences in the encoding of bitplanes, the main performance differences among algorithms is in the form that the significance information is coded.

In Figs. 7*b* and 8*b* we make a similar comparison to the one of Figs. 7*a* and 8*a*, but now with the vector versions in dimension 16. It can be observed that:

- For the vector cases, unlike the scalar ones, the number of bits spent encoding the significance information is much smaller than the spent bits spent encoding non-zero vectors.
- The difference between the algorithms is negligible (this can be confirmed in Table 4). This is because, as seen previously, the main difference between these encoders is in the encoding of significance information, and the number of bits spent encoding significance information is N times smaller than in the scalar case (N is the vector dimension). See (i) and (ii) below.

In Figs. 7*c* to 7*f* and 8*c* to 8*f* each scalar algorithm is compared with its corresponding vector algorithm for the four different vector dimensions/codebooks. Note that, as the vector dimension increases, the number of bits spent coding the significance information decreases, while the number of bits spent coding the non-zero coefficients tends to increase. This behaviour can be explained by the following facts:

(i) In general, if one uses vectors of dimension N, one will spend N times fewer bits in coding the significance information. This is so because there are N times fewer vectors than coefficients.
(ii) The main difference between the different encoders is in the form the significance information encoded. Since, using vectors of dimension N, one spends N times fewer bits encoding the significance information, then the differences

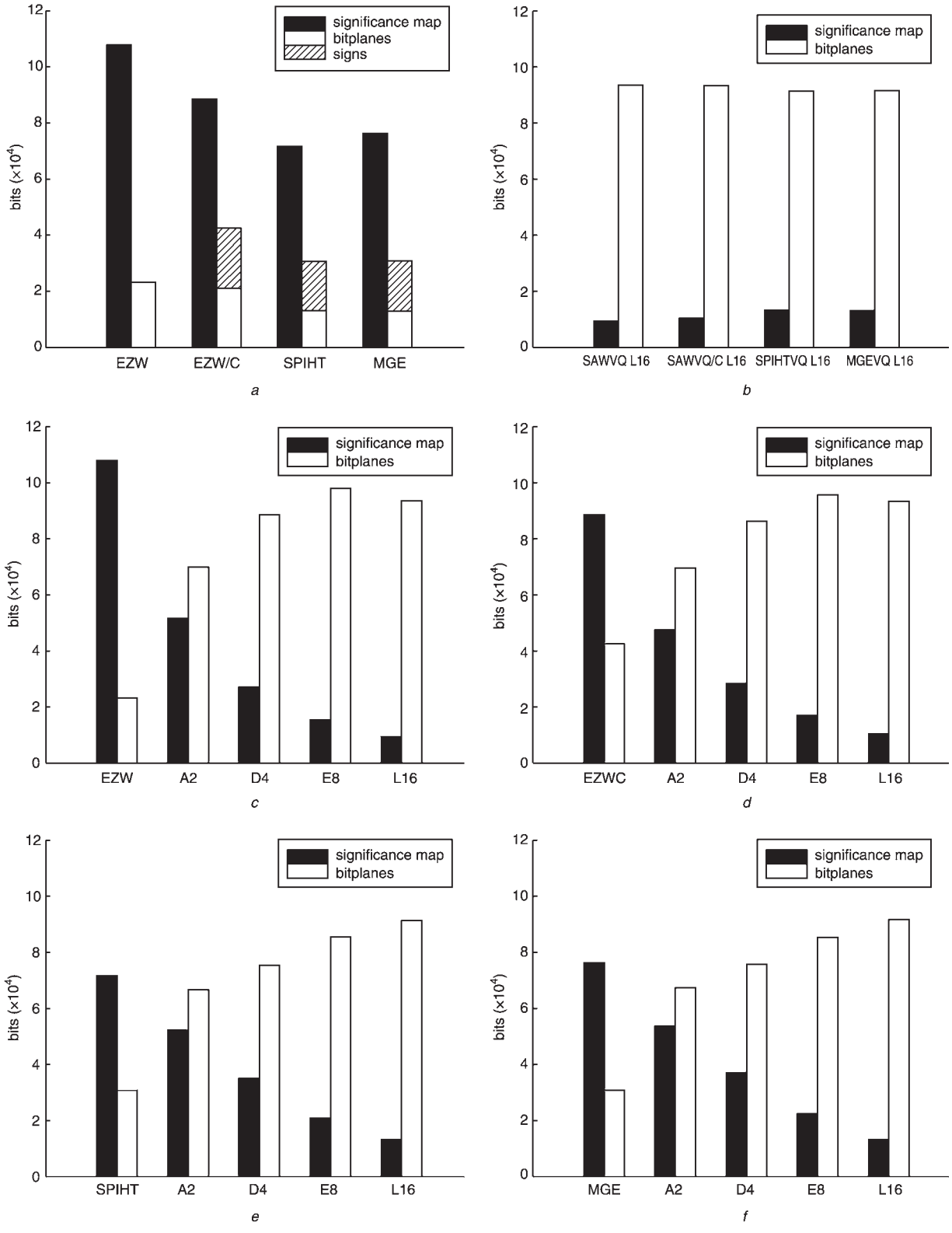


Fig. 7 *For 'Lena' at 35.91 dB, comparison of the bits spent in significance information and non-zero coefficient encoding*

a Scalar coders
b Vector coders
c EZW and SAWVQ
d EZW/C and SAWVQ/C
e SPIHT and SPIHTVQ
f MGE and MGEVQ

among the several algorithms will be divided by N, and will tend to become negligible.

(iii) In the scalar case, the magnitude corresponding to a bitplane is multiplied by 0.5 for each added bitplane (see (1)). On the other hand, for vector bitplanes, the magnitude corresponding to a bitplane is multiplied by $\alpha, 0.5 \leq \alpha < 1$, for each added bitplane (see (2)). Since, in all cases, the α used is larger than 0.5, then, in order for one to reach the same level of distortion, one needs in general more bitplanes than in the scalar case. In addition, since the value of α tends to increase with the vector dimension [10], the number of bits spent with the bitplanes increase with it. Also, one has

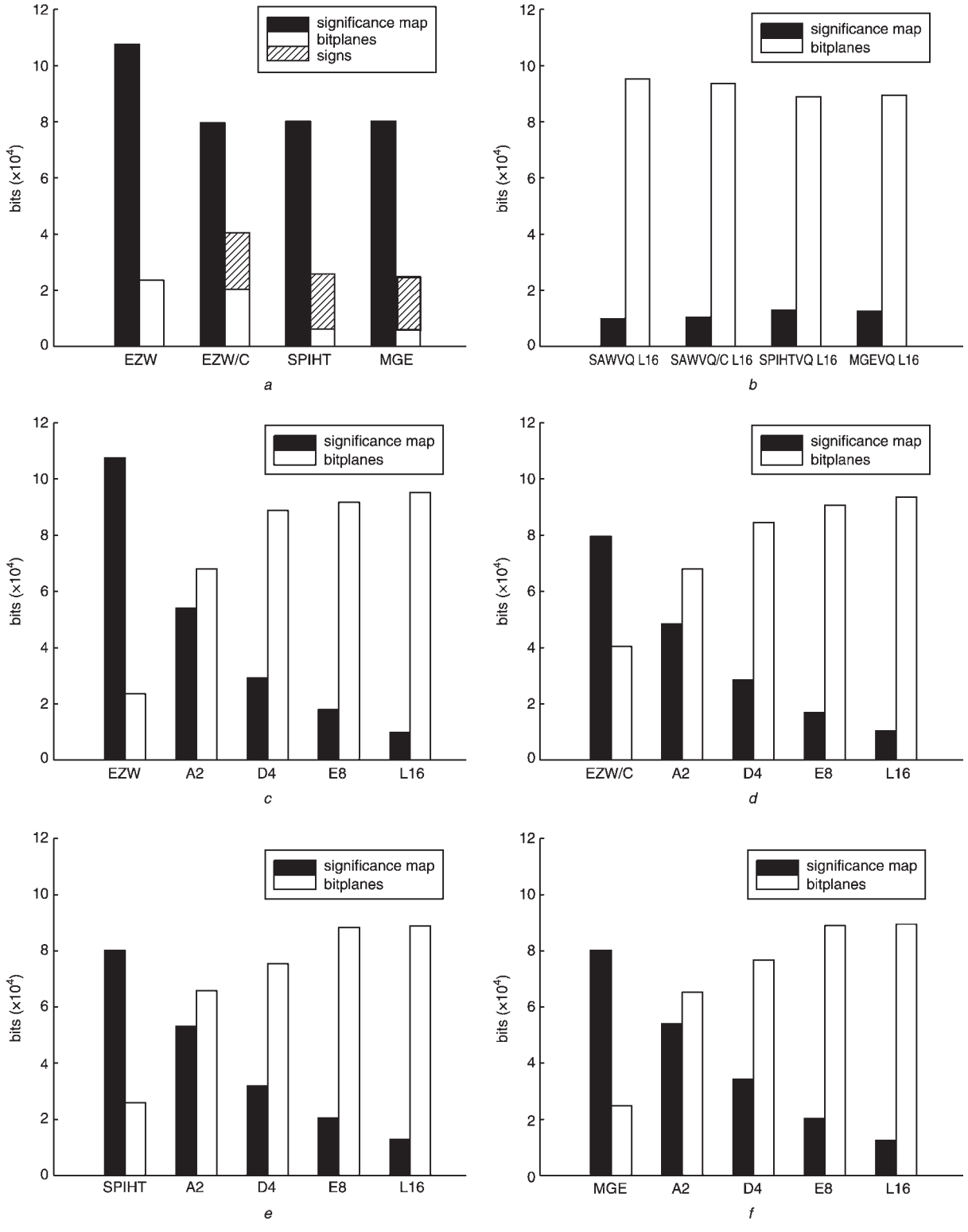


Fig. 8 *For 'Boats' at 34.39 dB, comparison of the bits spent in significance information and non-zero coefficient encoding*

a Scalar coders
b Vector coders
c EZW and SAWVQ
d EZW/C and SAWVQ/C
e SPIHT and SPIHTVQ
f MGE and MGEVQ

to consider the number of bits per coefficient used to encode each vector, as is shown in Table 7. We note that, despite the increase in α, the number of bits per coefficient decreases with vector dimension. Therefore, one effect tends to cancel the other. The net effect must be observed experimentally.

(iv) In the vector case, if a vector has, for example, just one significant coefficient, then all coefficients in the vector will be encoded as a non-zero vector; in the scalar case, this could correspond to just one non-zero coefficient, with the rest corresponding to significance information. Therefore, the vector bitplanes carry information that, in the scalar case, would correspond only to significance. This tends to

Table 7: Cardinality of each codebook, along with the number of bits/coefficient necessary to encode each vector

Codebook	Dimension	No. of vectors	No. of bits/coeff.
A2	2	6	$\log_2 6/2 = 1.29$
D4	4	24	$\log_2 24/4 = 1.15$
E8	8	240	$\log_2 240/8 = 0.99$
$\Lambda 16$	16	4320	$\log_2 4321/16 = 0.75$

Table 8: Bitrate for the scalar algorithms for 'Boats' image

Algorithm	Bitrate
EZW	0.5000
EZW/C	0.4580
SPIHT	0.4044
MGE	0.4010

Table 9: Bitrate for the vector algorithms for 'Boats' image

Algorithm	Bitrate			
	A2	D4	E8	$\Lambda 16$
SAWVQ	0.4650	0.4500	0.4184	0.4010
SAWVQ/C	0.4436	0.4310	0.4110	0.3970
SPIHTVQ	0.4535	0.4090	0.4152	0.3885
MGEVQ	0.4615	0.4230	0.4168	0.3890

increase the number of bits spent coding vector bitplanes and to decrease the number of bits spent coding significance information.

From the observations made above, one can conclude that there is very little to be gained in the performance of vector encoders by improvements in the algorithms themselves. This is so because such improvements would come mainly from the way the significance information is encoded, which would have negligible effect on the performance of vector encoders.

On the other hand, the compromise among vector dimension, number of bits/coefficient and value of α depends on the codebook used. One should note that we have opted for using regular lattices as codebooks mainly on a heuristic basis, because of their packing properties. Therefore, one can argue that in wavelet-based bitplane encoders, one area in which there is room for improvements is codebook design. In addition, it is expected that further developments of the theory of successive approximation vector quantisation [16] can lead to the design of better codebooks, which could tend to reduce the number of bits spent in encoding bitplanes, and therefore improved encoder performance.

A clear example of this potential can be seen in Figs. 7*c* and 8*d* where we note that an encoder based on the $\Lambda 16$ codebook uses less bits, both to encode the significance information and to encode the bitplanes themselves. This means that the vector bitplane decomposition using the $\Lambda 16$ codebook is better, in a rate distortion sense, than the one using the E8 codebook.

4 Conclusions

In this work, a comparative analysis has been made among several versions of both scalar and vector bitplane wavelet image encoders. They were compared both in terms of their rate $\times$ distortion performance and in terms of the number of bits spent encoding the significance information and the non-zero coefficients. It has been concluded that what differentiates the performance of the algorithms is the way that the significance information is encoded. Since the versions using vector bitplanes spend just a small percentage of bits coding significance information, then the differences in performance among the various vector algorithms are negligible. Also, the rate$z \times$ distortion performance of the encoders based on vector bitplanes is equivalent to that of the encoders based on scalar bitplanes. In fact, in most cases, it is slightly better. However, since most of the bits are spent encoding the vector bitplanes themselves, it is expected that only advances in the theory of vector bitplanes could lead to improvements in the performance of these coders.

5 References

1 Shapiro, J.M.: 'Embedded image coding using zerotrees on wavelet coefficients', *IEEE Trans. Signal Process*, 1993, **41**, pp. 3445–3462
2 Said, A., and Pearlman, W.: 'A new fast and efficient image codec based on set partitioning in hierarchical trees', *IEEE Trans. Circuits Syst. Video Technol.*, 1996, **6**, pp. 243–250
3 Andrew, J.: 'A simpler and efficient hierarchical image coder'. Proc. 1997 International Conference on Image Processing, Santa Barbara, CA, USA, 1997
4 Lan, T.-H., and Tewfik, A.H.: 'Multigrid embedding (MGE) image coding'. Proc. 1999 International Conference on Image Processing, Kobe, Japan, 1999
5 ISO/IEC JTC1/SC29/WG1, 'JPEG2000 Verification Model 8.6', 2000
6 Taubman, D.S., and Marcellin, M.W.: 'JPEG 2000: Image compression fundamentals, standards, and practice' (Kluwer Academic Publishers, 2001)
7 ISO/IEC JTC1/SC29/WG11, 'MPEG-4 Video Verification Model Version 8.0', July 1997
8 da Silva, E.A.B., Sampson, D.G., and Ghanbari, M.: 'A successive approximation vector quantizer for wavelet transform image coding', *IEEE Trans. Image Process.*, February 1996, **5**, (2), pp. 299–310
9 da Silva, E.A.B., and Craizer, M.: 'Generalized bit-planes for embedded codes'. Proc. 1998 IEEE International Conference on Image Processing, Chicago, IL, USA, October 1998
10 Craizer, M., da Silva, E.A.B., and Ramos, E.G.: 'Convergent algorithms for successive approximation vector quantization with applications to wavelet image compression', *IEE Proc. Vision, Image Signal Process.*, 1999, **146**, (3), pp. 159–164
11 Algazi, V.R., Robert, R., and Estes, J.: 'Analysis based coding of image transform and subband coefficients', *Proc. SPIE – Int. Soc. Opt. Eng.*, 1995, **2564**, pp. 11–21
12 Witten, I., Neal, R., and Cleary, J.G.: 'Arithmetic coding for data compression', *Commun. ACM*, 1987, **30**, pp. 520–540
13 da Silva, E.A.B., and Ghanbari, M.: 'On the performance of linear phase wavelet transforms in low bit rate image coding', *IEEE Trans. Image Process.*, 1996, **5**, (5), pp. 689–704
14 Conway, J.H., and Sloane, N.J.A.: 'Sphere packings, lattices and groups' (Springer-Verlag, New York, 1998)
15 Barreto, C.C., and Mendonça, G.V.: 'Enhanced zerotree wavelet transform image coding exploiting similarities inside subbands'. Proc. Int. Conf. on Image Processing, ICIP, Lausanne, Switzerland, 1996
16 Fonini, D.A., Jr., Craizer, M., and da Silva, E.A.B.: 'Quantized frame decompositions', in Cohen, A., Rabut, C., and Schumaker, L.L. (Eds.): 'Curve and surface fitting' (Vanderbilt University Press, Nashville, Saint-Malo, 1999), pp. 153–160